DESIGN, MODELLING AND SIMULATION OF 2 NOVEL 6 DOF HYBRID MACHINES

by Ahmed Asif Shaik

Student No.: 201293525

Email: ashaik@csir.co.za
Work Tel.: +27 12 841 3930
Cell No.: +27 72 430 0739

Supervisor: Prof. Glen Bright

Email: brightg@ukzn.co.za

In fulfilment of the academic requirements for the degree, Doctor of Philosophy (PhD) in Engineering at the School of Mechanical Engineering, University of KwaZulu-Natal.

Department of Mechanical Engineering
University of KwaZulu Natal
Durban, South Africa

12 October 2012

As the candidate's Supervisor I agree to the submission of this thesis.

Place: Durban

Prof. Glen Bright

Date: 12 October 2012

Declaration

I **Ahmed Asif Shaik (201293525)** declare that...

(i) The research reported in this thesis, except where otherwise indicated, is my original work.

(ii) This thesis has not been submitted for any degree or examination at any other university.

(iii) This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Place: Durban

Ahmed Asif Shaik (201293525)

Date: 12 October 2012

I. Acknowledgements

The work presented in this thesis was carried out under the supervision of Prof. Glen Bright at the School of Mechanical Engineering, University of KwaZulu Natal. I wish to thank Prof. Bright for his support and guidance.

Additionally I owe a great deal of gratitude to the following people and organisations:

- My parents and brother for their support and encouragement throughout my academic career.
- The CSIR (Council for Scientific and Industrial Research) and the University of KwaZulu Natal for all financial support received.
- Dr. Nkgatho Tlale, Riaan Coetzee, Peter Bosscha, for their support and advice.

II. Abstract

Industrial robot arms are an essential part of automated manufacturing, and perform tasks such as component assembly, welding, light machining, spray painting, etc. They are highly repeatable, can be calibrated to be sufficiently accurate and they eliminate human error. The serial robot architecture is by far the most ubiquitous in modern day manufacturing, as the technology is highly refined in its current state; the machine architecture provides great dexterity and it has a large useful workspace. This architecture however does have some problems, one of which is a large machine moving mass. The primary reason for this lies in the location of its motors and gearboxes. Due to the robot's significant inertia it utilizes a large amount of energy.

This thesis focused on the mechanical design, mathematical modelling and simulation of 2 robotic arm designs which had a hybrid nature. They were classified as hybrid due to the fact that their architectures departed from both the classic definitions of serial kinematics manipulators/machines (SKMs) and parallel kinematics manipulators/machines (PKMs). The primary design goal was to merge some of the advantages of both architectures, i.e. a large workspace to footprint ratio and high end-effector dexterity which was found in serial robots, combined with the low inertia of a parallel robot for improved dynamics. Serial and parallel robots were complementary, and these design goals could not co-exist in a single purist robot architecture. The designs had a full complement of 6 DOFs (degrees of freedom), 3 DOFs for spatial position of the wrist and 3 DOFs for orientation of that wrist. They also had a lower machine moving mass, a fact that was thought to improve speed and energy usage. A major contribution of this research PhD project was a comparative energy usage study, which was performed against the serial robot as a measure. This was done for both hybrid designs as well as another model which represented 2 existing patented designs. The purpose of that was to determine if lowering the machine moving mass would improve energy efficiency, and to determine which design was best.

III. Acronyms

BCGM	Base concentric geared mechanism
C	Cylindrical (2 DOF joint)
CAD	Computer aided design
COM/s	Centre of mass/Centres of mass
DMA	Decoupled motion architecture
DOF/s	Degree of freedom/Degrees of freedom
EOAT	End of arm tooling
H	Screw (1 DOF joint)
HKM	Hybrid kinematics machine
HM	Heavy motor
ISO	International Organization for Standardization
LM	Light motor
P	Prismatic (1 DOF joint)
PK	Parallel kinematic/s
PKFM	Parallel kinematic flexure manipulator/mechanism
PKM	Parallel kinematics machine/manipulator
PWM	Pulse width modulated
R	Rotational (joint)
RMS	Root mean square
S	Spherical (3 DOF joint)
SCE	Start, centre and end coordinates
SK	Serial kinematic/s
SKM	Serial kinematics machine/manipulator
U	Universal (2 DOF joint)

IV. Terms

Robot/Manipulator/Machine – These terms were used interchangeably throughout this thesis, and was meant to refer to an industrial robot.

End-effector – A tool attached at the end of the mobile platform or wrist. The purpose of the wrist was to position and orient the tool with regard to some coordinate system. The tool could be a gripping device, a welding torch, a spray painting head, orbital sanding tool, etc., anything that the user required.

Path/Trajectory – A continuous path that the end-effector tip must follow, combined with orientation information at each of those path points.

Degree/s of freedom (DOF/DOFs) – In robotics this refers to the motion of the end-effector. Typically it is the number of controllable axes (linear, rotary or screw), or the number of axes driven by some actuator (motor, pneumatic or hydraulic). The maximum number of DOFs a free object can have in space is 6, 3 for the spatial coordinates of its origin with regard to some base frame, and 3 angles which will define its orientation relative to that base frame. Some robotic systems have fewer than 6 DOFs, and focus only on the minimum end-effector DOFs required for a set of applications. Others have more than 6 DOFs, which gives the control system extra parameters to use when solving the robot's inverse kinematics for a particular pose. That helps with obstacle avoidance.

Actuators/Drives/Motors – Devices that convert electrical, hydraulic and pneumatic energy into robot motion. Modern day industrial robot actuators are fast, accurate AC servo drives.

Accuracy – The difference between the spatial coordinates and orientation that a robot was instructed to be at and the actual measured spatial position and orientation achieved.

Repeatability – A measure on the ability of a robot to repeatedly achieve a given position and orientation when given the same input angles or control signals. A repeatable robot may not be very accurate, and an accurate robot may not be repeatable.

Payload/Load bearing capacity – Refers to the maximum mass that can be carried by the end-effector. The maximum payload is the total mass carried by the robot at reduced speed while maintaining its rated accuracy. The nominal payload on the other hand is the mass that can be moved at maximum speed while maintaining rated accuracy. Additionally the size and shape of the payload affects these ratings.

Reach – The maximum radial distance the robot can position its end-effector tip from the robot's base origin.

Kinematics – Refers to the actual arrangement of joints/axes and rigid links in the robot. Common robot kinematics, include Cartesian, articulated, parallel and SCARA. It also refers to the study of motion in robotics without regard to the forces that cause it.

Resolution – The smallest increment in motion or distance that the robot end-effector or joint angles can make. It is dependent on the quality of sensors, and their ability to measure the smallest detectable motion on their axes (linear or rotary).

Servo controlled – Essentially it is feedback control where the driving signal is determined from the difference between the desired output position and a measured current position.

Work envelope/Work volume/Workspace – A boundary in 3D space that indicates the maximum reach of the robotic arm. Additional constraints could include minimum angles necessary for end-effector orientation.

Footprint (m^3) – Volume of space occupied by a robot in a given pose.

Isotropy – The isotropicity of a robotic manipulator is related to the condition number of its Jacobian matrix. A manipulator is completely isotropic if its Jacobian matrix is isotropic, i.e. the condition number of its Jacobian matrix is one.

Condition number – This number is associated with the linear equation $A \cdot x = b$, which provides a bound on the solution inaccuracy of x after an approximation. The condition number can be thought of as the rate at which the solution x changes for a change in b. If the condition number is large, a small error in b will cause a large error in x. If the condition number is small however, the error in x will not be much larger than the error in b. Problems

having low condition numbers are said to be well-conditioned, while those with high condition numbers are said to be ill-conditioned.

Decoupled motion architecture – A robot architecture where some of the primary motions such as translation along or rotation about an axis, are controlled by certain active joints, but not all.

V. Table of contents

VI. Index of figures

VII. Index of tables

1. Introduction

This research PhD study was undertaken from March 2008 to October 2012, the subject of which was in the field of articulated arm industrial robotics. The scientific contribution of the study focused on 2 concept 6 DOF arm designs, their mathematical modelling, as well as an analysis from an energy perspective.

1.1. Motivation for study

Industrial robotic manipulators could be split into 3 categories, these were serial, parallel and hybrid kinematics machines (SKMs, PKMs and HKMs respectively). A serial kinematics architecture was one in which each driven axis followed its predecessor in an open ended chain with rigid links that connected each pair of joint axes. Stated in a different way, each actuator was positioned at or close to the joint it controlled. A parallel kinematics architecture on the other hand fixed the location and arrangement of the actuators at the stationary robot base. A number of linkages connected the driven axes to the robot end-effector and formed closed kinematic chains.

The performance of SKMs with regard to speed, payload capacity, accuracy, etc., was heavily dependent on the component technology used in their construction. They required the highest resolution encoders for motion accuracy (possible external direct end-effector sensors for spatial position and orientation) and powerful motors for robot speed, payload carrying ability and stiffness. SKMs were used in a number of industries, but the most prevalent was in the automotive manufacturing sector. The reasons for their dominance was a mature, refined technology that had been rigorously tested, and was very versatile able to accomplish any programmable repetitive task. The actual design of the SKM by virtue of the SKM definition could not improve the robot's performance. The limitations of the serial robot lead to the development of specialized PKMs. Parallel mechanisms could be designed to be faster, carry larger payloads, and have greater accuracy, using the exact same underlying

technology. PKMs have had significant impact in 2 applications, the first was in the packaging of items where speed was paramount, and the second was in CNC machining where end-effector stiffness and accuracy were critical. PKMs themselves had severe disadvantages, and their attributes were generally complementary to SKMs. PKM architectures greatest strengths were speed, stiffness and accuracy whereas SKM architectures were characterized with a large useful workspace, a small footprint and greater dexterity.

Figure 1: Comparative illustrations of a PKM and SKM

Improving robotic tools used in manufacturing automation was one way to improve manufacturing processes. This could be achieved through merging advantages from both types of robot architectures. The core focus of this PhD project was an attempt to reach that goal with the design of a manipulator that could satisfy most of industry's demands, i.e. a robot that was versatile, had a large workspace (with minimal machine footprint), high speed, high accuracy and low energy consumption. Pure serial and parallel industrial robot architectures had disadvantages which could not be overcome on their own. Combining the advantages from both architectures in a hybrid machine was an option to achieve the desired objective.

1.2. Problem statement

The main problems with current SKM industrial robots (largest market share) were that they lacked speed and energy efficiency. These 2 problems could be overcome by a reduction in the moving mass of the robot. One way to do this was to move all the robot actuators to a fixed position at the robot base. Light weight mechanisms would then be used to transfer torque to the required/intended axis on the robot. The key questions addressed in this research study were:

- Could a 6 DOF robot be developed that had a large workspace, high dexterity and small footprint with all its actuators at a fixed location in space?
- Would that design be more energy efficient than an industrial serial robot?

1.3. Project aims

The aims and scientific contribution of this project were to:

- Design at least 1 novel 6 DOF robotic manipulator, that had a workspace comparable to that of the SKM with a similar footprint, but to fix all motors at 1 location like in a PKM to reduce the machine's moving mass.
- Design comprehensive simulated tests, and analyse energy usage to determine if that design was better than a serial robot.

1.4. Project objectives

The objectives of this project were to:

- Research various types of serial, parallel and hybrid kinematics machines.
- Research and design mechanisms that could be used in a novel hybrid manipulator that would give it a large workspace and minimal footprint, with high accuracy.
- Mathematically model the design for kinematic and dynamic simulations.

- Simulate kinematic motion (path following) in a CAD package.
- Design and construct a scaled model to verify functioning.
- Conduct a comparative energy usage analysis of the designs and validate the results.

1.5. Limitations

- **Design** – The end-effector did not form part of this project, i.e. grippers, welding heads, etc. The design was restricted to use rigid link mechanisms, although toothed belts and pulleys, or chains and sprockets could be used in some places. These were mentioned at the appropriate point in the mechanical design section.
- **Mathematical modelling** – The mathematical modelling was restricted to the kinematics, dynamics (which excludes payload) and estimated energy consumption of the designs. The payload would affect the joint torques for each design, which would then have an effect on the ratings (especially mass which was required for dynamics) of all motors. This fact was taken into account in the dynamic modelling by specifying the lightest and heaviest mass that each motor was expected to have (for the lightest and heaviest payloads respectively). This information only affected the dynamic modelling of the serial robot as it carried all its motors. Furthermore the kinematic models of all robots being compared were the same (thesis conceptual designs as well as existing serial and patented designs that will be mentioned), hence the torque contributions at each joint required for motion considering the payload only (negating the change in motor mass for different payloads) at each joint, would be the same. It was for these reasons that payload specifications was ignored.
- **Simulation** – The simulations only performed rigid body calculations, the bending and flexibility of components was not considered.
- **Prototyping** – Due to financial constraints a full scale prototype was not built. A miniature model was constructed and actuated with model helicopter servos.

Additionally all bearings and geared mechanisms were made as simple as possible to facilitate a rapid build.

- **Comparative studies** – These were performed on simplified, ideal mathematical models of the hybrid designs and a standard serial robot. Non-ideal effects, such as friction, etc., were ignored. The comparison was limited to an energy analysis only, and considered only the work done at the joints responsible for translation of the wrist centre. Strength of materials, shapes and components did not form part of the analyses.

1.6. Project specifications

Mechanical specifications:

- **Size** – The robot was a scaled version for the purpose of a kinematics and controls study. At full vertical stretch it fitted in a cylinder of radius 200mm by length 500mm.
- **Reachable workspace** – The workspace of the end-effector covered a hemisphere which had a 450mm radius.
- **Position accuracy and repeatability; robot speed and acceleration; payload carrying ability** – This scaled version's sole purpose was the study of 1 HKM design, its motion, kinematics and possible short comings. It was not intended to perform any functional operations. Measurements of accuracy, repeatability, speed, acceleration and payload carrying ability were of no value.

Simulation specifications:

- **Reachable workspace –** The workspace of the end-effector covered a hemisphere which had a radius of 2m.
- **Paths –** Simulations were carried out for straight line vertical, horizontal and radial paths of length 1.066m, which completed in times 5s and 0.25s (where the static and

dynamic response was dominant respectively). Simulations were also carried out for circular paths, in which each of the first 3 angles were varied individually. This was done for 2 configurations (where possible) of the mass moment about the joint controlled, which gave the best and worst case joint torque. These paths were also completed in 2 separate times, 40s and 2s. Joint 1 had a range of 180º, joint 2 a range of 90º and joint 3 a range of 180º. For additional information see tables 13 and 14 on pages 172 and 173 respectively.

Software and control specifications:

- **User interface** – A simple, uncluttered user interface which had high level user control. The user was able to control the model in either the forward or inverse kinematics sense. In the forward kinematics sense the user could select a specific joint for control. In the inverse kinematics sense, the user could load a specific path and watch the end-effector traverse that path (as it solved for control angles in the back ground).
- **Control** – The control system used feedback control, with sensors that monitored joint angles. Low level motor control was implemented using a micro-controller on the robot arm, whereas the high level path planning and kinematic solvers were on a separate higher performance computing platform.

1.7. Research publications

At the time of submission there were 4 publications (3 conference papers and 1 journal article) on the work presented in this PhD thesis, of which I was lead author. It is hereby acknowledged that the content of these publications were used in this thesis. The CARs&FOF 2011 conference paper was 1 of 14 papers selected from its proceedings, to appear in a special issue of the IJISTA journal. It had to be reworked, and expanded to comply with the journal's guidelines.

1. **CARs&FOF 2011**, 26th International Conference on CADCAM, Robotics and Factories

of the Future, 26-28 July 2011, Kuala Lumpur, Malaysia: "Novel 6 DOF hybrid machine design", A.A. Shaik and Dr. N.S. Tlale, Council for Scientific and Industrial Research (CSIR), Pretoria, South Africa; Prof. G. Bright, University of KwaZulu Natal, Durban, South Africa. Vol. 2, pp. 631-639. ISBN: 978-983-44947-3-5.

2. **ROBMECH 2011**, 4th Robotics and Mechatronics Conference of South Africa, 23-25 November 2011, Pretoria, South Africa: "6 DOF, low inertia, concept design for an industrial robotic arm", A.A. Shaik and Dr. N.S. Tlale, Council for Scientific and Industrial Research (CSIR), Pretoria, South Africa; Prof. G. Bright, University of KwaZulu Natal, Durban, South Africa. ISBN: 978-0-620-51897-0.

3. **M2VIP11**, 18th International Conference on Mechatronics and Machine Vision in Practice, 6-7 December 2011, Brisbane, Australia: "Novel light-weight 6 DOF robotic arm", A.A. Shaik and Dr. N.S. Tlale, Council for Scientific and Industrial Research (CSIR), Pretoria, South Africa; Prof. G. Bright, University of KwaZulu Natal, Durban, South Africa. ISSN: 1908-1162.

4. **IJISTA 2012**, International Journal of Intelligent Systems, Technologies and Applications: "A new hybrid machine design for a 6 DOF industrial robot arm", A.A. Shaik and Dr. N.S. Tlale, Council for Scientific and Industrial Research (CSIR), Pretoria, South Africa; Prof. G. Bright, University of KwaZulu Natal, Durban, South Africa. Vol. 11, Nos. 1-2, May 2012, pp. 63-80. ISSN (Online): 1740-8873; ISSN (Print): 1740-8865.

1.8. Thesis outline

This thesis was composed of 6 chapters. The chapters were a logical breakdown of the project into its major constituent parts and contributions, and is outlined below.

- ***Chapter 1, Introduction –*** Introduced the topic of the thesis, and listed project aims, objectives, specifications and publications.
- ***Chapter 2, Literature review –*** Presented a review of some serial, parallel and hybrid kinematics robots, and illustrated some of the more popular

designs. A detailed comparison of advantages and disadvantages between SKM and PKM architectures was presented.

- ***Chapter 3, Mechanical design and modelling –*** Presented the mechanical designs of 2 HKMs, as well as a detailed functional description of the machines. Both systems were reduced to a single simplified mathematical model so that the forward and inverse kinematics could be solved concisely, and was followed by those solutions. Machine dynamics were included, with 4 simplified robot dynamic models.
- ***Chapter 4, Scaled functional model –*** This chapter described the most important parts of a scaled functional model, including actuators, electronic control, software user interface and illustrations of a mechanical build.
- ***Chapter 5, Simulation results –*** Discussed simulations used to generate data for the linkage design of the first HKM, the forward and inverse kinematics, trajectory following and a major contribution of this thesis an energy usage comparison across 4 simplified dynamic models.
- ***Chapter 6, Conclusion –*** Concludes the thesis and summarized everything that was done in this research PhD study. It discussed the objectives, correlated them with what was achieved and indicated project success.
- ***References***
- ***Appendices*** – Contains all source code for simulations and data tables.

1.9. Chapter summary

This chapter served to introduce the reader to the project, which was the design of a hybrid manipulator/robot. It provided a brief motivation for the study, indicated the problem statement, listed the project aims, objectives, specifications and limitations.

Two novel 6 DOF machine design concepts were created. They both had a lighter machine moving mass than an equivalent serial robot, a large useful workspace and high dexterity. It was expected that they would have an improved dynamic performance over current serial

robot technologies, and a reduced energy consumption. This project was aligned to an international drive in which robot manufacturers and researchers from across the world aim to lower manipulator energy consumption. Achieving this goal would make their robots more environmentally friendly and competitive.

2. Literature review

2.1. General benefits and trends of industrial robots

Robotic manipulators are a central part of high technology industries. They are used for operations such as component assembly, pick and place, welding, cutting, spray painting, etc.

According to *IFR (Press release 2012)* [1] robot sales for 2011 was estimated at about 150,000 units, an increase of about 30% when compared to 2010. The market made significant gains since the severe downturn of 2009. In particular, 2011, saw an increase of demand from general industry, i.e. industries excluding the automotive sector.

World Robotics (2011) [2] said that 2010 sales amounted to US$ 5.7 billion, doubling revenue from the previous year. When including the cost of software, peripherals and systems engineering, the worldwide market value for robot systems for that year was estimated at US$17.5 billion. The total number of robots sold since their introduction at the end of the 1960's was about 2,142,000. The estimated number of operational robots worldwide at the end of 2010 was 1,035,000 and was expected to grow to about 1,308,000 by the end of 2014 at a rate of just under 6% year on year.

Industrial robots offered manufacturing companies the following main benefits as stated by *ABB Robotics (2007)* [3]:

1. Reduction in operating costs as they didn't require lighting, air conditioning, training, stringent health and safety checks and administration.
2. Improved product quality and consistency while reducing waste due to the robot's accuracy and repeatability.
3. Increased production output rates as they could be left running overnight and during weekends with little supervision.
4. Increased manufacturing flexibility as they could be programmed for a variety of

product processes and changed when required.

According to *World Robotics (2009)* [4], the main reasons for robot manufacturers and research institutes to invest in the improvement of industrial robotic systems were:

1. Decreasing product life cycles and shorter time-to-market.
2. Greater flexibility for production change over.
3. Reduce energy, material waste, operating and capital costs.
4. Improve product quality and consistency, and production output rates.

Zoppi (2004) [5] mentioned that economic factors and the availability of automation technology had created changes in the manufacturing paradigm, with 2 pertinent concerns. These were machine modularity and new concepts to help reduce time to market.

Meike and Ribickis (2011) [6] said that the cost of energy had increased in many industrial countries, and in some cases like Germany, electricity prices fluctuated with a factor of 2.3 over the last decade. Also many countries had committed to a reduction in CO^2 emission. These 2 reasons have prompted energy efficient measures in manufacturing. The automotive sector is a large consumer of energy with automation accounting for a sizeable portion. The authors stated that 15-28% of energy consumed by a vehicle during its life cycle was in the production phase, with electrical energy standing at 8%. More than 95% of the work in the body shop was completed by robotic applications. As such robotics have a high impact on manufacturing cost and total CO^2 emission during a vehicle's life. Even small improvements in efficiency could lead to significant savings on both fronts. Furthermore the power consumption of industrial robots is dynamic, and the motors are required to make quick starts, stop and direction changes in minimal time. For example the power consumption of a regular 6 axes robot with a 200kg payload could range from 0.5kW on stand-by to 20kW at peak, depending on the application, tools used, mass moved, trajectory, and a number of other variables. Due to the large market share, 6 axes robots with a payload range of 150kg-250kg were competitively priced, and were used in applications where they were over specified. An energy consumption comparison between 2 Kuka robots, the KR16 and KR210,

following an identical trajectory with a 16kg payload with identical cycle time revealed that on average the larger, heavier duty KR210 required 2.2 times more energy.

Robot manufacturers have devised ways to save energy and one method was recuperative motor braking. In this scheme when one motor is braking, the energy is buffered in capacitors through a DC-Bus, this energy is available through that bus to other motors as they accelerate. Also movement profiles were modified in correlation to cycle time. At lower speeds consumption drops, as well as running a robot at constant speed with decreased acceleration. Hence if there was time to spare during a production phase operation, subsequent tasks were completed slower. This conservative drive style had the added benefit of preserving the robot's mechanical components thereby increasing its life expectancy. Additionally energy could be saved during robot shut down, stand-by and trajectory optimization during active usage, through technical advancements such as asynchronous brake management and brake power adjustment. Sourced from *Meike and Ribickis (2011)* [6].

RobotWorx (2007) [7] added that minimal, efficient end of arm tooling (EOAT) would improve energy savings. The robot's structure could be made lighter using lighter tooling and could be moved more easily. As an example, modern welding guns have smaller transformers and cables, and use direct current control.

IFR (Whittman 2011) [8] stated that the future trends for robot manufacturers were to minimize environmental impact by improving robot energy efficiency, reducing waste, and adopting new environmentally friendly technologies. There were 3 basic ways to improve robotic energy efficiency, i.e. through advanced energy efficient drive technologies, consistent weight reduction through lightweight construction and optimized motion control. High-quality synchronous servo motors with permanent rare-earth magnets were the stand choice for robots. Some could achieve efficiencies in excess of 90% and act as generators during braking.

IFR (Staubli 2011) [9] said that the globe has been spurred to adopt green manufacturing techniques. In Europe there was a framework in place with tax incentives, for establishing

industrial energy management systems. International regulatory agencies such as the ISO had identified energy management in industrial sectors as a priority due to the potential to save energy and reduce CO^2 emissions. Furthermore the benefits of efficient energy management extended to larger business profits.

2.2. Classification of industrial robot architectures

There were 3 common types of industrial robots, the first being 'articulated' or 'jointed' arm and was the most dexterous and versatile. The kinematics resembled that of a human arm, more will be said about this later as it falls under a more general classification. The next was 4 DOF SCARA (selective compliance assembly robot arm) robots, which had 2 parallel rotational joints and a translational joint for spatial positioning. The last DOF rotated the wrist. Common applications were for pick and place, assembly and packaging. The third type was a 'gantry' robot otherwise known as a Cartesian robot, and was generally a large rectangular structure. It had 3 actuated prismatic joints usually at 90º to each other for spatial positioning, and additional DOFs (up to 3) at the movable platform for tool orientation. Unlike the previous 2 robots the machine footprint was larger than its workspace, but its structure allowed it to carry heavier payloads and exert/withstand greater forces. Gantry robots were used for materials handling, pick and place operations and in machine tools. These 3 common types of industrial robots fall under a single classification called serial manipulators. The remaining 2 classifications are parallel and hybrid manipulators.

Donelan (2007) [10] said that the architecture of a manipulator or rigid body mechanism could be described by means of a graph whose vertices/nodes were the components, and whose edges represented a joint between components. The graph could also be represented in the opposite manner with vertices/nodes indicating joints and edges as components. The graph did not fully specify the robot but represented partial topological (nature and DOFs of each joint) and geometric (size and placement of components) information.

The graphs for serial, parallel and hybrid classifications are illustrated in figure 2 on page 15. In the figure the nodes/vertices represented by circles indicate the robot's links/members

and the lines represent the robot's joints. The letters P and B refer to movable platform (or end-effector, which could also refer to tooling) and base respectively. The joints could be rotational (R), prismatic (P), screw (H), universal (U) or spherical (S). More importantly the joints represent axes, or structural DOFs. Links could be of 0 physical length if 2 axes intersect (e.g. 2 R joints at 90º), but the joints were not represented as co-incident circles on the graph.

A serial chain or graph was defined as one in which each node had at most 2 connections and the graph did not close in on itself at any point, to form a cycle/loop (either major or minor). It was open ended as shown in figure 2a, and was called a path. Manipulators with this architecture were called serial. Related to the serial graph was the tree, shown in figure 2e, and manipulators that had this architecture were grippers and robotic hands with fingers. As mentioned in *Donelan (2007)* [10].

A parallel kinematic manipulator had a platform and base which were connected via a number of independent serial chains or legs (at least 2), most but not necessarily all of which had to be actuated somewhere on the chain. Also some chains could have multiple actuators, and the manipulator's DOF usually equalled the number of actuators. The most common location for the actuator was at a stationary position on the base which reduced the inertia of that chain. The comprising serial arms could be identical and each chain could be symmetric. That was not necessary though and could have different DOFs each, as shown in figure 2b. From *Carricato and Parenti-Castelli (2004)* [11], and *Zoppi, et. al. (2004)* [12].

According to *Bonev (2002)* [13], an n-DOF fully-parallel mechanism was composed of n independent legs which connected the mobile platform to the base. Each of those legs was a serial kinematic chain which had only 1 motor to actuate, directly or indirectly, 1 of the joints. If more joints on each independent chain were actuated, *Donelan (2007)* [10] said that it is called hybrid-parallel.

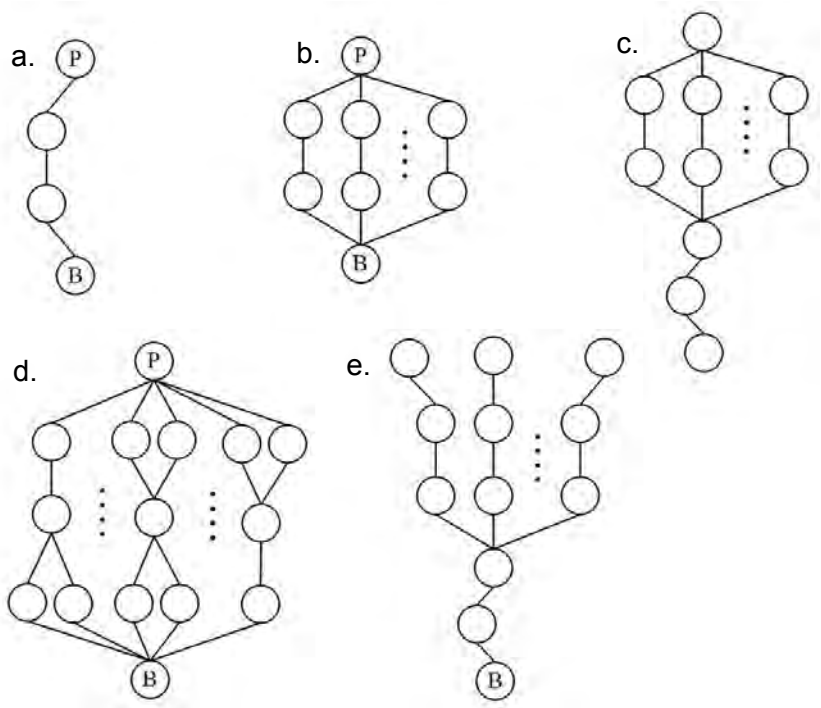


Figure 2: Graphical illustration of serial, parallel and hybrid chains or graphs - minor adaptation from Zoppi, et. al. (2004) [12]

a. Serial chain
b. Parallel chain
c. Hybrid chain
d. Inter connected parallel chain
e. Tree structure - related to serial

A hybrid mechanism was one that comprised both serial and parallel sections as shown in figure 2c. The parallel and serial sections could be anywhere on the chain, which was why the graph did not have the indicators P and B. *Bonev (2002)* [13] also stated that if the mechanism was not serial and was not fully-parallel as stated previously, then it was hybrid. This definition would then classify tree structures like fingered hands as hybrid.

If the serial chains in a PKM were interconnected by common members, then it was further classified as an Interconnected-Chains PKM (IC-PKMs). The interconnected chains were actually hybrid serial-parallel chains. A generic graph of an IC-PKM is shown in figure 2d. In such an IC-PKM, some actuators could drive more than 1 chain and some links connected to the end-effector were shared by more than 1 chain. In comparison with PKMs, IC-PKMs offered the advantages of simpler synthesis for adaptation to specific task requirements and higher overall mechanical stiffness. Their main drawback was a complex position analysis as

both the forward and inverse kinematics were expressed by systems of coupled non-linear algebraic equations. From *Zoppi, et. al. (2004)* [12].

2.3. Comparison of parallel and serial architectures

According to *Bruzzone, et. al. (2003)* [14] parallel robot designs in general had the advantages of increased stability and arm rigidity. Additionally there was high repeatability (due to reduced arm flexing) and the high stiffness of a closed-loop kinematics structure allowed it to exert strong forces in its workspace. [14]

Since the motors (which were responsible for most of the manipulators inertia) were positioned in a fixed arrangement at the stationary base, the speed of displacement of the end effector was greater, from *Tasora, et. al. (2005)* [15]

Her, et. al. (2003) [16] mentioned that PKMs had a higher precision and larger payload capacity and also stated that their speed, rigidity, and work space, depended on their configuration. Furthermore in general, most PKMs had a closed form inverse kinematics solution which was obtained through complex vector analysis, matrix inversions, and intense computations. The authors described an alternate computationally efficient method to overcome the problems encountered in solving the inverse kinematics problem. [16]

A PKM's forward kinematics could not be described in closed-form and was more difficult to solve than its inverse kinematics. Also calibration was difficult, as stated by *Tang, et. al. (2003)* [17].

The load of the end effector was distributed among many legs, and could be purely axial but that was dependent on the machine configuration, from *Rowe (2002)* [18]. *Kossowski and Notash (2001)* [19] said that PKM designs which carried loads primarily in tension/compression avoided the additional mass and volume of structural elements that carried loads in bending, and that components loaded in tension only did not need to be capable of withstanding buckling. *Kuen (2002)* [20] provided an example of a parallel manipulator that could move a payload of 600kg while only weighing 35kg.

Serial robots in general were optimized with little thought to the overall mass of the system,

rather mass was distributed to minimize moments produced about joints in order to reduce required motor torques, as related by *Kossowski and Notash (2001)* [19]. Furthermore massive structural components were needed to resist bending moments produced by the arm's weight and payload. PKMs by contrast could be designed to support larger loads for a given weight, with greater stiffness, by virtue of comprised closed loop geometries. [19]

Persson and Anderson (2003) [21] stated that each link in a serial robot's kinematic chain was required to support the masses of its link and motor, and the masses of all links and drive units that followed it. The inertia was therefore considerable when compared to PKMs, and that limited the capability of the robot in terms of its dynamic performance and acceleration. [21]

Martín, et. al. (2008) [22] also mentioned that the required torque on the actuators in a Cartesian SKM machine was four times higher than in some configurations of a PKM structure when moving the same mass.

Moon and Kota (2002) [23] said that each link in a serial machine had flexing errors which were additive, and resulted in a higher total end-of-arm flexing error as compared to PKMs. Generally all errors, such as manufacturing errors, gear backlash, hysteresis, etc. resulting from a serial structure were cumulative and amplified. PKM structures by contrast had the effect of averaging all errors. Also it was a challenge to achieve high accuracy using conventional joints, however large displacement compliant joints could improve the error averaging effect in PKMs and could lead to sub micron positioning accuracy. [23]

Articulated serial robots bend at high load and vibrate at high speed, so despite the advantage of a large workspace the positioning accuracy was poor, as stated by *Dasgupta (2008)* [24].

Briot and Bonev (2007) [25] attempted an answer to the question of whether parallel robots were more accurate than serial robots in their paper. They focused on the assertion of error averaging and claimed that it had not been addressed earlier. Acknowledging that they could not find a general way to tackle all PKM designs and all errors, they narrowed their view to the simplest serial robot and its parallel counterpart with 2 DOFs each having only input joint

errors on its desired workspace. They said that of all sources of positioning error (design errors, link compliance or flexibility errors, thermal expansion, etc.), joint sensor errors were the most significant. Their comparison was limited to draw any far reaching conclusions but it did indicate that PKMs were theoretically more accurate than serial robots when considering input joint errors only. Furthermore they also stated that a robot's mechanical design, manufacture and calibration affected its accuracy more than optimal kinematic design.

Bruyninckx (2005) [26] said that practical experiments with some prototypes of parallel structures indicated that their accuracy and stiffness were inferior to that of classical serial machines. The first reason was compliance in the ball screws of prismatic joints. The second was complexity in design and construction as well as the inclusion of many passive joints. Each of those joints had to have strict tolerances and be capable of withstanding high forces. Ball-in-socket joints, and possible leg collisions also imposed joint and motion limitation constraints. Lastly the calibration of the kinematic structure was difficult.

An article by *Burisch, et. al. (2005)* [27] compared the structures of serial and parallel micro SCARA robots, for high precision assembly. They stated that the serial structure had a resolution map which only achieved sub-μm accuracy in very limited areas of the available workspace, whereas the parallel structure achieved sub-μm accuracy over almost the complete workspace with a symmetrical resolution map.

According to *Hao and Merlet (2005)* [28] PKMs were less sensitive to temperature, had a lower energy consumption, a lower manufacturing cost and were more reliable. They offered good design variation which allowed designers to stretch their creativity and conceptualize machines with varying architectures, far more than they could do with serial topologies.

PKMs usually had a larger footprint to workspace ratio which was the most significant disadvantage. This was due to the location of motors and the resulting configuration. There were some exceptions like the Tricept, but other devices such as those based on the Hexapod PKM took up a large work volume. Also the performance of PKMs was dependent on their geometry and optimal design was necessary in their development. From [28] and *Lou, et. al. (2004)* [29].

Lou, et. al. (2004) [29] also stated that the complex input-output relationship, and the abundance of workspace singularities added to PKM disadvantages. Together with *Bonev and Gosselin (2000)* [30], they stated that the workspace was a basic and important index in the design of a parallel manipulator. It was often irregular-shaped due to complex kinematic structures. A design for maximum workspace could neither have a maximum regular workspace, nor good dexterity over its workspace [29]. The complete workspace for high DOF PKMs was a multi-dimensional entity which was impossible to visualize, and algorithms for various subsets of it were proposed [30].

PKMs dexterous workspace with desired orientation capability, was often a small part of the reachable workspace, from *Merlet (1995)* [31]. *Tol, et. al. (2002)* [32] added that work volumes contracted with tilt in platform angles, and that PKMs generally could not rotate continuously about the platform's vertical axis.

Furthermore *Chablat, et. al. (2004)* [33] said that the performances of maximum speed, force, accuracy, and stiffness at different points in the Cartesian workspace (as well as for different directions at those given points) varied significantly, and for machining applications those were serious drawbacks. In contrast the requirement was a PKM that had a regular shaped workspace with acceptable kinetostatic performances throughout. [33]

Dasgupta (2008) [24] said that additional limbs/kinematic chains could be added to a robot while maintaining the same movable platform DOF. Those robots were called redundant robots, since they used more inputs than were necessary. It was stated that by adding redundancy the workspace could be enhanced, and singularities as well as obstacles could be avoided. *Liu, et. al. (2003)* [34] also mentioned the use of redundant actuation to eliminate singularities and improve manipulator performance, since additional legs separated the constraint function of the movable platform from that of actuation. This advantage however came at a cost of increased mechanical complexity and the chance of leg interference. A redundant robot could navigate to a particular reachable point in an infinite number of ways, hence performance optimization and appropriate trajectory selection were necessary. [24], [34]

According to *Bruzzone, et. al. (2003)* [14] payload variations on the PKM end-effector drastically affected machine behaviour, due to the fact that the ratio between machine moving mass and payload was significantly lower than in SKMs. The complex kinematics and dynamic models also made control far more difficult than in serial machines.

Honegger (1998) [35] also said that the dynamics of PKMs were highly non-linear and coupled. The controller had to compensate for dynamic forces in order to minimize tracking errors.

Clavel et. al. (2002) [36] mentioned that several technological developments had been made on the design of robots capable of reaching 5 faces of a cube with superior dynamics, precision and cost performances. They had limited success due to their small rotational range of the end-effector, proximity of singular positions, and small workspace to footprint ratio.

The following paragraph was summarized from *Liu, et. al. (2004)* [37], *Ng, et. al. (2006)* [38] and *Laribi, et. al. (2007)* [39]. PKMs were well suited to the field of machine tools due to their advantages of high stiffness and high accuracy. Industrial machines based on a gantry style 5 DOF serial chain had weak points at the last two rotating joints (for tool orientation). Recent trends in industry were towards high speed machining, as well as a demand for higher accuracy and greater dexterity. For such reasons, more novel PKMs were researched and developed, the most popular having 6 DOF. Most however suffered from a number of problems.

Liu, et. al. (2002) [40] stated that most PKM research had been done on machines with 6 DOFs which had small useful workspaces, were riddled with design difficulties and had an extremely difficult forward kinematics problem. On the other hand the kinematics of parallel mechanisms with 2 and 3 DOFs could be described as closed forms. Moreover not all the singularities of a 6 DOF parallel mechanism could be found readily, but those were rapidly identified for PKMs with 2 and 3 DOFs. Those reasons made PKMs with less than 6 DOFs more attractive for industrial applications. [40]

The problems associated with large DOF PKMs prompted researchers to look at designs

having less than 6 DOF, as well as hybrid designs, according to *Carricato and Parenti-Castelli (2004)* [11], *Tang, et. al. (2003)* [17] and *Liu, et. al. (2004)* [37].

Zoppi, et. al. (2004) [12] said that it was easier to design a limited DOF special purpose parallel mechanism to fit the requirements of a particular task, than a general purpose 6 DOF parallel mechanism with a satisfying workspace.

With simpler architectures and cheaper construction due to fewer parts when compared to 6 DOF counterparts, they had greater advantages in applications where fewer than 6 DOFs were required as stated by *Carricato and Parenti-Castelli (2004)* [11].

Giberti, et. al. (2001) [41] also mentioned that most applications of pick-and-place did not require all 6 DOFs for the end-effector and 3 or 4 were sufficient. Specific PKMs could be developed to provide the required 3 or 4 DOFs for the end-effector, thus avoiding extra complexity of further actuators in higher DOF platforms.

Briot and Bonev (2008) [42] stated that there was an increasing trend in the use of 3 DOF planar parallel robots that were well designed, manufactured and calibrated for precision applications. They further stated that errors in the position and orientation of a parallel robot were due to several factors such as manufacturing errors (whose effect could be minimized through calibration), backlash (which could be eliminated through proper choice of mechanical components), compliance (which could be eliminated by using rigid structures at the expense of increased mass) and active-joint errors (a result of finite encoder resolution, sensor and control errors).

Carricato (2005) [43] on the other hand said that reducing the end-effector mobility had not generally resulted in simpler architectures and many designs appeared complex, possessing a large number of links and joints. Also they did not seem to be completely free from defects that affect higher DOF PKMs, such as singularities, complicated control, sizeable footprint, stiffness, link interference, etc. To overcome the above problems some researchers focused on either decoupling movement (input-output kinematic relations) and/or attaining full isotropy. *Fattah and Ghasemi (2002)* [44] stated that an isotropic manipulator was superior in kinematic accuracy and did not have singular configurations. The 'isotropicity' of a robotic

manipulator was related to the condition number of its Jacobian matrix. A manipulator was completely isotropic if its Jacobian matrix (for both forward and inverse kinematics) was isotropic, i.e. the condition number (see iv. used terms on page vii) of its Jacobian matrix was 1.

Zheng, et. al. (2004) [45] said that in spite of some advantages, lower chain PKMs with fewer DOF also had a reduced stiffness and speed.

As mentioned in *Cui and Zhu (2006)* [46] spot welding and painting were among the earliest applications for articulated robots, with a payload less than than 50 kg and repeatability in the range of 100μm. Applications of pick and place, and packaging had relatively lower requirements on repeatability and stiffness, with payloads varying from 1-500kg. The authors also stated that machining or material removal including deburring, grinding, milling and sawing, required high stiffness and that a robot's stiffness and accuracy determined the quality of the machined product. Hence those applications were not suited for articulated robots, but they were not ruled out entirely.

According to *Persson and Andersson (2003)* [21] PKMs could improve the precision in industrial applications, such as laser welding, water jet cutting, plasma cutting, gluing and assembly. Flexible PKMs could be used where high precision was required, i.e. in coordinate measuring machines, laser cutting and electronic component assembly. PKMs could also be used for automation in foundries to clean castings, grind, cut, drill and for rough machining, applications that required precision and stiffness.

Lastly *Bruyninckx (2005)* [26] stated that parallel and serial robots were complementary or dual, the weak points of serial designs were the strong points of parallel designs and vice versa.

Table 1: Summarized comparison of parallel and serial mechanisms

Advantage: X	**Parallel mechanisms**	**Serial mechanisms**
Higher stability and arm rigidity	X	
Greater accuracy and repeatability	X	
Higher stiffness	X	
Greater speed and acceleration	X	
Load distribution among actuators	X	
Dynamic behaviour immune to payload variations		X
Lower machine moving mass	X	
Lower energy consumption	X	
Lower manufacturing cost	X	
Lower sensitivity to environmental conditions	X	
Larger workspace to footprint ratio		X
Higher dexterity	X	
Geometry independent performance		X
Design variation	X	
Simpler control		X
Simpler forward kinematics		X
Simpler inverse kinematics	X	
Predictable dynamics		X
Simpler calibration		X

2.4. Industrial robot arm designs

As stated by *Donelan (2007)* [10] practical 6 DOF articulated industrial robots were wrist-partitioned or decoupled, which separated the position problem of the end-effector from its orientation. The end-effector reaches a desired orientation via a 3 DOF R-jointed spherical wrist, where all 3 rotational axes intersect at a point, the wrist or sphere centre C. *Hayes, et. al. (2002)* [47] said that the term spherical was used since all points on the wrist moved on concentric spheres when C was stationary. The translation problem was to move C (also the origin of the end-effector reference frame) within the reachable workspace. Euler angles could be used to describe the motion of the end-effector relative to the wrist centre. The first 3 DOFs determined the spatial position of the wrist centre C via a 3-joint arm which had 0, 1,

2 or 3 P-joints and a complementary set of R-joints. Partitioning in this manner simplified singularity analysis as discussed in section 2.7 singularities on page 51. [10], [47]

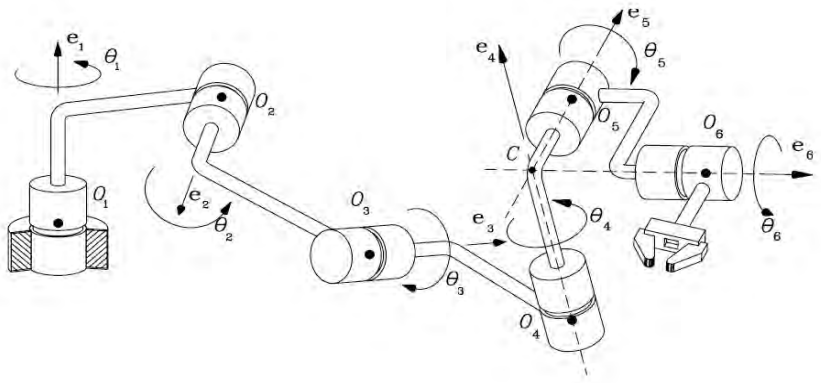
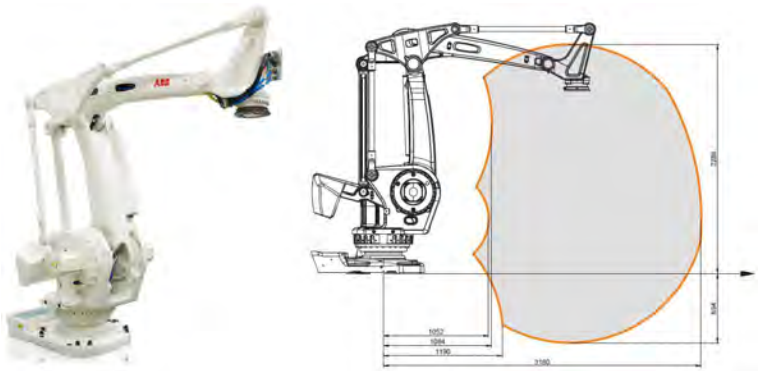


Figure 3: General wrist partitioned 6R structure
from Hayes, et. al. (2002) [47]

Figure 4: Various industrial robots offered by Kuka Robotics
from Kuka website

a.



b.

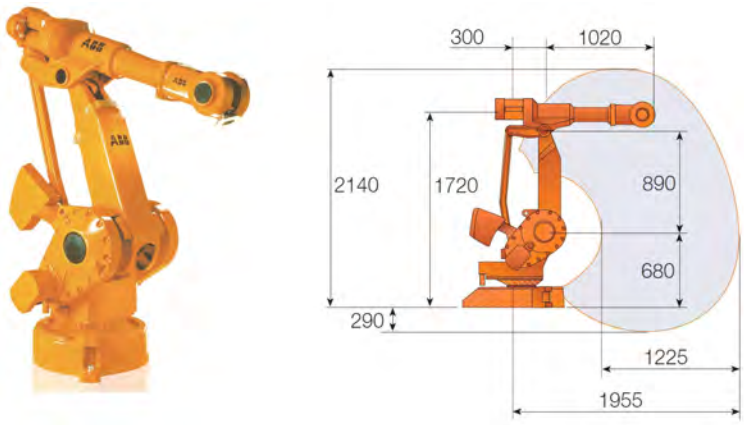


Figure 5: IRB 760 and IRB 6640
from ABB Robotics

a. IRB 760, from *ABB* [48] b. IRB 6640, from *ABB* [49]

Figure 4 illustrates a few industrial robots from Kuka Robotics. The top left design is a 4 DOF SCARA robot, while the bottom right is a 4 DOF palletising robot. The remaining designs are 6 DOF general purpose robots, which were wrist-partitioned and had different payload carrying abilities. The distal arm usually extends past the 'elbow' joint, with the wrist motors and the actual wrist on either side of that joint. The aim of that was to balance the mass at that joint to make it less taxing on the actuator. Additional measures to move dead mass closer to the base involve the use of torque transfer links as in the palletising robot. Figure 5 showing 2 of ABB's robots also uses this idea. ABB's palletising robot placed the motors responsible for 3 DOFs (following that for rotation about the vertical) close to the base (the first motor still had to rotate the collective mass). *Kuen (2002)* [20] stated that serial robots were massive due to the requirements on link rigidity and the serial propagation of high dynamic forces through the structure. These designs represent most of the variety available to serial robots.

2.5. Parallel industrial robot designs

It was stated in *Merlet (2000)* [50] that there were over 200 different architectures of PKMs at the time it was published, and *Martín, et. al. (2008)* [22] stated that few of those were practical architectures for industrial use. This section will show some of the more recent PKM designs and some of the most popular. The detail and number was limited for the sake of time and space. For more development news go to *http://www.parallemic.org*.

Carricato and Parenti-Castelli (2002) [51] said that PKMs could be divided into mixed, spherical and translational parallel manipulators (MPMs, SPMs and TPMs respectively).

The conventional Gough-Stewart manipulator or Hexapod consisted of a fixed base with universal joints that connected it to 6 prismatic legs (struts) which then connected to a platform via ball-and-socket joints (6-UPS). It was a very rigid kinematics linkage, more so than serial counterparts of the same size/weight. It also had a significantly higher payload rating, and the force-output to manipulator-weight ratio was generally an order of magnitude larger. Their main disadvantages were a very limited workspace, complex forward kinematics which had no closed form solution, complex control algorithms, coupling problems related to position and orientation movements, and precise spherical joints were difficult to manufacture at low cost. As related by *Ng, et. al. (2006)* [38], *Zhu, et. al. (2005)* [52] and others.

An earlier article by *Tlusty, et. al. (1999)* [53], whose central theme was the comparison of serial (gantry style) structures and parallel (Hexapod) structures as machine tools, asserted that some of the claims regarding rigidity were false. Depending on the structure of the Hexapod they suggested some reasons for the discrepancy between theory and fact, i.e. prismatic drive flexibility, flexibility of ball joints and unfavourable transmission of motion from the strut to the platform. The result was lower platform stiffness that varied strongly throughout the workspace.

A fully parallel 6 DOF milling machine called the Hexaglide was presented by *Honegger (1998)* [35] shown in figure 6. The kinematics of the structure was likened to that of the Hexapod, but instead of having variable length prismatic actuation on the legs, the mobile platform was guided into position by fixed length bars which were connected to 6 linear drives, on 3 railways (2 per rail).

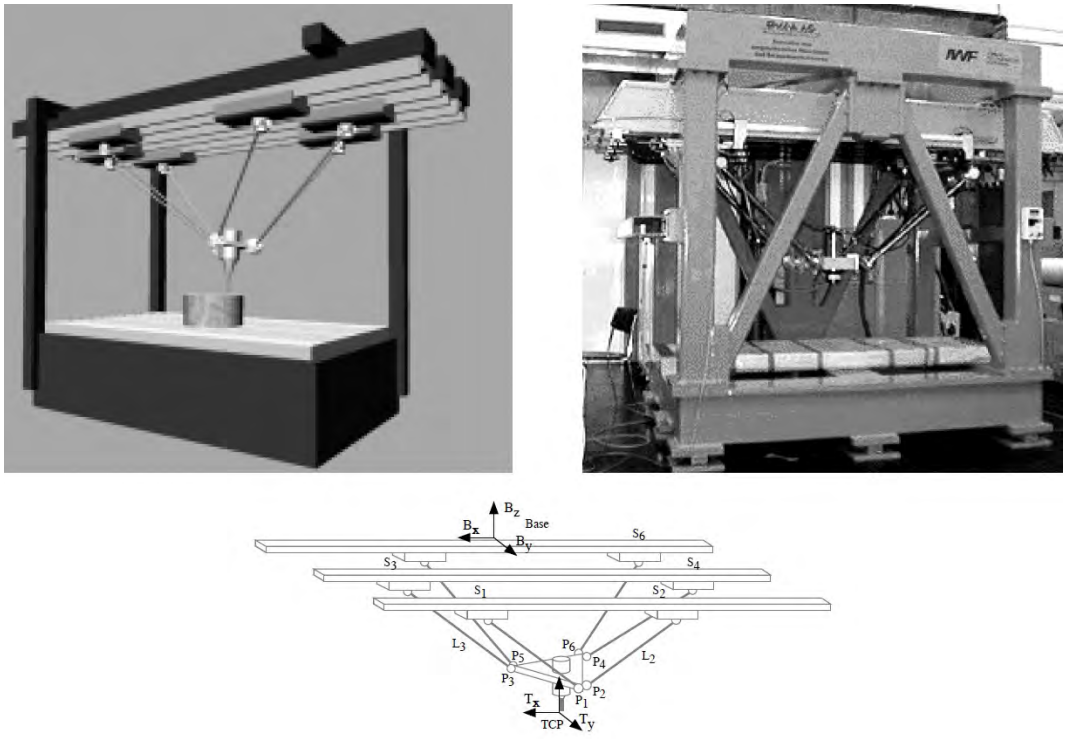


Figure 6: 6 DOF Hexaglide platform
from Honegger (1998) [35]

Hopkins and Williams (2002) [54] created a 6 DOF PKM by combining and modifying existing designs. The 6-PSU manipulator consisted of a platform and 6 legs, where each leg had an actuated prismatic joint, a spherical joint, a solid link and a universal joint which connected each leg to the movable platform. The spherical and universal joints were passive. The six prismatic joints moved vertically with respect to the base, and the authors claimed that it was a big improvement over standard Hexapod designs.

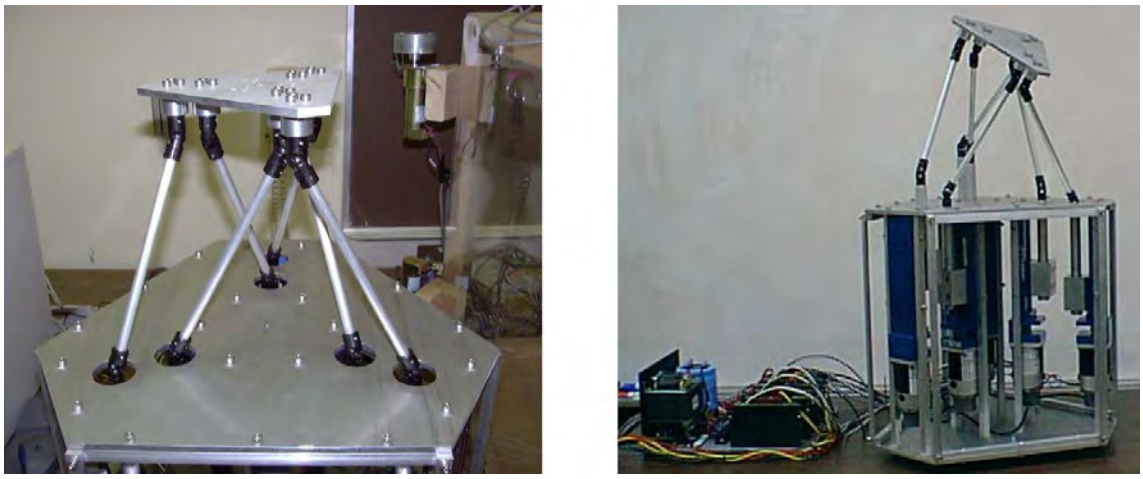


Figure 7: 6-PSU platform
from Hopkins and Williams (2002) [54]

The design by *Yang, et. al. (2004)* [55], shown in figure 8, of a 3-RPRS 6 DOF PKM had decoupled motion architecture (DMA). This meant that some of the primary motions such as translation along or rotation about an axis, were controlled by certain active joints but not all (complete de-coupling would mean that each actuator in the manipulator would be responsible for exactly 1 DOF about 1 axis). This was due to the fact that all joint axes, excluding the 3 spherical joints at the leg ends, were parallel to each other and perpendicular to the base plane. Translation in a horizontal plane and rotation about a vertical axis were controlled by the 3 active rotational joints. Translation in the vertical direction and rotation about horizontal axes were controlled by the 3 active prismatic joints. DMA made the kinematics of the machine simple (and to solve it a projection technique onto the horizontal and vertical planes was used) which simplified displacement, singularity and workspace analyses. There were at most 12 sets of solutions for the forward kinematics and 8 sets for the inverse kinematics. The authors also mentioned that conditions for the inverse, forward, and combined singularity configurations were found geometrically without formulating the Jacobian matrices, and that a computationally efficient geometric algorithm was used for the analysis of the constant-orientation workspace. With features such as a simple kinematic control model and a large cylindrical workspace with high stiffness in the vertical direction, it was suitable for light machining and assembly tasks.

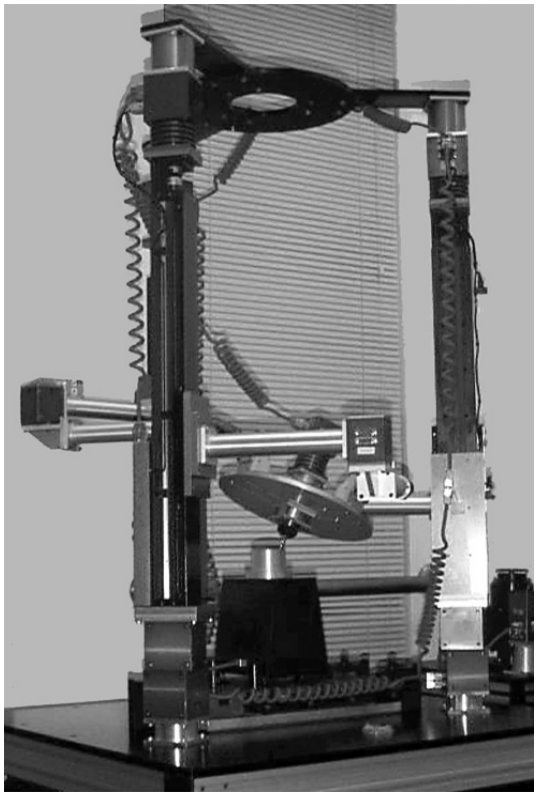
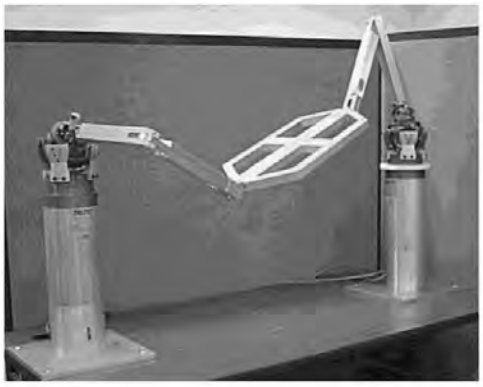


Figure 8: A 6 DOF PKM with DMA
from *Yang, et. al. (2004)* [55]

a.



b.

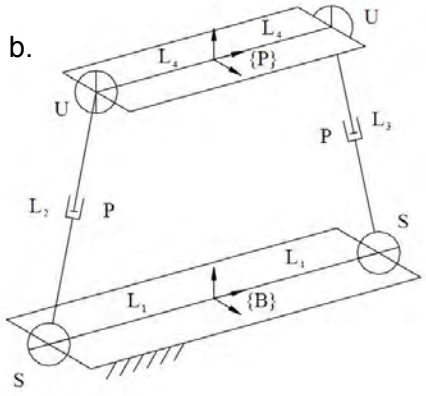


Figure 9: Spherically actuated platform manipulator
from Williams and Poling (2001) [56]

Williams and Poling (2001) [56] described the design of a 6 DOF 2 legged PKM. There were 2 SPU chains with active S-joints. This manipulator had the disadvantages of moment loading at the actuators (all loads were not axial), unlike in manipulators with prismatic

drives. Also the inverse kinematics was difficult to solve. The main purpose of this concept was to explore the use of spherical actuation. The general conclusion was that a 2 arm PKM with 6 DOF was not suitable for industry.

A 3 limbed PKM termed Selectively Actuated Parallel Manipulator (SA-PM) with DMA was designed by *Jin, et. al. (2004)* [57]. It had 3 equally spaced CPPRR chains. In this design the mobile platform could have 3 DOF spherical motion, 3 DOF pure translational motion, 3 DOF hybrid motion, or complete 6 DOF spatial motion (due to DMA). The type of motion selected depended on the active drive chosen for each leg, which could be rotary, linear or both (spiral) on a single axis. This could be achieved via an actuated cylindrical joint on each chain. The cylindrical actuator consisted of 2 DC servo motors, 2 timing belt-drives for power transmission and a ball-screw spline mechanism. The ball-screw mechanism achieved the stated 3 modes of motion on a single shaft by selective rotation of the ball-screw and spline nuts through coordinated motion of both motors.

A 5 DOF parallel robot HITA-STT combined 4 parallel axes and 1 rotational table. The table allowed 360º placement of the part for either real 5 face machining or turning applications. The robot had a low machine moving mass and thus reached high accelerations, and due to the architectural configuration achieved a large work volume. The kinematics structure also maintained a stiff configuration of the support bars over the whole rotational range of the spindle. Excluding the ball-and-socket joints mounted on the spindle, the rotation of all other ball-and-socket joints did not exceed +/-15 degrees. From *Clavel et. al. (2002)* [36] with an illustration on figure 11.

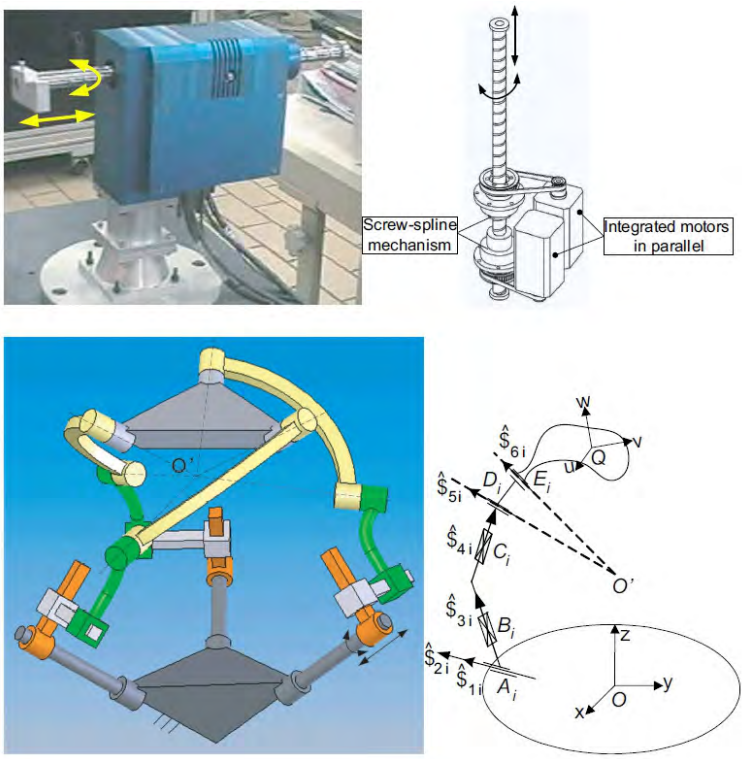


Figure 10: Selectively actuated 6 DOF 3-CPPRR manipulator from Jin, et. al. (2004) [57]

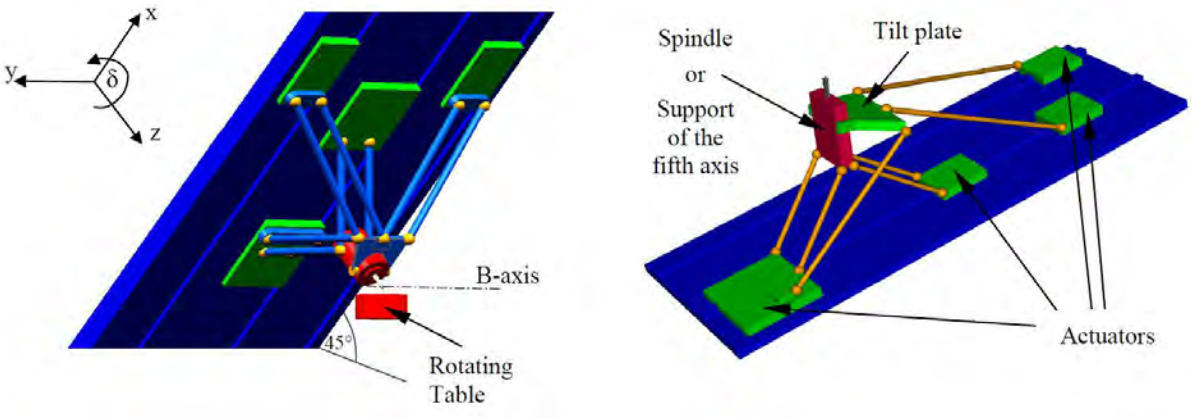


Figure 11: HITA-STT, a 5 DOF PKM from Clavel et. al. (2002) [36]

An important class of PKMs were 3 DOF tripods which had platforms connected to the fixed base by 3 legs that could move with pure spherical, translational or hybrid spherical-translational motion. *Laribi, et. al. (2007)* [39] indicated the trend to focus on 3 DOF PKMs in

which the end-effector's motion was restricted to translation along the 3 Cartesian axes with respect to the base. The DELTA robot was an example of such and consisted of a moving platform connected to a fixed base via 3 parallel kinematic chains. Each chain contained an actuated rotational joint at the base platform. Parallelograms formed by links and universal or spherical joints on the lower arms of each chain, restrained the orientation of the movable platform with respect to the base. The smaller mass and higher stiffness of the arms, were needed to achieve a shorter cycle time with reduced actuator power and higher accuracy. A fourth leg (UPU) could be added, to transmit rotary motion from the base to an end-effector mounted on the mobile platform. This robot ushered in a new era in manufacturing with machines that could reach extremely high dynamic performances. *Zhu, et. al. (2005)* [52] said that the DELTA robot achieved accelerations of up to $50 m.s^{-2}$ in experimental environments and $12 m.s^{-2}$ in industrial applications, which made it suitable for pick-and-place operations of light objects (up to 150 ppm as compared to 60-65 ppm for a SCARA). More recently it was adapted for 3 or 5-axis machining, from *Laribi, et. al. (2007)* [39]. A model of the DELTA robot is shown on figure 12. To improve its accuracy *Deblaise, et. al. (2004)* [58] presented a geometric calibration method, which took into account the elasticity of links, one source of end-effector placement error. The Delta design has been used in industry for several years and ABB Robotics sells one such version, the FlexPicker.

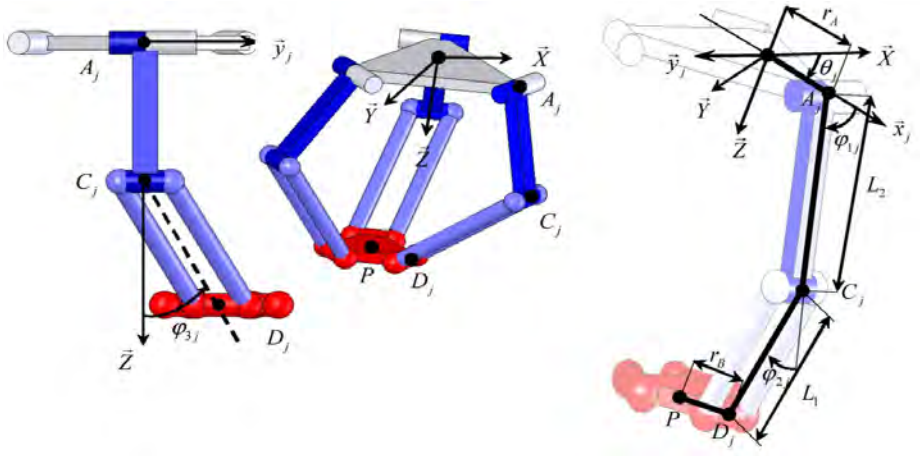


Figure 12: The Delta robot
from Laribi, et. al. (2007) [39]

Zhu, et. al. (2005) [52] designed the TAU robot shown in figure 13. It was an unsymmetrical PKM which had 3 actuators mounted on the fixed base, arranged in a line (I-structure TAU). TAU was similar to the DELTA in that both had 3 actuators mounted on the fixed base with each actuator driving 1 proximal arm. There were 6 passive distal arms that connected the proximal arms to the moving platform. The major difference was that in the DELTA design 2 passive distal arms were connected to 1 proximal arm in a 2-2-2 form whereas in the TAU design, it was a 3-2-1 form. The 3 distal arms connected to the first proximal arm were parallel to each other and had equal length. The 2 distal arms connected to the second proximal arm were also parallel to each other and had equal length. The last distal arm was connected to the last proximal arm. All proximal to distal arm connections were via U-joints, and all distal arm to movable platform connections were through S-joints. The TAU design had the advantages of a smaller footprint, larger singularity free workspace (significantly larger than most PKMs, due to the collinear vertical axes), a higher stiffness and reduced energy consumption. Also both the forward and inverse kinematics had closed form solutions, which made control easier and it had better performance with regard to force control.

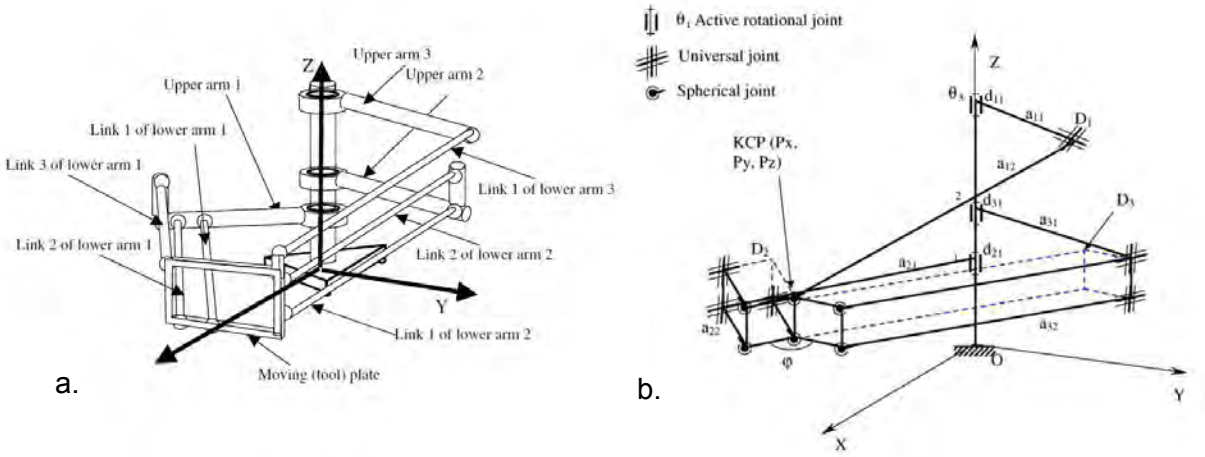


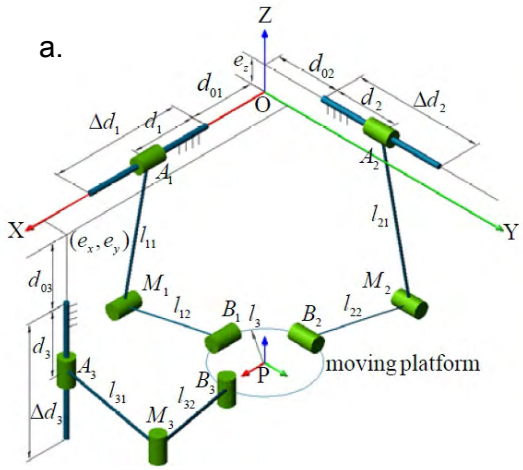
Figure 13: TAU robot design
from Zhu, et. al. (2005) [52]

a. One configuration of TAU b. Kinematic simplification

A 3-PRRR Cartesian Parallel Manipulator (CPM) was developed by *Kim (2005)* [59] and behaved like a conventional X-Y-Z Cartesian robot. This was due to the perpendicular

arrangement of the 3 active prismatic joints, which also lead to a 1-to-1 correspondence between input and output displacements, velocities, and forces. Furthermore the Jacobian matrix was always isotropic over the entire workspace. The 3 rotational joints and prismatic axis in each leg were parallel to each other, but perpendicular to those in other legs. A simple set of linear equations could be derived for the forward and inverse kinematics. A low machine moving mass made this manipulator suitable for applications that required high speed and accuracy.

a.



b.

Figure 14: Cartesian PKM
from Kim (2005) [59]
a. Kinematic model b. Prototype Cartesian PKM

Another translational PKM with prismatic actuators, classified as 3-CRR was developed by *Kong and Gosselin (2002)* [60] shown in figure 15. The C and R joint axes on each leg were parallel, and the translational DOF of each C-joint was driven. The driven axes were concurrent at a point. The authors detailed the geometric condition required for manipulator isotropy which then gave it a number of advantages, the first being simple forward and inverse kinematics. They also said that the Jacobian matrix of this design was constant, which implied that its inverse could be pre-calculated and done just once for multiple calculations of the forward kinematics and velocity analyses. Also this design did not have any rotation or uncertainty singularities.

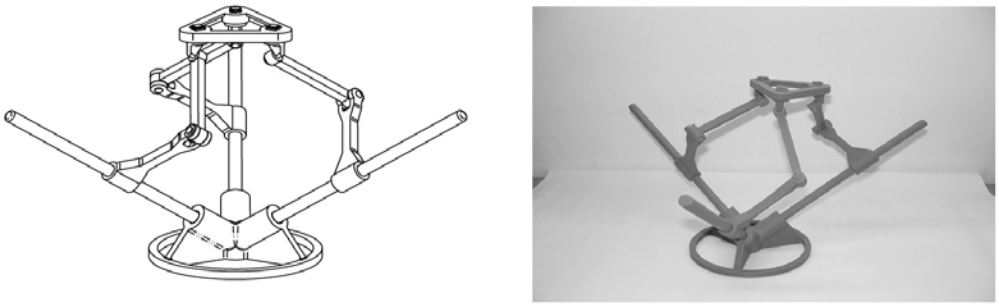


Figure 15: 3 DOF 3-CRR translational PKM
from Kong and Gosselin (2002) [60]

Giberti, et. al. (2001) [41] developed TORX, a 3-PUU 3 DOF translational PKM, with an actuated prismatic joint. The pure translational motion of the end-effector was realized by driving 3 fixed length legs along linear guides on the truss, instead of changing the length of those legs. The pneumatic linear actuators and their heavy prismatic guides were fixed on the truss, which lowered moving masses and simplified design. See figure 16. *Righettini, et. al. (2002)* [61] modified the TORX design slightly and created a 3-PSS structure, with spatial parallelograms that maintained a constant orientation of the movable platform. See figure 17.

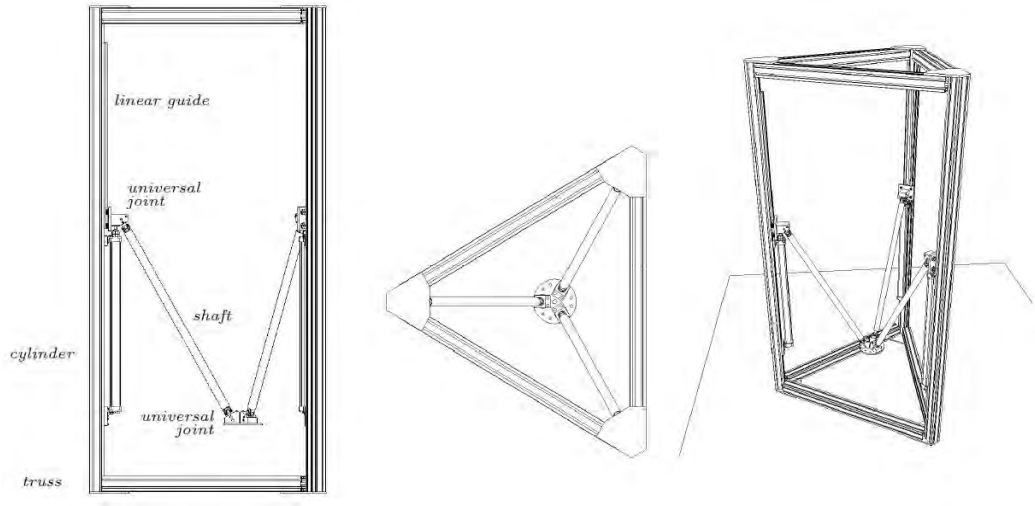


Figure 16: TORX, a 3 DOF translational PKM
from Giberti, et. al. (2001) [41]

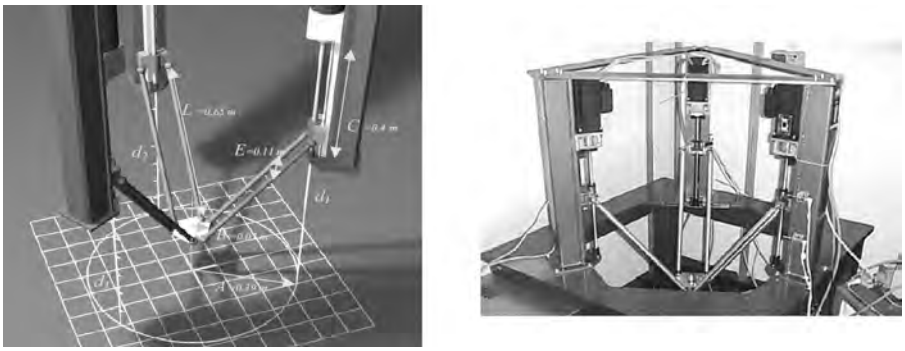


Figure 17: 3-PSS platform using spatial parallelograms
from Righettini, et. al. (2002) [61]

A 3-PUU compliant PKM was developed by *Xu, et. al. (2006)* [62] shown in figure 18, which had a micro-workspace for micro-manipulation and positioning. It was a major challenge to achieve ultra-high precision motion in PKMs using conventional joints, which suffer from manufacturing errors, backlash and hysteresis. Compliant mechanisms such as flexure hinges, could be used in PKMs for precision applications due to their outstanding characteristics in terms of vacuum compatibility, zero backlash, zero error accumulation, zero non-linear friction, and ease of manufacture. Applications for these flexure PKMs would be in the fields of optics, precision machine tools, and micro component fabrication.

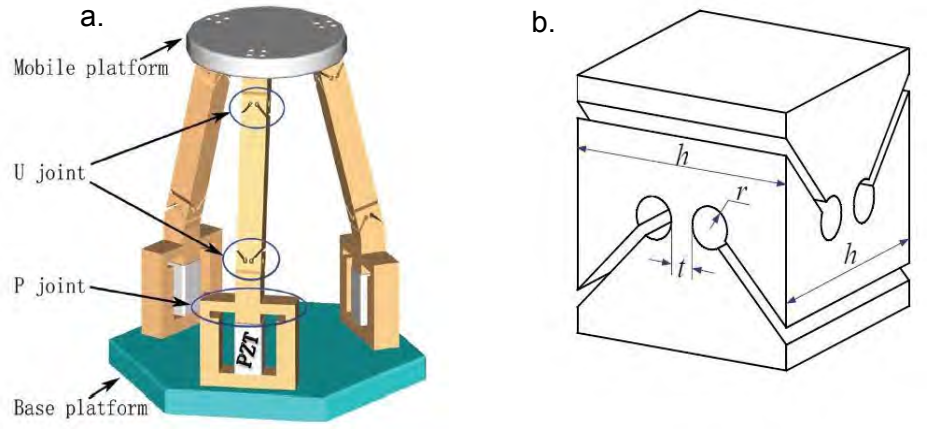


Figure 18: A compliant 3-PUU PKM
from Xu, et. al. (2006) [62]

a. CAD Model b. Compliant universal joint

Li and Xu (2006) [63] developed a 3-PRC PKM which achieved 3 translational DOFs using actuated prismatic joints. The authors said that the design used fewer links and joints than expected, and had a simpler structure than most existing translational platforms which lead to a reduction in cost as well as complexity of the device.

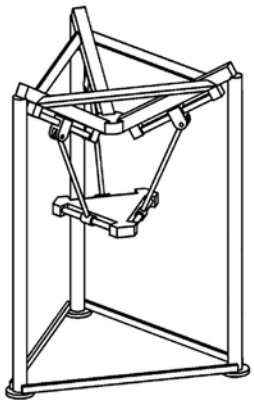


Figure 19: 3-PRC design
from Li and Xu (2006) [63]

Figure 20: Adept's Quatro
from IFR (Adept 2011) [64]

IFR (Adept 2011) [64] developed the 4-RSS Quattro, which was a 4 limbed design based on the Delta robot having 3 translational and 1 rotational DOFs. It could reach accelerations of 15m.s^{-2} and carry a payload of 5kg. The purpose of the Quattro design was energy conservation, and according to a report on benchmark tests it consumed up to 25% and 35% less power than the Delta and SCARA robots respectively, while performing the same

operation with identical payload. When compared to an articulated arm on its fastest cycle performing the same operation, Quattro consumed 43% less average power.

Carricato (2005) [43] developed a manipulator exhibiting 4 DOF Schoenflies motion, i.e. the movable platform could freely translate in space and rotate about a fixed direction. Each DOF was directly actuated by a motor on the fixed frame, through a constant 1-to-1 velocity relation. Both the forward and inverse Jacobian matrices were diagonal and constant throughout the workspace. Consequently full isotropy was achieved which made kinematic analysis trivial, and no significant computation was required for real-time control. It also mitigated leg interference problems. Two models of the design are illustrated in figure 21.

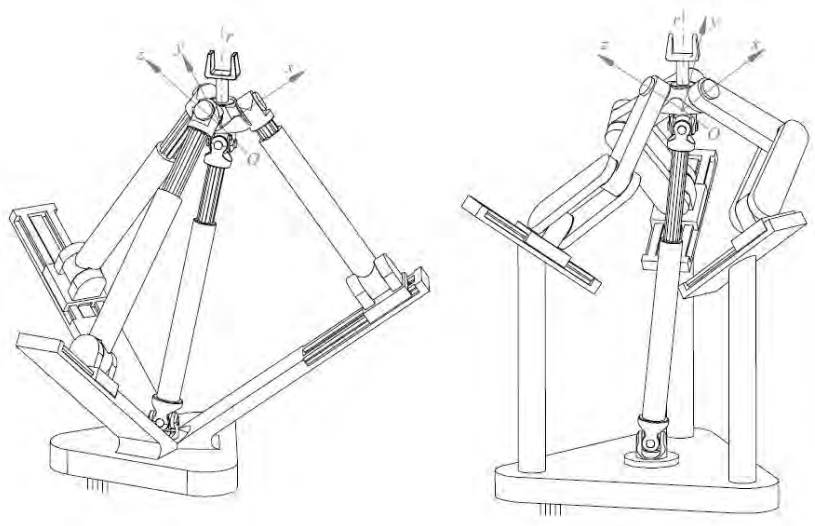


Figure 21: PKM designs capable of Schoenflies motion
from Carricato (2005) [43]

Ng, et. al. (2006) [38] claimed a modular design for a 3-legged micro PKM with applications in micro-machining and assembly. According to *Yang, et. al. (2001)* [65] the modular design concept was introduced in the development of PKMs, to overcome difficult problems of trajectory planning and application development associated with complex kinematic structures. They also mentioned that the concept of modularity had been used in the design of serial-type industrial robots for the reasons of flexibility, ease of maintenance, and rapid

deployment. Their definition of a modular PKM system was a set of standardized modules, such as actuators, passive joints, rigid links, mobile platforms, and end-effectors, that could be rapidly assembled into a complete robot with various configurations. *Ng, et. al. (2006)* [38] said that by adding or subtracting rigid links, and interchanging spherical and universal joints, their design could be configured to have pure rotational movements, pure translational movements or a combination of both, in the micro range. The translational version is shown in figure 22a, with the spherical version in 22b which had a fourth strut linking the base to the movable platform.

a. b.

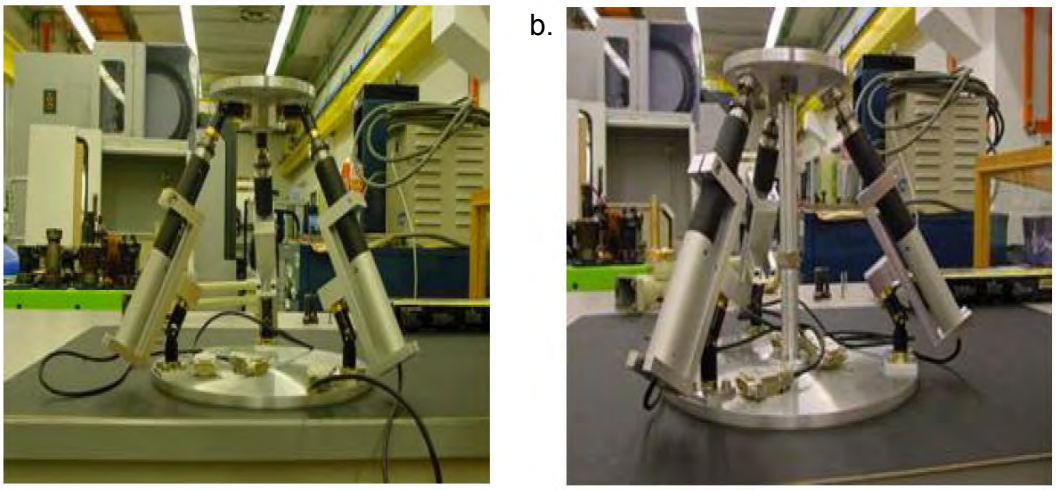


Figure 22: 3 Legged modular micro PKM
from Ng, et. al. (2006) [38]
a. 3 DOF translational version b. 3 DOF rotational version

Orientational PKMs (or parallel wrists) allow the platform to rotate about a fixed point and could be used to orient a body in space. One such fully de-coupled design came from *Carricato and Parenti-Castelli (2004)* [11] and is shown in figure 23. The authors said that its architecture made it compact, robust, and potentially capable of resisting higher forces and torques, for given dimensions, than serial counterparts. The platform was controlled by 2 planar slider-crank linkage-type kinematic chains, and overcame disadvantages usually affecting PKMs.

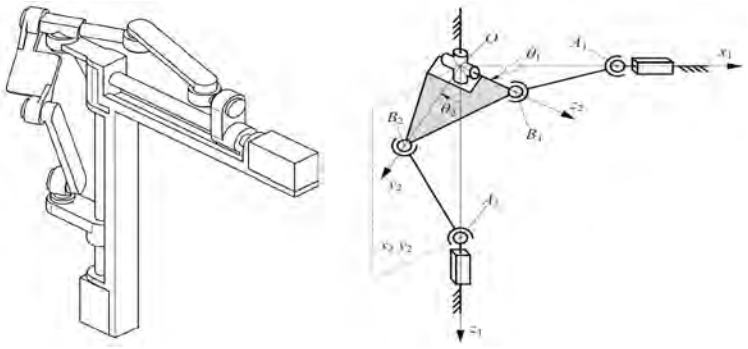


Figure 23: Fully decoupled 2 DOF parallel wrist
from Carricato and Parenti-Castelli (2004) [11]

Zoppi (2004) [5] discussed a new class of 4 DOF PKMs that allowed 3 DOF spherical orientation and 1 DOF translation of a movable platform, which they called the Agile Eye, shown in figure 24. It was composed of 3 identical legs with each leg having 3 rotational joints (2 links per leg) subject to the condition that all joint axes were concurrent (meet at a single point, the sphere centre O). In other words the leg chains were spherical crank-rod bonds, with the base joint of each leg chain actuated. In such an arrangement the end-effector had spherical mobility about O, i.e. it moved on a spherical surface and rotated about a radial axis. If the sphere radius was made infinite (while maintaining link length), the joint axes on each link became parallel and the mechanism turned into a planar 3 DOF PKM, shown in figure 24c. Hence the Agile Eye was a spherical version of the planar 3-RRR planar mechanism, which had a desired orientation workspace corresponding to a cone with a 140º opening.

Zoppi (2004) [5] called his 4 DOF variant ArmillEye, which had 1 translational DOF and 3 decoupled rotational DOFs. ArmillEye was composed of 4 identical legs, and each leg had 5 rotational joints (3 links, with 2 joints that were fixed to each other at 90º). The first 2 links of each leg formed a spherical crank-rod bond, with joint axes concurrent at the common centre of rotation O (the first joint was actuated). The end of the spherical crank-rod bond then attached to a straight link with 2 parallel rotational joints. All legs then connect to the end-effector. This is shown in figure 25a. As in the discussion with the Agile Eye, if the radius of

the spherical part of ArmillEye was made infinite, while preserving link lengths, the spherical part became planar as shown in figure 25b. Both planar versions of the mechanisms were known before the spherical versions.

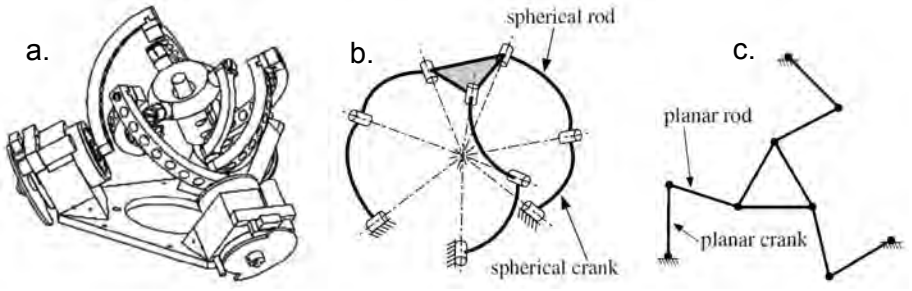


Figure 24: Agile Eye, a 3 DOF spherical mechanism
from Zoppi (2004) [5]

a. Agile Eye CAD model
b. 3-RRR Spherical kinematic representation
c. 3-RRR Planar kinematic representation

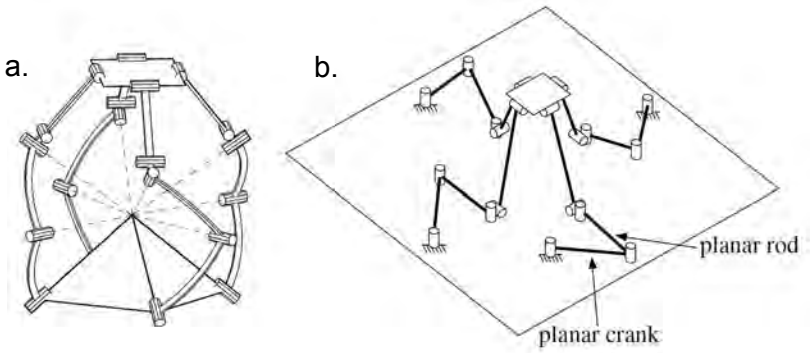


Figure 25: ArmillEye, a 4 DOF mechanism
from Zoppi (2004) [5]

a. 4-RRRRR Spherical kinematic representation
b. 4-RRRRR Planar kinematic representation

Vischer and Clavel (2000) [66] presented a spherical 3 DOF concept they called Argos, which was a spatial structure with its joints arranged around a common centre. The end-effector was connected to its base by 3 identical kinematic chains, where each chain was attached to the base plate by a rotational joint whose axis pointed to the virtual rotation centre. Each axis carried a pantograph, a planar parallelogram with either 4 rotational joints, or 2 rotational joints and 2 spherical joints at its corners. The distal spherical joint on each

pantograph's distal link lies on the surface of a common sphere, and that spherical joint links the pantograph to the mobile platform. The pantograph could be driven at either of its first 2 rotational joints. *Vischer and Clavel (2000)* [66] stated that Argos offered a few advantages over existing spherical mechanisms such as the Agile Eye, 1 of which was a simpler mechanical design (all joints of 1 kinematic chain lie on a plane). This implied that links were easy to manufacture and had less interference during motion. Also joint misalignments would not create supplementary reaction forces in the joints lowering mechanism lifetime.

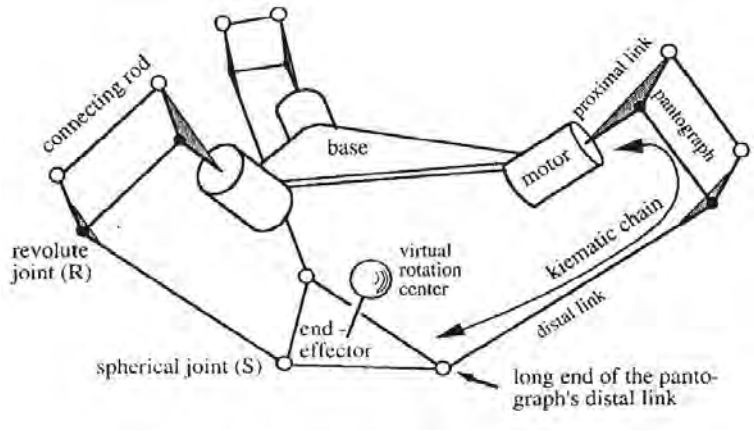


Figure 26: Argos, a 3 DOF sperical mechanism
from Vischer and Clavel (2000) [66]

2.6. Hybrid industrial robot designs

Due to increased modern day automation and flexible production, new applications demand higher performances from industrial robots. The limitations on pure serial and pure parallel robot technology have prompted industrial robot engineers to create hybrid structures combining the 2 architectures with the aim of creating machines that could improve manufacturing processes. These hybrid systems used a combined arrangement of parallel and serial mechanisms to extract the desired features. The basic principle behind most HKM designs was to divide the task of manipulation into position and orientation tasks. Usually 2 and 3 DOF PKMs were used as component building blocks. Recently more hybrid machines have been designed so as to obtain higher speeds, mobility, and flexibility. This section lists some of the more recent designs as well as some of the most popular.

Two commercialized 5 axis hybrid machining centres were mentioned in *Laribi, et. al. (2007)* [39], i.e. the Tricept (by SMT Tricept AB and DS Technologies Sprint, shown in figure 36a) and the Eclipse (DS Technologies Sprint).

Mohammadipanah and Zohoor (2009) [67] described an 8 DOF hybrid manipulator with a parallel mechanism which had 3 translational DOFs and a serial mechanism which had 5 DOFs. It was stated that the manipulator had a large workspace, could reduce energy consumption and had closed form kinematics (forward and inverse). As mentioned by the authors, 5 DOF serial robots were suitable for many applications like arc welding, spot welding, gluing and polishing, tasks which often require end-effectors that had at least 1 axis of symmetry. They were used mostly due to the advantages of a large workspace and their ability to position tools at inner, difficult to reach sections of the work piece. As the workspace requirement increased (robot reach), so to did the actuating energy with larger, heavier serial robots. The authors said that moving the base of a smaller serial robot was more energy efficient than using a larger serial robot having a greater reach, and presented data to support that claim. It was for this reason that they used a 3 DOF PKM translational platform in their design.

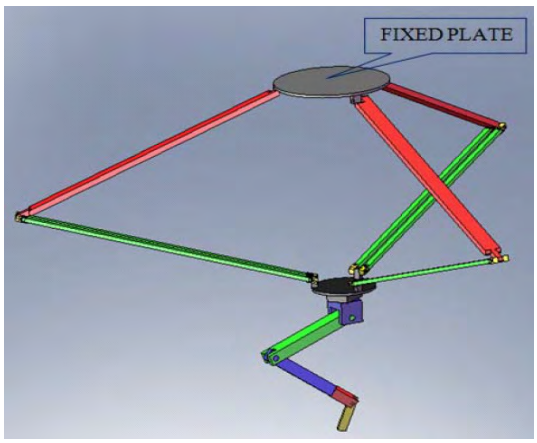


Figure 27: An 8 DOF hybrid machine design
from Mohammadipanah and Zohoor (2009) [67]

Choi, et. al. (2003) [68] designed a 7 DOF hybrid machine, which had a 4 DOF PRRR serial structure and a 3 DOF 3-RPS parallel structure. The design is shown in figure 28. The 3-RPS parallel structure provides 1 translational and 2 orientational DOFs to the mobile platform. The hybrid manipulator was developed for the construction industry to support the human worker by carrying large payloads, and manipulating it in a large workspace. The pneumatic actuators on the third serial axis as well as the parallel section, helped the robot to absorb impact and disturbances from external forces.

Sørensen, et. al. (2004) [69] presented a practical implementation of a modular reconfigurable hybrid manipulator system, which included several different mechanical modules, i.e. an off-the-shelf serial robot arm, custom built PKM, and a custom built sequential placer mechanism. A distributed modular robot controller was developed to convert each mechanical module into a self-contained building block, with a connection for power and network communication bus interface to other modules. The authors mentioned automation of shipbuilding as a potential use for such systems within the heavy industry. Existing machinery in large scale production shipyards were generally based on a combination of two concepts, i.e. a 3 axis gantry system for large movements, coupled with a small serial robot (5-7 DOF) providing the required dexterity for the welding process.

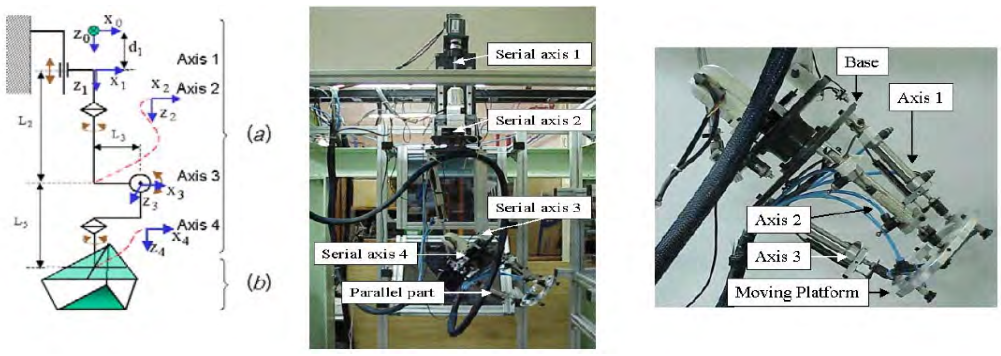


Figure 28: 7 DOF hybrid machine
from Choi, et. al. (2003) [68]

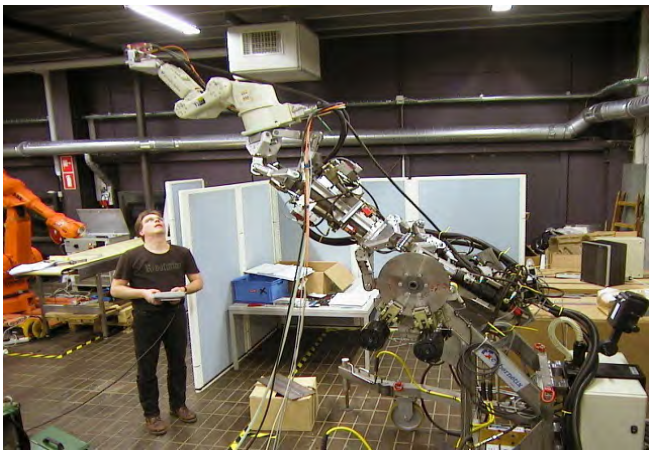


Figure 29: Hybrid reconfigurable manipulator
from Sørensen, et. al. (2004) [69]

A 6 DOF hybrid robot was designed by *Stocco and Salcudean (2000)* [70] which consisted of 2 sets of 5-bar linkages mounted on rotational bases. A movable platform was connected to the output points of both 5-bar linkages via universal joints. The platform exhibited 5 DOF motion, 3 DOFs of translation and 2 DOFs of rotation. Even though each 5-bar linkage on its rotational base could position its output point in 3 DOF, coupling the 2 output points together via the platform removed 1 DOF. The last DOF was realized by adding a seventh motor, which could be mounted on 1 of the 5 bar linkages. This would rotate the platform about the axis defined by the 2 universal joints. The authors said that the advantages of this design were a large non-singular workspace, fewer linkages and joints which reduced mass, friction, backlash and the possibility of leg interference. Their objectives were to produce a simple design that could be tailored for specific applications, to reduce inertia by positioning motors close to the fixed base, and to simplify control computations with a design that had closed form solutions for both the forward and inverse kinematics.

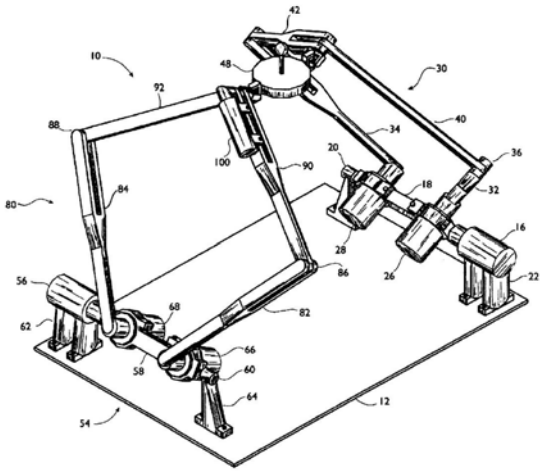


Figure 30: Hybrid 6 DOF machine
from Stocco and Salcudean (2000) [70]

Zheng, et. al. (2004) [45] proposed a 6 DOF hybrid mechanism which connected 2 3-UPU parallel mechanisms serially. One UPU mechanism was translational which would position the mobile platform and the other was spherical which would orient that platform. This arrangement decoupled orientation from translation and made control easier through simpler kinematics. The design was first proposed for use as a compensating platform (for a ship's motion) in a deep-sea mining system.

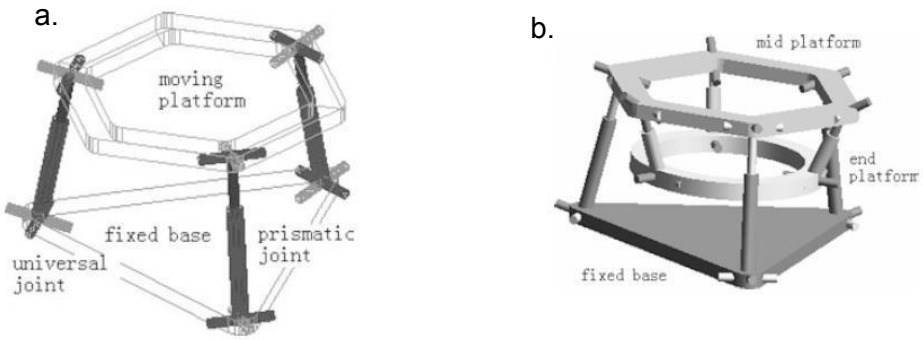


Figure 31: Hybrid manipulator consisting of 2 3-UPU mechanisms
from Zheng, et. al. (2004) [45]

a. Sketch of a translational 3-UPU PKM
b. 2 3-UPU PKMs connected serially, one translational the other spherical

Joshi and Tsai (2003) [71] described a hybrid concept of 2 3-DOF PKMs working in parallel, shown in figure 32. One platform accomplished the task of positioning a tool in 3-DOF space, while the other oriented the work piece about a centre point. The authors stated that these 2 types of mechanisms could be connected in series to form a conventional HKM or in parallel to form a cooperating machine. They further stated that the advantages of both PKMs and SKMs were retained while their disadvantages were minimized thus making the proposed cooperating machine architecture a superior candidate for the next generation of machine tools.

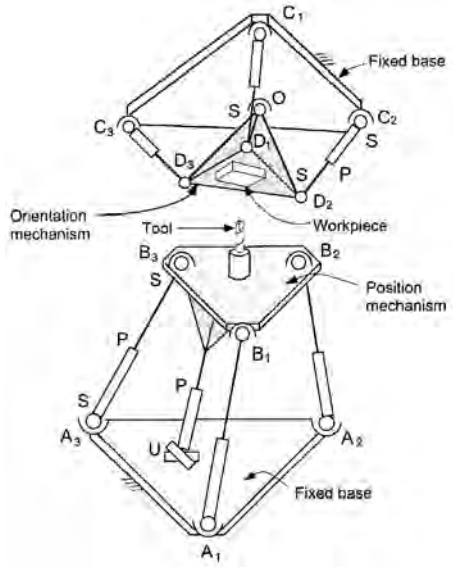


Figure 32: 2 Cooperating 3 DOF PKMs
from Joshi and Tsai (2003) [71]

An article by *Yan, et. al. (2010)* [72] presented a hybrid mechanism used as a forging manipulator for heavy duty tasks in automatic open die forging processes. Here metal was neither confined nor restrained and was formed by compressive forces, creating complex-shaped products in all sizes having good quality, strength and homogeneity. The main platform had 3 translational DOFs and 2 rotational DOFs. It had a sixth DOF, which was decoupled, that could rotate the gripper a full 360º. Most of the actuators were hydraulically driven, which allowed for heavy duty manipulation.

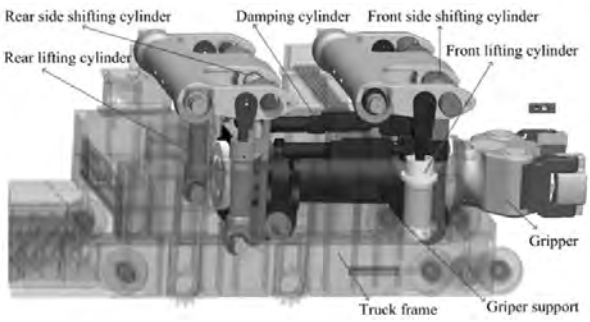


Figure 33: Hybrid forging manipulator
from Yan, et. al. (2010) [72]

Tol, et. al. (2002) [32] combined 2 devices into 1 macro/micro machine, to minimize their weaknesses and maximize their strengths. The macro stage PKM allowed for large displacements in 3D, while the 2 DOF micro PKM compensated to fine tune force control and accuracy. The authors said that their design provided a low cost solution for high precision deburring operations.

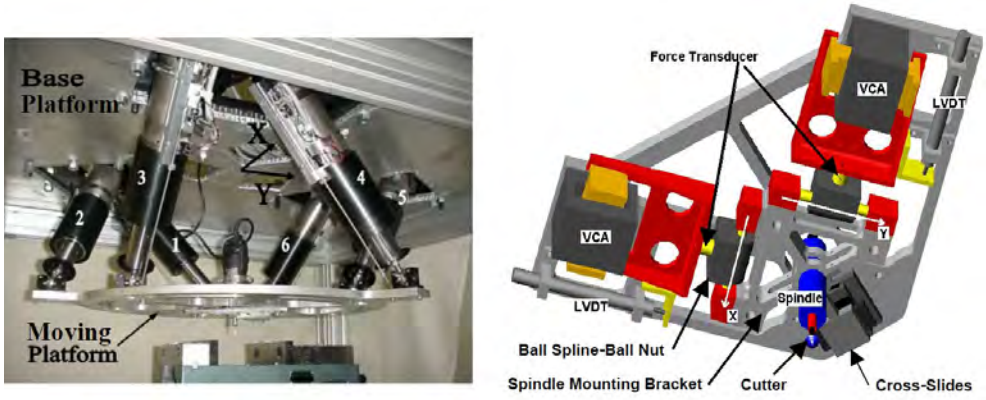


Figure 34: Micro/Macro hybrid manipulator
from Tol, et. al. (2002) [32]

A hybrid system by *Badescu and Mavroidis (2004)* [73], shown in figure 35, was composed of 3 DOF modules for translation (35a) and orientation (35b). The translation modules had a 3-UPU structure, and the orientation modules had a 3-UPS structure, with composing parts nearly identical facilitating modularity. Figure 35c shows 2 cooperating robots and 35d shows a 5 tiered hybrid design with alternating modules for translation and orientation.

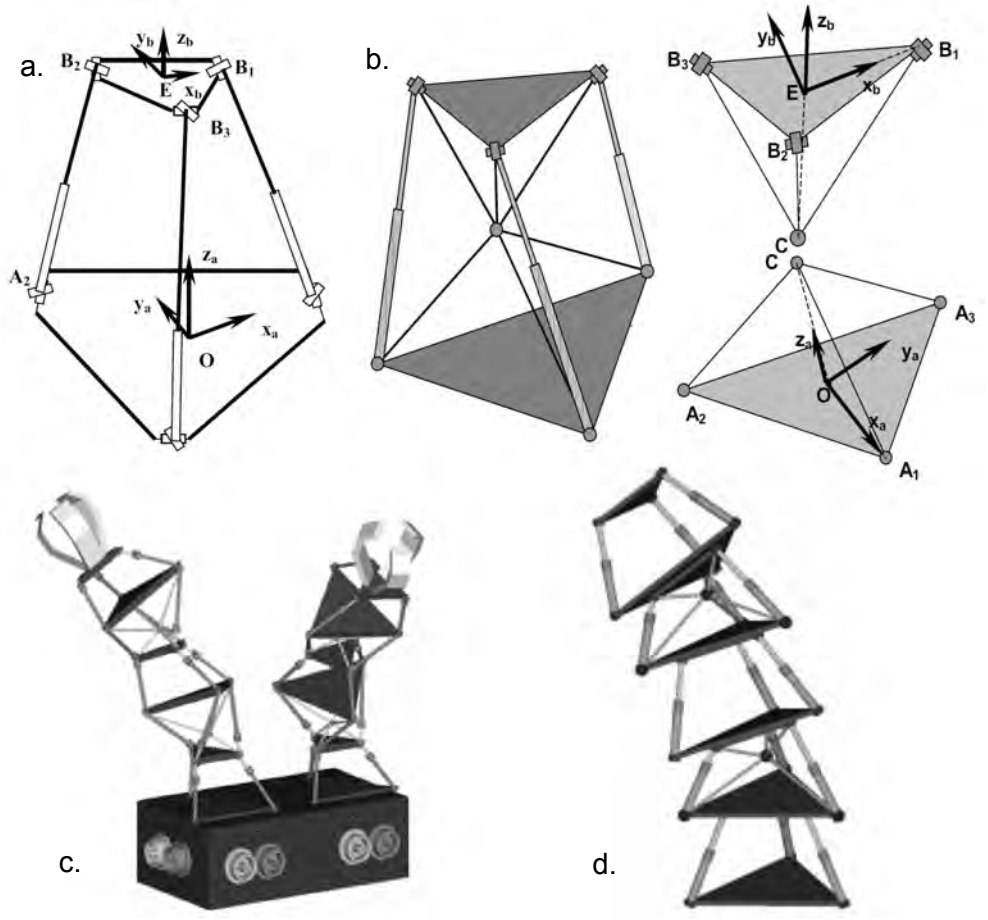


Figure 35: Reconfigurable redundant robotic arms
from Badescu and Mavroidis (2004) [73]

a. 3-UPU translational component b. 3-UPS spherical component
c. Multiple redundant arms
d. Hybrid structure with 3 translational component and 2 pherical components

Zhu, et. al. (2005) [52] mentioned the Tricept robot which had a 3-DOF translational PKM structure and a 2/3-DOF serial spherical wrist. This combination made the workspace large relative to the robot size. The translational part could be regarded as a reduced Stewart platform having 3 active legs. The central passive leg enforced an orientation constraint on the platform. The inverse kinematics had a closed form solution whereas the forward kinematics required iterative numerical algorithms. Other advantages included high machining productivity; good stiffness in both static and dynamic cases and it was priced reasonably. Applications were for light machining i.e. aluminium, plastic, wood and some composites. It could also replace conventional robots when they did not meet the required accuracy and/or process forces involved in applications like friction welding.

Poppeova, et. al. (2011) [74] discussed the Trivariant which was similar in structure to the Tricept, and consisted of a 3-DOF PKM positioning platform coupled to a serial 2-DOF orientation wrist. The difference was in the PKM part of the design where there was 1 active leg aligned with one passive leg.

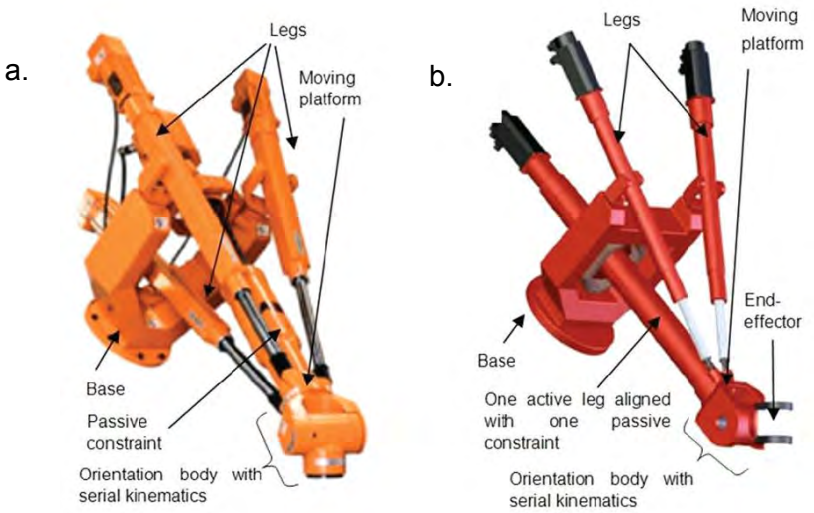


Figure 36: Hybrid machining platforms
from Poppeova, et. al. (2011) [74]

a. Tricept b. Trivariant

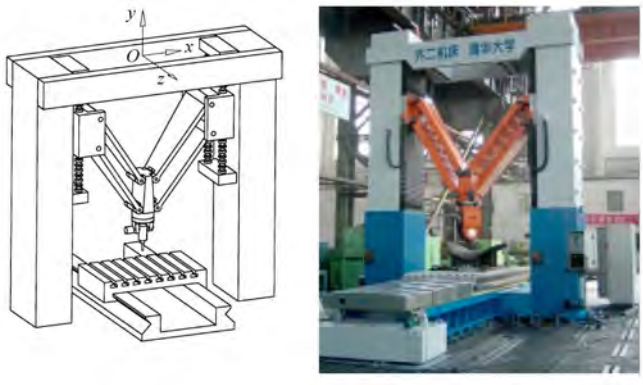


Figure 37: 5 Axis hybrid machine tool
from Tang, et. al. (2003) [17]

Figure 37 illustrates a design by *Tang, et. al. (2003)* [17], where each leg had one 4 bar parallelogram. Multiple parallelograms on each leg increased the system's redundant

constraints, which simultaneously improved overall stiffness and accuracy. Since the links were plates this added to the advantages mentioned. The machine tool consisted of a planar 2 DOF parallel mechanism for planar positioning of the end effector, a 2 DOF serial wrist for its orientation and a 1 DOF linear table which moved the work-piece.

2.7. Singularities

According to *Ceccarelli, et. al. (2007)* [75] the extensive study of manipulator singularities led to several classification definitions, of which the 3 main were:

1. **'Boundary space'** - The reachable boundaries (internal or external) of the workspace, where the end-effector loses 1 or more DOFs, and occurs for both parallel and serial architectures.
2. **'Self motions'** - Manipulator configurations where the end-effector was movable in a localized space even when all actuated joints were locked. In other words the moving platform could not resist any force and were generally eliminated by design, from *Chablat, et. al. (2004)* [33]. These singularities only occur in PKMs.
3. **'Boundary space with self motions'** - Concurrently loses and gains DOFs, which due to the self motions singularity can only occur in PKMs.

Ceccarelli, et. al. (2007) [75] also said that singularities could be classified as:

1. **'Configuration'** – These occur at a particular configuration or pose of the manipulator, and applies to both parallel and serial topologies.
2. **'Architecture'** - These were caused by certain architectures and were inherent to the kinematic design, they did not depend on the specific configuration of the manipulator, and were found in PKMs only.
3. **'Formulation'** – These were due to the mathematical model used for numerical analysis which could be avoided by changing the model.

Zoppi (2004) [5] mentioned another classification for singularities having 6 types, i.e.

redundant input (RI), redundant output (RO), impossible output (IO), impossible input (II), redundant passive motion (RPM) and increased instantaneous mobility (IIM).

There are 3 types of singularities for 6-R serial manipulators, as related by *Hayes, et. al. (2002)* [47]:

1. **'Elbow singularity'** - Stated differently the elbow singularity was the workspace boundary (when the proximal and distal arms were parallel) and could be avoided by keeping the end-effector at a safe distance from its limits.
2. **'Shoulder singularity'** - Shoulder singularities occurred when the centre C of the spherical wrist was coincident with the z-axis of the first driven joint.
3. '**Wrist singularity'** - These were also called end-effector orientation singularities, and occurred when 2 wrist axes became collinear. Due to the construction of the wrist this maybe impossible for consecutive axes, but usually happens with the first and third wrist axes (first axis was perpendicular to second and second was perpendicular to third, but third was not necessarily perpendicular to first).

Perng and Hsiao (1999) [76] said that as a serial robot approached a singular configuration, the inverse kinematic solution required unrealistically high joint velocities and the robot experienced large control deviations. Similarly when a non-redundant parallel robot approached its singular configurations, the robot became statically indeterminate. This meant that there was a non-zero output velocity for the movable platform when all actuated joint velocities were zero (locked joints). Consequently the platform collapsed under gravity, along its singular curve (surface). They also required unrealistically high joint forces when close to singular configurations, and lead to large control deviations in the PKM. In a mathematical context singularities occurred when the inverse kinematics Jacobian became ill-conditioned and non-invertible.

Jacobian matrices relate the velocity in the task space to the active joint velocity. In equation form this was expressed as $B\mathbf{t} = A\dot{\mathbf{q}}$, for all robotic manipulators. In this symbolic representation vector $\dot{\mathbf{q}}$ represented the joint rates, and $\mathbf{t}$ represented the twist array which contained the movable platforms translational (linear) velocity vector $\mathbf{v}$ and the orientational

(angular) velocity vector **ω**. The Jacobian could be expressed as a single matrix J, $\mathbf{t} = B^{-1} A \dot{\mathbf{q}} = J \dot{\mathbf{q}}$. Solutions for $\dot{\mathbf{q}}$ and **t** were possible when A (forward kinematics Jacobian) and B (inverse kinematics Jacobian) were respectively invertible. Parallel manipulator singularities occurred when A, B, or both, became ill-conditioned, non-invertible or singular; whereas serial manipulator singularities occurred when A was non-invertible. Hence another classification arose (*Ceccarelli, et. al. (2007)* [75] and others):

1. **'Type 1 singularity'** - A was singular and B was invertible.
2. **'Type 2 singularity'** - B was singular and A was invertible.
3. **'Type 3 singularity'** - Both A and B were concurrently singular.

'Type 2' and 'Type 3 singularities' occurred in PKMs only, and 'Type 3 singularities' corresponded to the 'self motions' classification mentioned earlier, *Ceccarelli, et. al. (2007)* [75]. 'Type 1' and 'Type 2 singularities' corresponded to 'actuator' and 'end-effector' singularities as mentioned in *O'Brien and Wen (2001)* [77].

Carricato (2005) [43] said that designs which completely decouple a manipulator's DOFs had simple kinematics, simple control and a large workspace. He also mentioned special families of translational PKMs with decoupled motion, which had no uncertainty configurations and straightforward kinematics. Those manipulators exhibited a constant orthogonal Jacobian matrix for the forward kinematics and a diagonal Jacobian matrix for the inverse kinematics throughout their workspace. Full isotropy was achieved for such mechanisms when the inverse Jacobian matrix was constant and orthogonal, which ruled out type 1 singularities.

Carricato and Parenti-Castelli (2002) [51] stated that a manipulator was defined as isotropic if it had at least 1 isotropic configuration, and it was defined fully-isotropic if it was isotropic in its entire workspace.

Fattah and Ghasemi (2002) [44] said that for an isotropic design, the Jacobian matrix $J = B^{-1} A$ had non-zero identical singular values or it had a condition number of 1. This implied that matrix $J J^T$ or $J^T J$ was proportional to an identity matrix, which could be written as

$J^T J = \alpha^2 I$.

O'Brien and Wen (2001) [77] also said that PKMs contained unstable 'self motion' singularities, with no equivalent in serial mechanisms. As a solution to the problem, the authors applied additional kinematic constraints through passive joint braking which were used in the close proximity of unstable poses, instead of redundant actuation. The authors said that 'kinematic instability' was traded for 'local unmanipulability'. The brake system had the advantage of not needing extra actuator/sensor pairs or PKM architecture redesign, and could be used only when required for system stabilization, but had to be designed properly to avoid task axis cross-coupling issues.

As mentioned in *Liu, et. al. (2003)* [34], the configuration space of a PKM was not explicitly known but was implicitly defined by the manipulator's closure constraints. Good understanding of the nature of singularities as well as their relation with kinematic parameters and the configuration spaces was of great importance in the design, planning and control of the system.

Donelan (2007) [10] said that singularities could provide mechanical advantage. When in the close proximity of such configurations, large movement of joint variables resulted in small motion of the end-effector. This could be utilised for load-bearing capacity or for fine control of the end-effector. Also in some cases the singularity provided 'dwell', where the output trajectory was close to stationary for some time which allowed some tasks in a production process to be performed.

Bonev, et. al. (2003) [78] said that the conventional method of deriving the input-output velocity equation for a PKM was by differentiating the inverse kinematic equation, a tedious process that could lead to parameterization errors. Reciprocal screws provided a better approach with additional geometrical insight for a complete and precise description of singularity types.

Carricato and Parenti-Castelli (2002) [51] said that 'self motions' were prevalent in manipulators with less than 6-DOFs. In particular the movable platform of some translational PKMs that experienced 'self motion' lost the capability of pure translation and gained an

instantaneous rotational DOF.

Lloyd (1998) [79] presented a method for handling the singularities of serial manipulators (it could not be used for 'self motion' type singularities in PKMs or robots with complex geometry), where the manipulator's workspace was transformed into a de-singularized workspace. The author said that the adjusted workspace differed from the original only near a singularity surface, where a deformation was applied in the direction normal to the surface. The end-effector could then be moved anywhere in the altered workspace with well-behaved bounded joint velocities and accelerations. The method offered some advantages, in particular for the outer workspace boundary of 6-R serial robots, where it was difficult to perform tasks.

Hayes, et. al. (2002) [47] also mentioned a path alteration method for singularity removal between 2 robot configurations, and said that 'removable singularities' occurred in robots having 3, 4, or 5 DOF, whereas singularities of 6 DOF PKMs were non-removable.

2.8. Justification for a new robot design

At this point it was considered prudent to reiterate some of the most important points made earlier to justify the creation and study of a new robot design:

- 6 DOF robots that can position and orient a work piece were needed as they could be used in a wide range of applications.
- Serial robots were slow, inaccurate and consume significant energy due to a large machine moving mass, a result partly attributed to the location of their motors and gearboxes (which also forces the need for more material on the connecting links to make the robot stiffer).
- There hasn't been a significant innovation in the 6 DOF serial robot design in decades.
- Parallel robots were complex in both design and control, they had large footprints and small workspaces which had a large number of singular positions.

- There is a global drive to reduce energy consumption in industrial robotics (due to the increasing cost of energy and the resulting negative environmental impact).

There was adequate justification for a new design that had a large workspace, small footprint, high end-effector dexterity (6 DOF), and low machine moving mass. The reduced machine moving mass would cut energy requirements while maintaining the same robot speed and accuracy specifications. Serial and parallel robots were complementary, and these design goals could not co-exist in a single purist robot architecture. Combining the advantages from both architectures in a hybrid machine would achieve the desired result.

2.9. Chapter summary

This chapter presented the literature review for the project in a manner that would establish a case for it. Firstly robots are essential to industry, to automate production tasks for high output, quality and consistency. There were a number of reasons to improve industrial robots, and a global drive to reduce energy consumption. There were a few ways in which this could be done but one method was by reducing moment arms about actuated joint axes. This would be accomplished by positioning actuators closer to the base, and reducing machine moving mass (removing excess mass from links and EOAT).

There were 3 types of industrial robot architectures, serial, parallel and hybrid, and a comprehensive comparison between serial and parallel robots was presented. A few serial arm designs were illustrated and those represented the general forms of articulated robot design. The lack of design variation indicated that this was an area for some innovation. Some designs showed that robot manufacturers moved heavier motors closer to the machine base, but had not yet succeeded in a design to keep those motors stationary.

A number of different PKM designs were presented, a clear indication of the geometric variation available. PKMs reduced machine moving mass in general, by fixing all their actuators at the machine base and by using light weight linkages. PKM designs had a number of limbs, usually with equal radial spacing which bend outwards. That gave those

designs a large footprint in relation to their workspace. An area that could present some research opportunities would be designs that reduced the machine footprint and increased the workspace.

PKMs and SKMs were complementary with regard to their advantages and disadvantages. Hybrid designs combined SKM and PKM components and extracted the best features of each architecture.

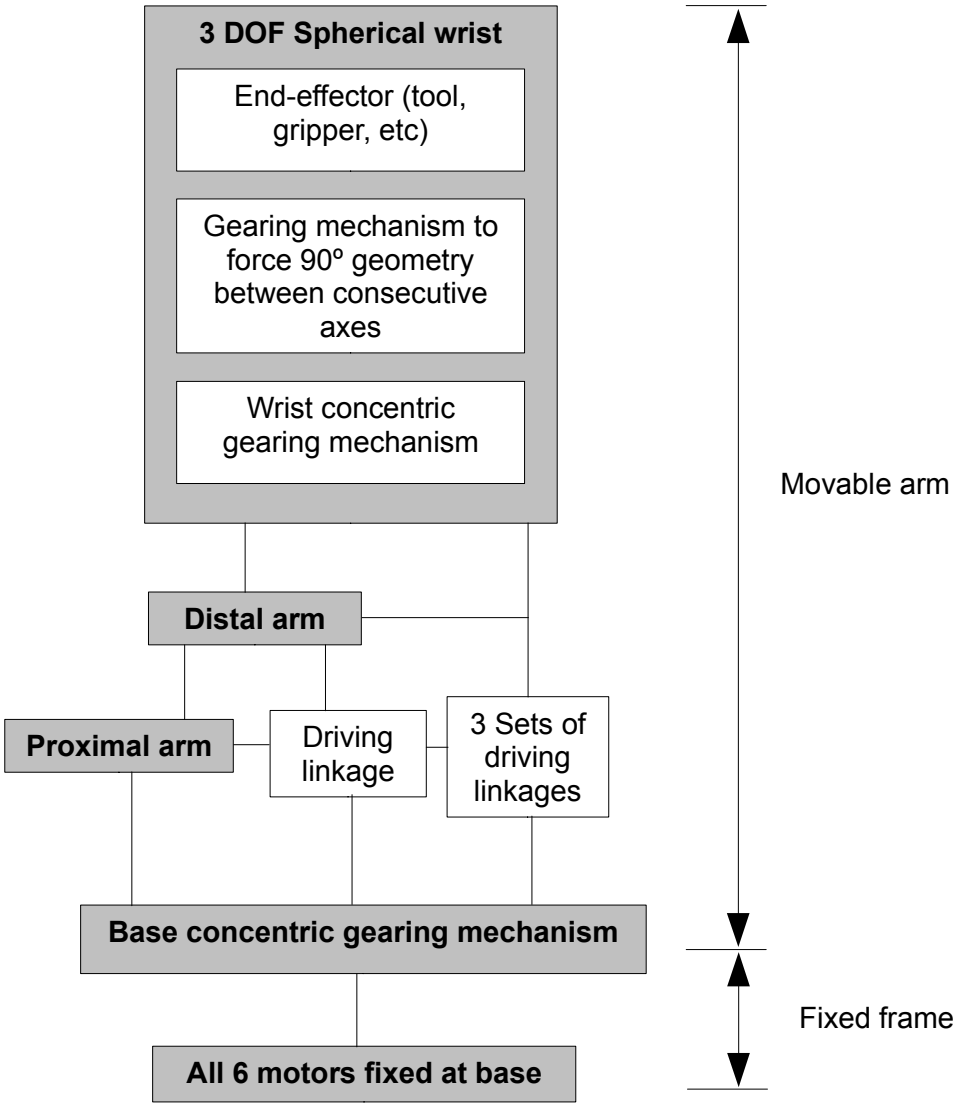
Lastly singular configurations had to be accounted for and methods to detect and avoid them must be implemented.

3. Mechanical design

3.1. Concept and design novelty

The concept was to create a manipulator that had a workspace to footprint ratio comparable to a serial robot, with a low inertia, high speed and improved energy efficiency closer to that of a parallel robot. These requirements were conflicting and did not exist concurrently in pure serial or pure parallel robot architectures. A manipulator that satisfied these requirements would be a significant improvement over current industrial robotic arms.

The machine should have 6 DOFs and a large workspace with high dexterity, which would allow it to perform a multitude of operations. That would make it a replacement for the general purpose serial robot arm. To realize this the concept designs used a concentric base geared mechanism, torque transfer linkages and spherical wrist, see figure 38. The concentric base geared mechanism was a complex mechanical component that allowed actuation of 6 DOFs by 6 motors attached to the fixed base of the machine. This mechanism was critical to ensure a large useful workspace with minimal inertia as it fixed the location of the 3 largest/heaviest motors required for spatial position of the end-effector. The inertia could be further reduced with the use of high strength, light weight linkages using composite materials which should maintain the arm's desired stiffness/rigidity. Furthermore there would be additional mass savings as the links used no longer had to be as strong since they didn’t carry the motor mass. If the same motors were used (fixed at the base), the robot arm could reach even further (main links could be made longer), or move faster, or to some degree combine increased reach and speed.



The fixed frame holds the concentric gearing mechanism in place; the proximal arm supports all driving linkages, and the distal arm supports the upper half of the wrist driving linkages.

Figure 38: Design layout for a novel base driven 6 DOF robotic arm

Robot manufacturer's product guides have many robotic solutions and some of those have mechanisms in place that try to minimize the inertia of the arm. Two of those products were illustrated in figure 5, other manufacturers have similar designs. They used control arms to position the motors closer to the base. This implementation however could only minimize the effect of the motors mass contribution since it was still being moved, only now it was closer to the base. The designs in the subject of this thesis completely eliminated the dead mass problem, and fixed all those motors to a static spatial location. They maintained an equivalent machine footprint, reach and dexterity.

3.2. Design description

The design will be described from the bottom up and consisted of 5 main sections as shown in the design layout, these will be illustrated and explained further. Before getting to that however, some strategies used to improve robot accuracy with regard to end-effector placement had to be mentioned. These could be incorporated into the design, but were not fully illustrated and were indicated as 'black boxes'. They were positioned at the robot's 6 major joint axes.

3.2.1. Methods to improve robot's positioning accuracy

There were a few ways in which a robot's positioning accuracy could be improved, these included the use of harmonic drives, cycloidal drives, planetary drives, self-locking gears, spring-loaded gears, braking systems and external end-effector tracking. Harmonic drives shown in figure 39 were zero backlash according to *Lauletta (2006)* [80] (who cited the definition of backlash as the difference between the tooth space and tooth width, which was 0 since there was contact on both sides of the tooth flank) and less than 1 arc minute (0.017º) according to *GIG (1997)* [81]. Either way they were highly accurate, large reduction gear drives which had single stage ratios that could be as high as 160:1. It consisted of an elliptical wave generator, a flexible cup (which had radial flexibility but torsional rigidity) called the flexspline, and a rigid circular spline. The wave generator fitted into the flexspline which in turn fitted into the circular spline. The elliptical shape of the wave generator, stretched the flexspline outwards to engage with the circular spline. The pre-load stresses were well below the materials endurance limit and ensured long life. *Lauletta (2006)* [80] said that 30% of the teeth were engaged at all times, in contrast with about 6 teeth for a planetary arrangement, or 1 or 2 for a spur gear set. The author also mentioned that wear in conventional gearing increased backlash over time. In harmonic drives the elastic radial deformation behaved like a stiff spring and compensated for increased space due to wear, which mitigated an increase in backlash. Hence performance remained constant over the lifetime of the drive. When applied to a robot the circular spline would be fixed to one link, the flexspline to another and the output of a drive mechanism to the wave generator.

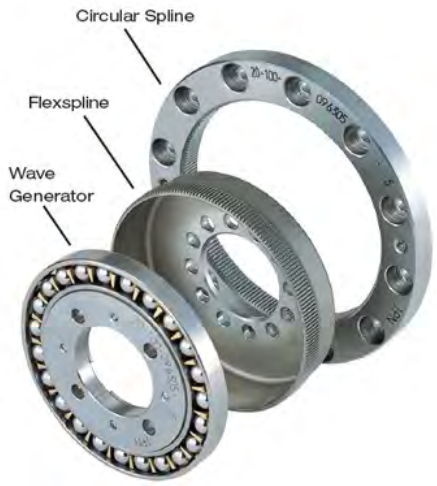


Figure 39: Exploded view of a harmonic drive
from Lauletta (2006) [80]

According to *MSD (2001)* [82] cycloidal drives transmitted power like conventional gears, but came in compact, more efficient packages with concentric shaft orientation. They used non-circular or eccentric components to convert input rotation into a cycloidal motion. That was then converted back into smooth concentric output rotation which reduced the speed in the process. They achieved reduction ratios as high as 200:1 in a single stage, and maintained high efficiencies unlike worm gearing or multi-stage helical gearing, but were less efficient than spur gears. Also less heat was generated as compared to worm gears. If the drive were to break, failure would not be disastrous as the components interact in a rolling manner, and an increase in noise levels would serve as an early warning to failure. Harmonic and planocentric drives were general types of cycloidals. *GIG (1997)* [81] mentioned as disadvantages, a low torsional stiffness (cycloidal drives had a higher torsional stiffness than harmonic drives), speed/torque ripple (non-linear output, especially evident at low speeds) and lower efficiency when compared to the planetary gear sets.

GIG (1997) [81] said that precision planetary gear heads designed to have high torque with low backlash were often used in a variety of applications that required high performance precise motion. Their advantages included high input speeds in excess of 5000 RPM (peaking at 10,000 RPM), high torque with long life (50,000+ hours) due to planet load sharing for heavy loads, less than 3 arc minutes (0.05 degrees) of backlash, compact design

and low noise. Planetary drives offered a range of ratios from 3:1 through to 100:1 (sometimes higher) whereas harmonic and cycloidal drives could not typically go below 50:1.

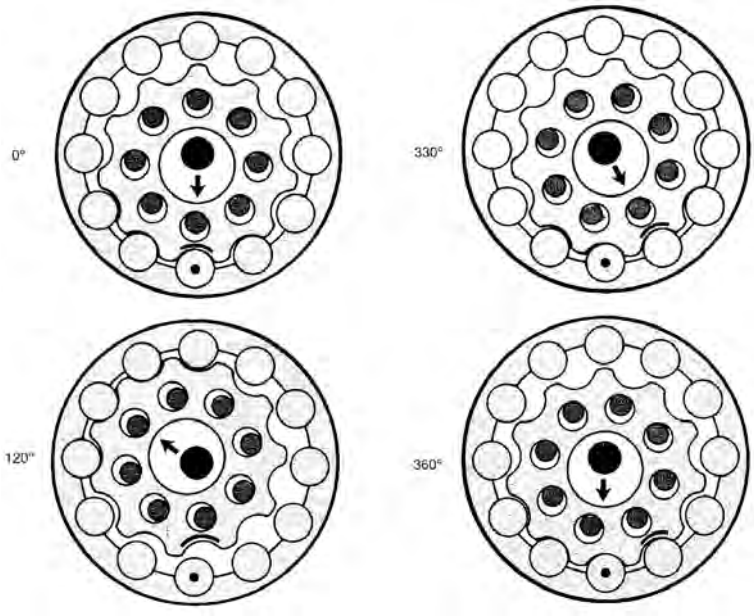


Figure 40: Illustration of cycloidal drive motion from MSD (2001) [82]

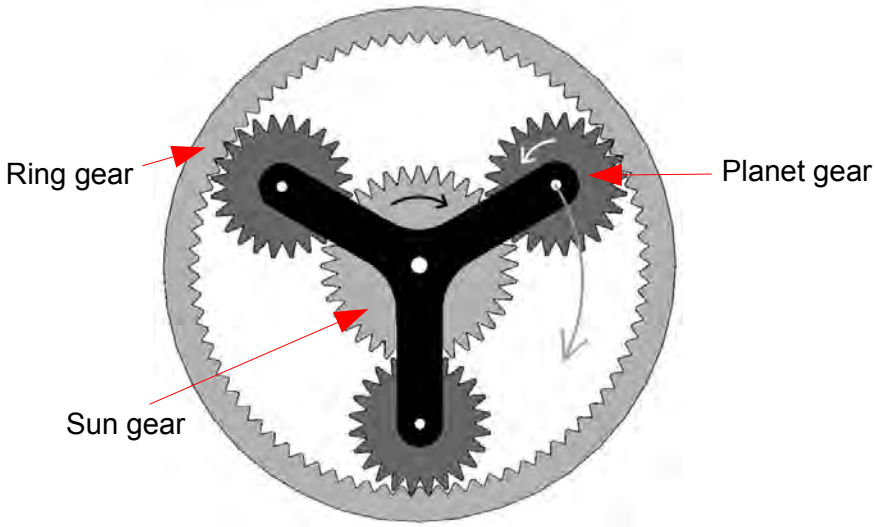


Figure 41: Planetary gear set

The source *GIG (1997)* [81] was more favourable towards planetary gear heads. It stated that in general gears that exhibited a rolling action (planetary) in contrast to a total or partial

sliding action (harmonic and cycloidal), would have a higher efficiency and longer life. Planetary drives had an efficiency greater than 90% while harmonic and cycloidal drives averaged around 80% and 70% respectively. The source also said that the most common gauge for gear accuracy was backlash between engaging teeth, which did not indicate all 'lost motion' within the gear drive. 'Lost motion' was defined as total system backlash, and was a combination of tooth backlash, torsional stiffness, and hysteresis which provided a more complete measure of gear head accuracy. The near 0 backlash capability of harmonic and cycloidal drives was offset by their low torsional stiffness, and greater hysteresis when compared to the planetary gear head. Hence harmonic and cycloidal drives were used in low to medium loading applications. The source concluded that of the 3 technologies, planetary gear heads exhibited the least amount of 'lost motion'.

In self-locking gear sets, illustrated in figure 42, the pinion (usually smaller gear) could drive the gear but the gear could not back drive the pinion via the output load due to gravity or inertia (harmonic and cycloidal drives were also non-back drivable). These could come in the more popular worm (with lead angle less than friction angle) and worm gear/wheel sets as well as spur or helical gears. Self-locking worm gears had some limitations, i.e. the crossed axis arrangement, relatively high gear ratio, low speed, low gear mesh efficiency, increased heat generation, etc. *Kapelevich and Taye (2010)* [83] described the design of parallel axis self-locking gears which could utilize any gear ratio (1:1 to much higher), be external or internal, have symmetric or asymmetric teeth, and could form part of planetary drives or other multi-stage gear drives. These gears had a mesh efficiency much higher than worm gears, closer to conventional gears and resulted in less heat generation. The location of the pitch point P for all conventional gear drives minimized friction to ensure high driving efficiency. When back driven by the output load or inertia, they continued to rotate freely, as the friction moment (or torque) was insufficient to stop rotation. In self-locking gear design, the pitch point P was at a position that increases friction (it also had to have a sufficient friction angle) and *Kapelevich and Taye (2010)* [83] mentioned the design geometry required to maintain relatively high (> 50%) driving efficiency. The self-locking helical gears shown in figure 42c-e, achieved an efficiency of more than 85%. Self-locking gears were sensitive to

operating conditions and locking reliability was affected by lubrication, vibration, misalignment, etc. This type of gearing was essential in applications where it was critical to stop motion when power was lost. For robotic applications self-locking gears could be used as the last set of gears in a gear train to limit the effects of backlash to the very last set of gears. Hence the relative play between consecutive links will be limited to 1 gear set (or 1 gear set backlash per axis). This effect would be additive for a 6 DOF robot and hence it would feel the effect of backlash for 6 gear pairs in total.

Persson and Andersson (2003) [21] said that backlash and elasticity in gearboxes had a significant influence on robot precision as well as error transmission, and that backlash between meshing gear teeth could cause impact, reduce system stability, generate noise and undesired vibration in industrial robots. Compound gears with spring pre-load, could eliminate backlash at the cost of some flank interference (contact on both sides of a tooth).

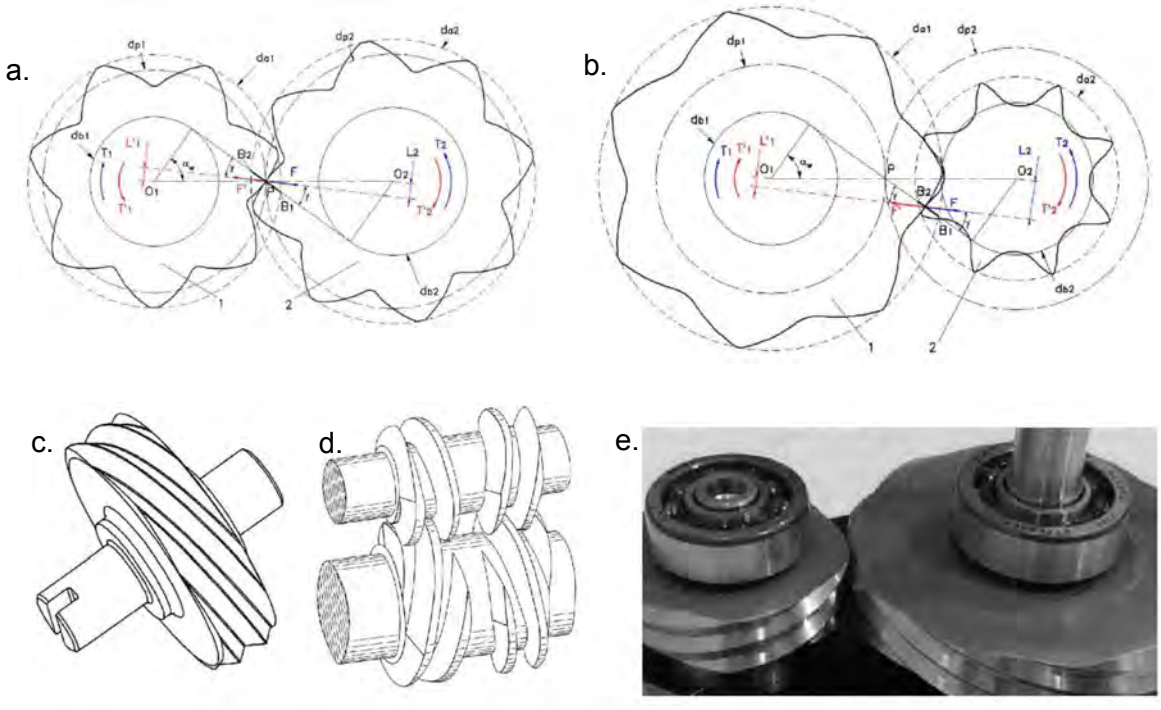


Figure 42: Self-locking gears
from Kapelevich and Taye (2010) [83]

a. Conventional spur gear design
b. Self-locking spur gear design
c. Helical gear
d. Double helical gear
e. Prototype self-locking helical gears

The spring loaded compound gear was comprised of 2 identical gears having a spring between them which forced them apart to engage with a third gear at 2 points. The spring had to be designed for a particular load, if the load was exceeded the spring compressed and positioning inaccuracy caused by backlash returned. This persists until the load drops below the maximum rated load the compound gear was designed for. Spring loaded compound spur gears could be used in planetary gear sets (replacing the simple planet gears) and in self-locking gears. A compound spur gear is illustrated in figure 43a.

Additionally braking systems could be used to hold the relative angular position between 2 consecutive links. This was essential only when changing motor driving direction as all the gears in that mechanical chain had to re-engage in the new direction before any controlled motion could take place. The braking mechanism was used when a direction change was required or an external disturbance was detected to prevent position inaccuracy, hunting and oscillation.

Lastly direct end-effector tracking could be used to minimize positioning errors. Rotational encoders used for joint angle tracking have a position uncertainty and when link lengths are considered this results in end-effector positioning errors. By directly measuring the end-effector position and feeding this back to a control system the errors could be reduced to the measuring sensitivity and accuracy of the external system. There are a few technologies that could be used, such as laser tracking and vision systems. The K600 from Nikon Metrology used controlled LED beacons and 3 linear cameras to determine position. Additionally by placing such LEDs on various link elements and inputting a number of robot positions (moving each axis separately, then in pairs, and then the machine as a whole and tracking links) a more accurate robot model could be made taking into account offset axes, link length errors, etc. Using a better model in the control system could result in better end-effector positioning.

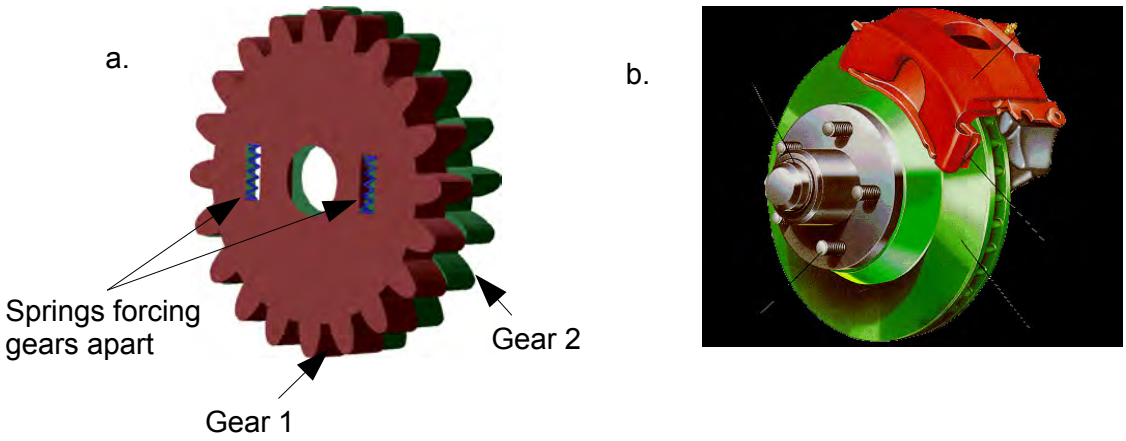


Figure 43: Ways to improve robot positioning accuracy

a. Spring loaded anti-backlash gear sets b. Braking system

3.2.2. Base concentric geared mechanisms (BCGMs)

The base concentric gear mechanism was essential to the idea of fixing all the motors to 1 stationary location at the base, and there were 2 design embodiments which will be illustrated and whose functioning will be explained. The first design used bevelled gears meshing at 90º and was designed in Solid Edge V17, however that CAD package did not have a toolbox for bevelled gears. The design was therefore illustrated with spur gears having large teeth that could be meshed at 90º. This allowed a cheap scaled model to be built with parts laser cut from Perspex. For the second design a migration to SolidWorks 2010 was necessary which had gear wizards and made design more convenient.

3.2.2.1. First embodiment

The motor units and their associated gear boxes (shown in figure 44a) had a fixed position at the robot base (6 sets in all, shown in figure 44b and c). Their arrangement allowed them to occupy dedicated space for themselves as well as to mesh with the gears of the next part of the design, the concentric gear drive. The first design used bevelled gearing.

The concentric gear drive consisted of 7 concentric sections. The outermost section mounted on the fixed base and did not move relative to the 6 inner sections. The 6 inner sections were all capable of rotating independently of each other while remaining concentric.

Each section held its nearest inner section in place (the innermost section did not hold anything), via a double ball race bearing (illustrated in figures 45 and 46). The inner bearings did not carry a vertical load, they facilitated the transfer of torque from the base motors to the designated driver links for the next stage. The outermost bearing was the only one that carried a vertical load, which was the complete mass of the moving machine.

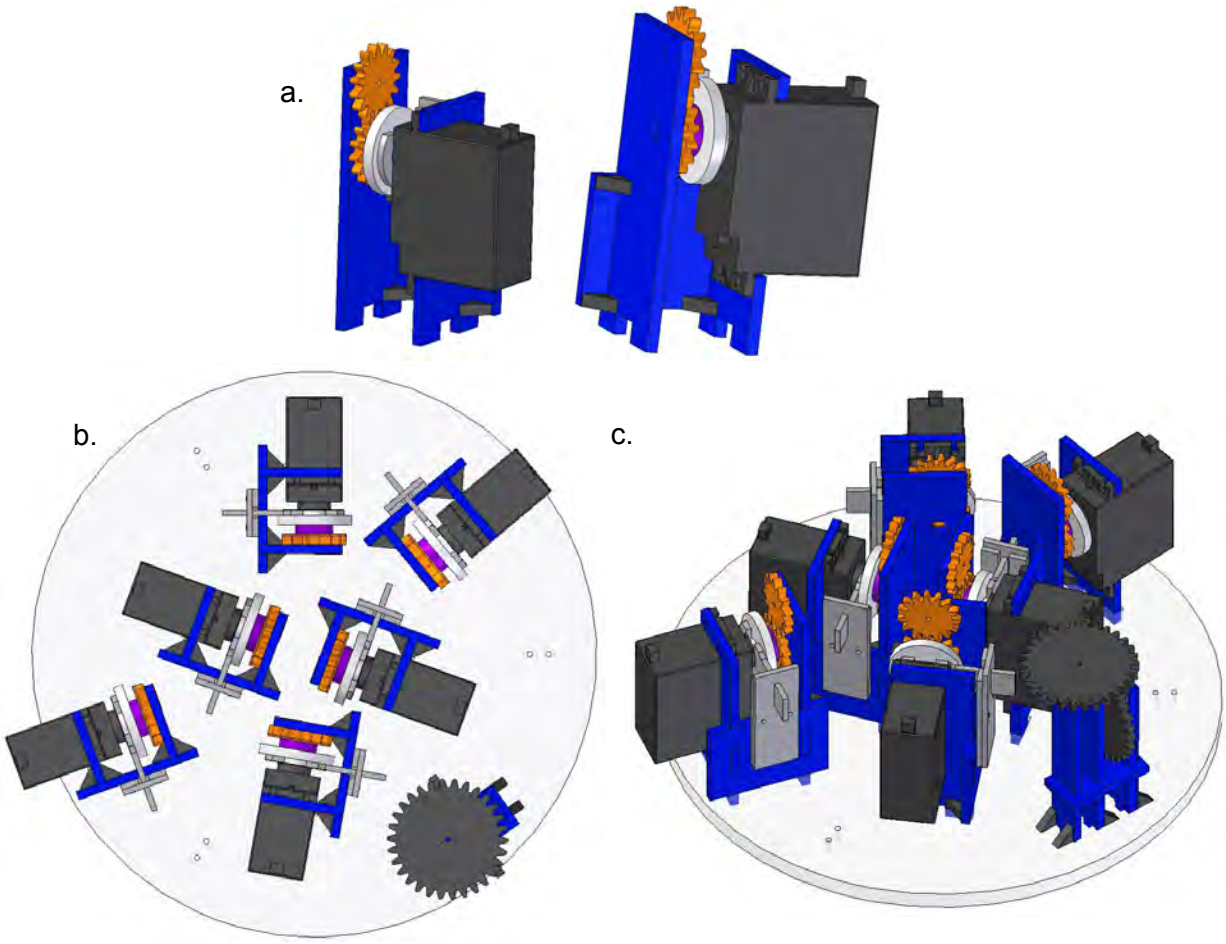


Figure 44: Motor units illustrating their position and arrangement on a fixed base

a. Motors and their gearboxes
b. Top view of motors and their arrangement
c. 3D view of motors and their arrangement

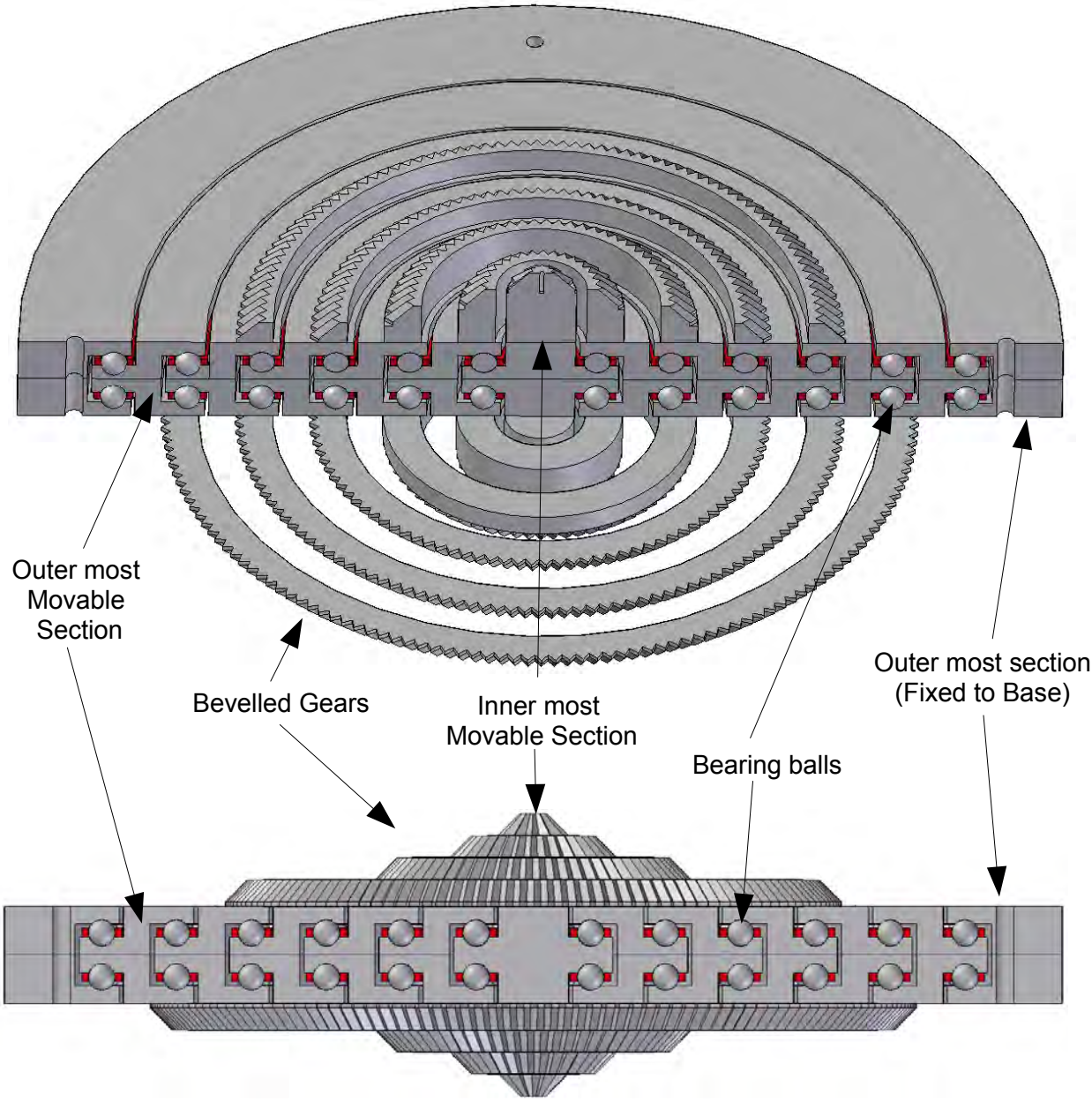


Figure 45: Cross sectional views of concentric gearbox

The 5 innermost sections had bevelled gears on both their top and bottom halves (see figure 45 and 46). The sixth section (counted from the inside moving outward) had a bevelled gear on its bottom half only. On its top half it had a physical mounting for 5 bevelled gears with collinear axes which meshed with the top half gears of the concentric gear drive at 90º (see figure 47). The sixth section (outermost movable section) was responsible for moving the robot arm about the vertical axis.

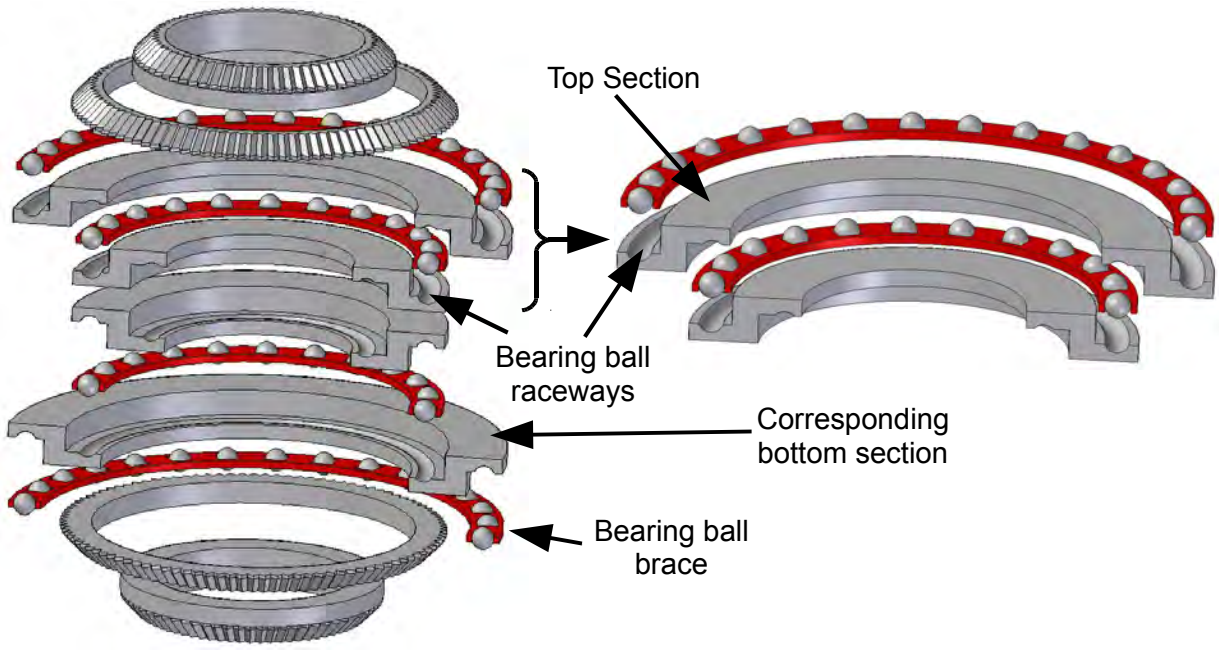


Figure 46: Exploded view of 2 consecutive sections

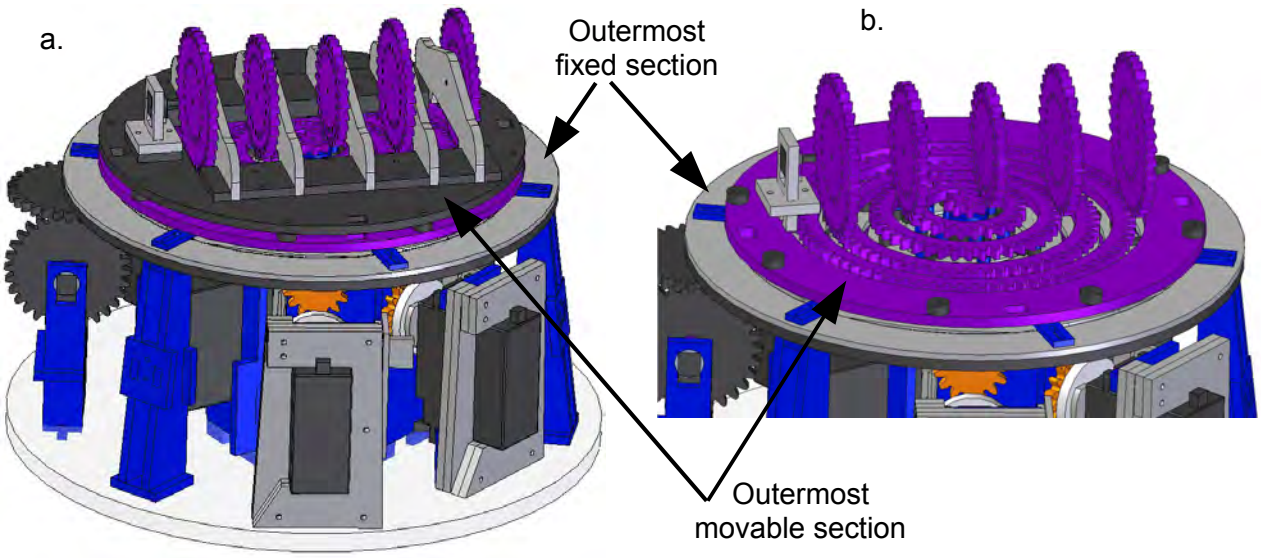


Figure 47: Bevelled gears mounted on top half of BCGD

a. All vertical bevelled gears, with their structural mountings
b. Bevelled gear meshing, with structural mounting hidden

3.2.2.2. Practicality of concentric gear drive design

To prove that the concentric design worked the author created a test mechanism from perspex, this is shown in figure 48. Each concentric unit was split into layers and laser cut,

the layers were then bolted and glued together to give the correct shape. The raceways were 2 concentric discs with a gap between them, so that actual contact with the bearing balls were 2 circular lines from above and 2 circular lines from below. The bearing ball brace had squares cut out (equally spaced on a circumference) and made contact with each bearing ball at 4 points. As such the bearings had minimal resistance when rotating.

The problems experienced with this method of experimentation and construction can easily extend to precision manufacturing, and in fact amplify those that would be encountered. The first problem with the material was that the thickness specification had a large tolerance, the 5mm thick sheets used ranged from 4.5mm to 5.5mm, and each sheet had a different thickness at opposing ends. Additionally the laser cutting wasn't accurate, it would cut into the profile or extend out a little and at some places the laser wasn't perpendicular to the sheet (when you consider the fact that the laser flat bed wasn't 100% level and that the material had a varying thickness).

Even with these problems the concentric drive worked to satisfaction. One major problem of note was that since each unit supports each unit on the inside, having large manufacturing tolerances on the raceways would allow some planer and vertical translational movement of the concentric portions of the drive. The inner most section of the drive has the largest displacement, however this wouldn't be a major problem when the rest of the drive is considered as the gears on the top and bottom halves of the drive mesh with gears that are not allowed to translate either vertically or in a planar fashion.

Actual manufacturing of the concentric portions would start with a solid disc. The top half raceway would be machined out, the disc would then be turned over and the bottom half raceway machined. Adequate material would be spared on the correct side of the disc for the cutting of gear teeth. Holes would also be cut, and threaded on each plate so that the 2 plates of each concentric section could be bolted together. Assembly would then start from the inside moving out. The biggest disadvantage of the drive is the number of parts that go into its construction as well as the number of processes required to manufacture each part. Also manufacturing tolerances would have to be tight to limit the vertical and planar translation.

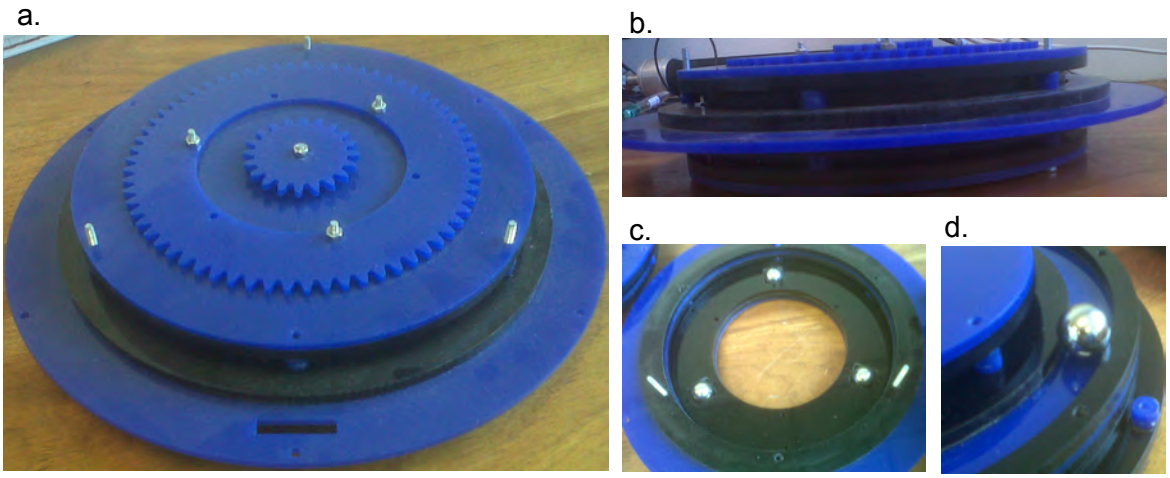


Figure 48: Perspex model of concentric geared mechanism

a. Assembled concentric mechanism
b. Side view of mechanism
c. Bearing balls and ball brace in raceway
d. Bearing ball in raceway

3.2.2.3. Second embodiment

The motor units and their associated gear boxes (shown in figure 49a) had a fixed position in space (6 sets in all, shown in figure 49b-c). Again the arrangement allowed them to occupy dedicated space for themselves as well as to mesh with the gears of the next part of the design, the concentric gear drive. The gears in this design variation were spur gears and resulted in a more compact design.

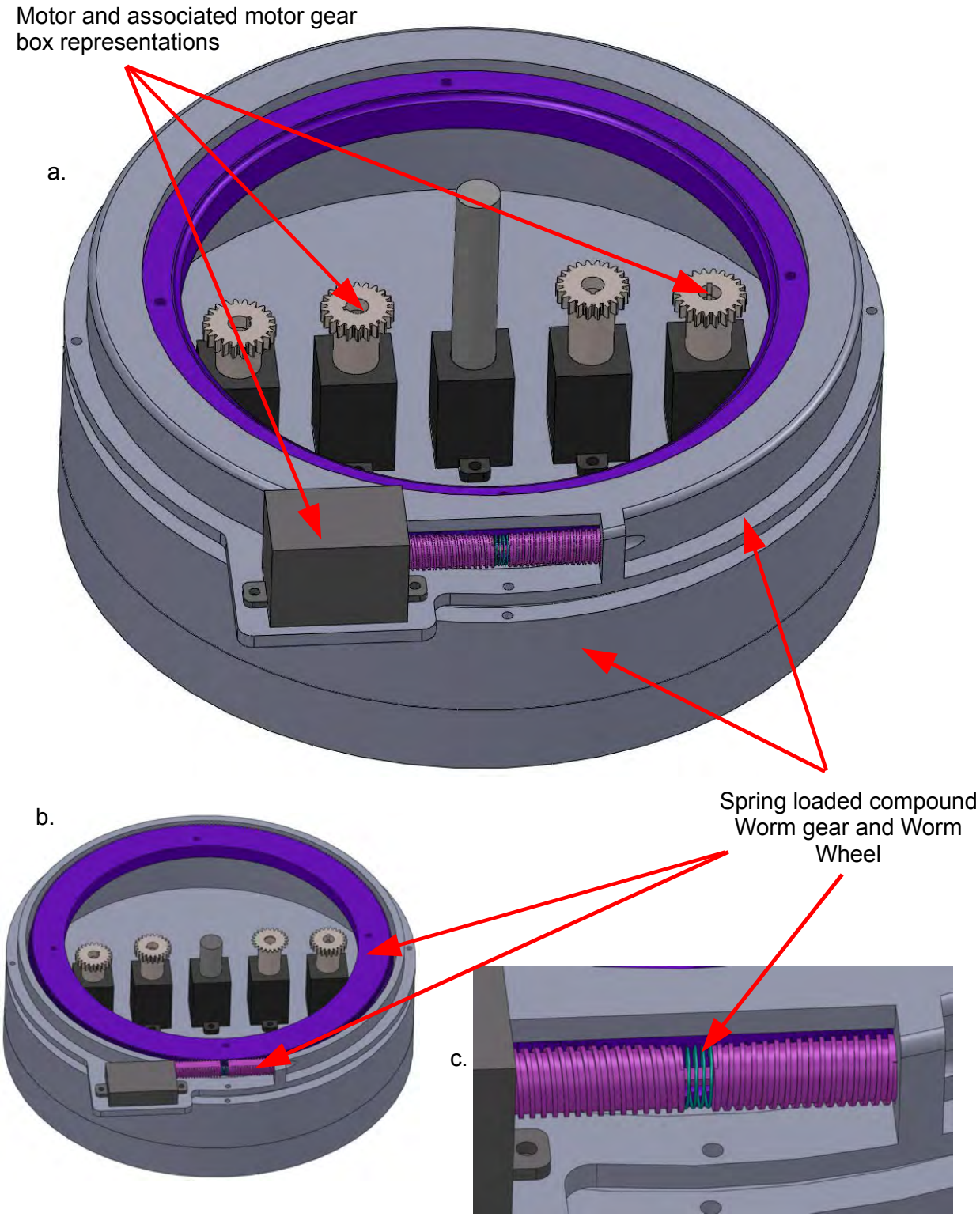


Figure 49: Motor units illustrating their position and arrangement on a fixed base

a, b. Motor units and position c. Spring loaded worm

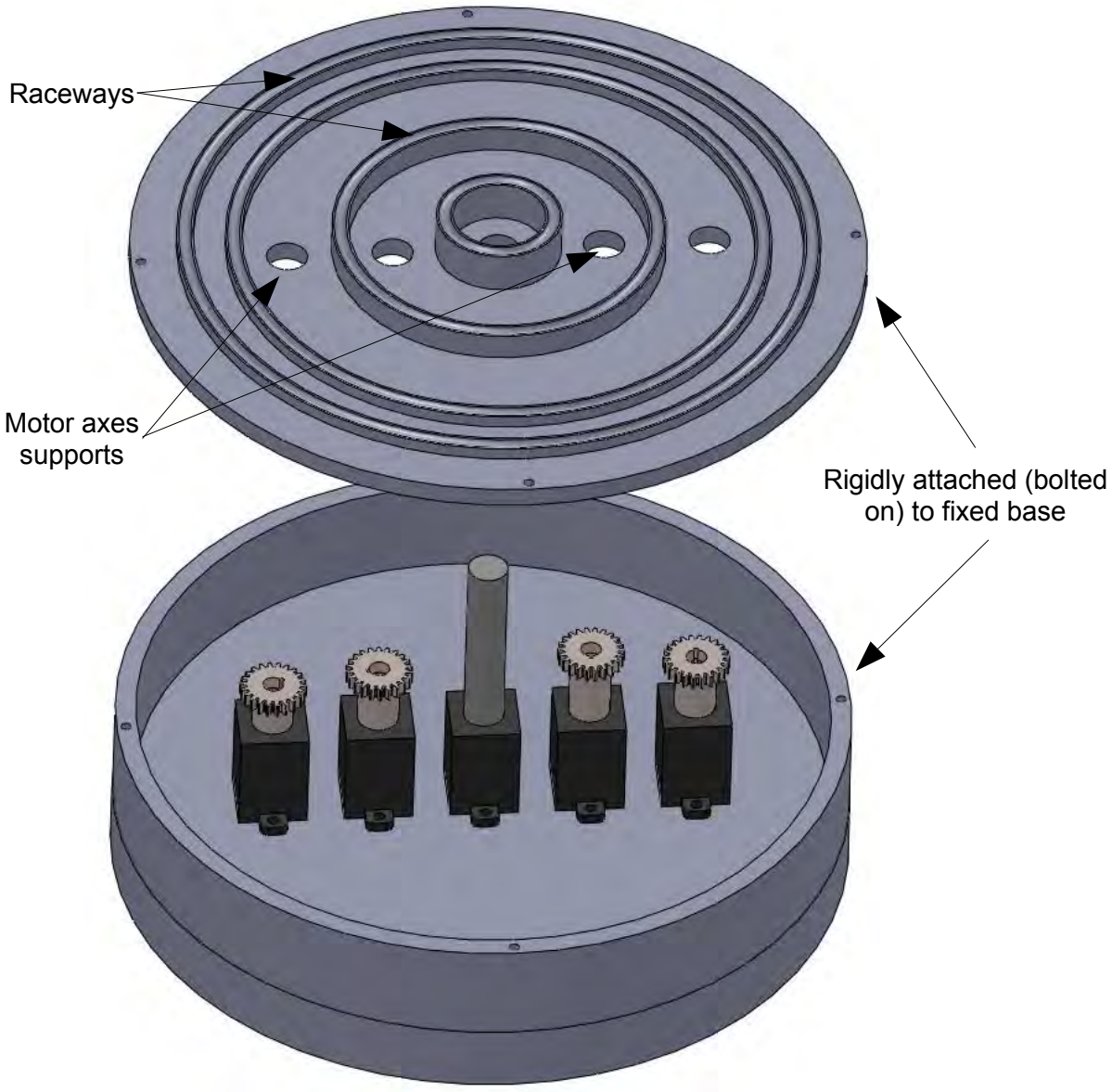


Figure 50: Raceways and motor axes supports

Figure 51: Outermost section securing 6th movable section to fixed base

The sixth movable section, in this case, was a worm wheel. The movable structure of the arm was bolted to this worm wheel and it controlled the arms position about the vertical. This was the first controllable axis in the robot arm.

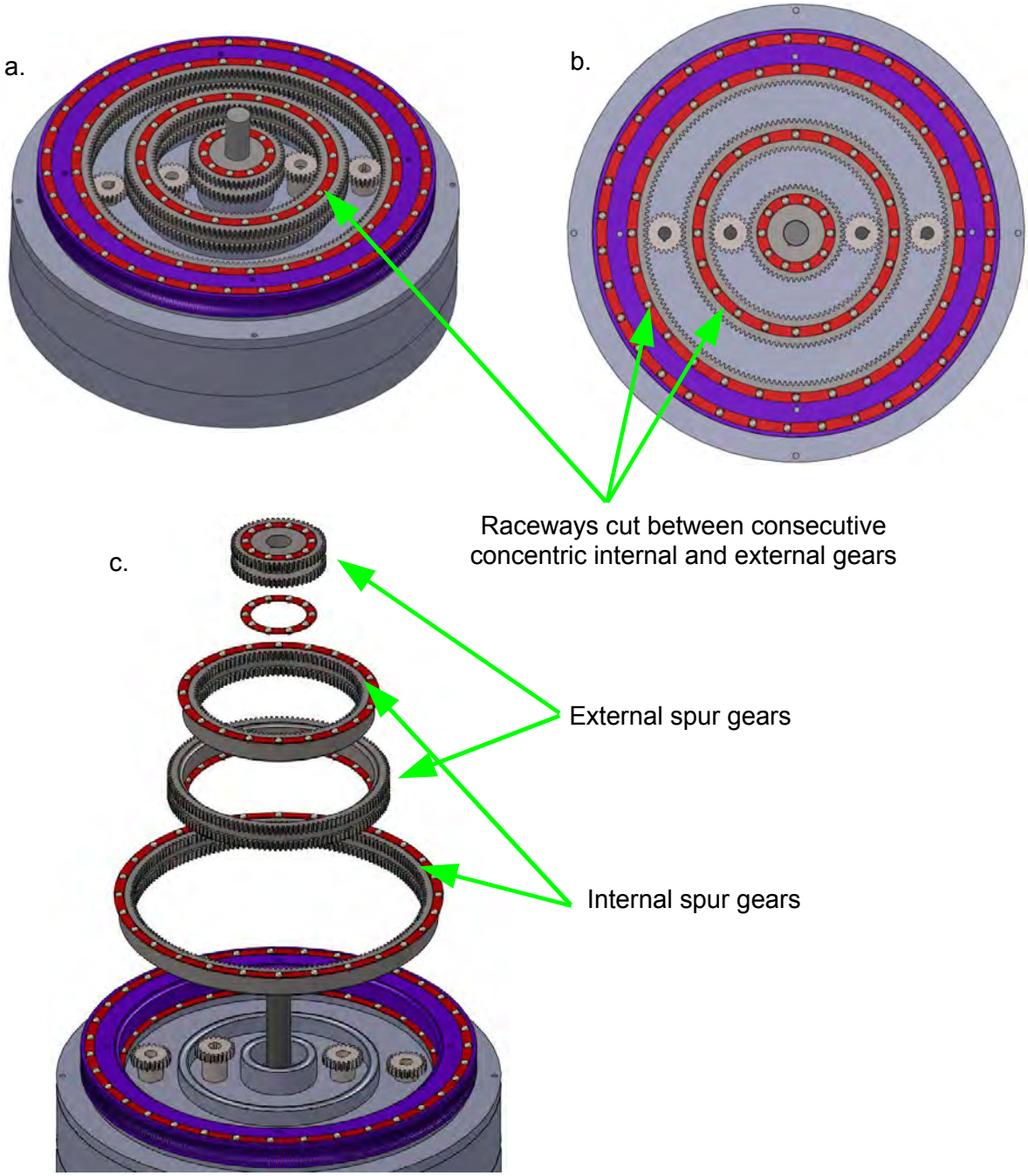


Figure 52: Various views of design 2's BCGM

a. Isometric view
b. Top view
c. Exploded view

The 6 inner sections were all capable of rotating independently of each other while remaining concentric. The concentric gears were held in position by top and bottom covers which themselves had raceways which were extruded to provide an exact fit. The bottom cover was illustrated in figure 50 on page 73, and the top cover is shown in figure 55 on page 78. Each set of consecutive internal and external spur gears on the concentric base drive

share a single raceway and ball bearing, however each of those gears could have its own dedicated ball bearing and raceway. The dedicated raceway design was omitted to keep the illustration both neater and easier to understand. This is shown in figures 50 to 53.

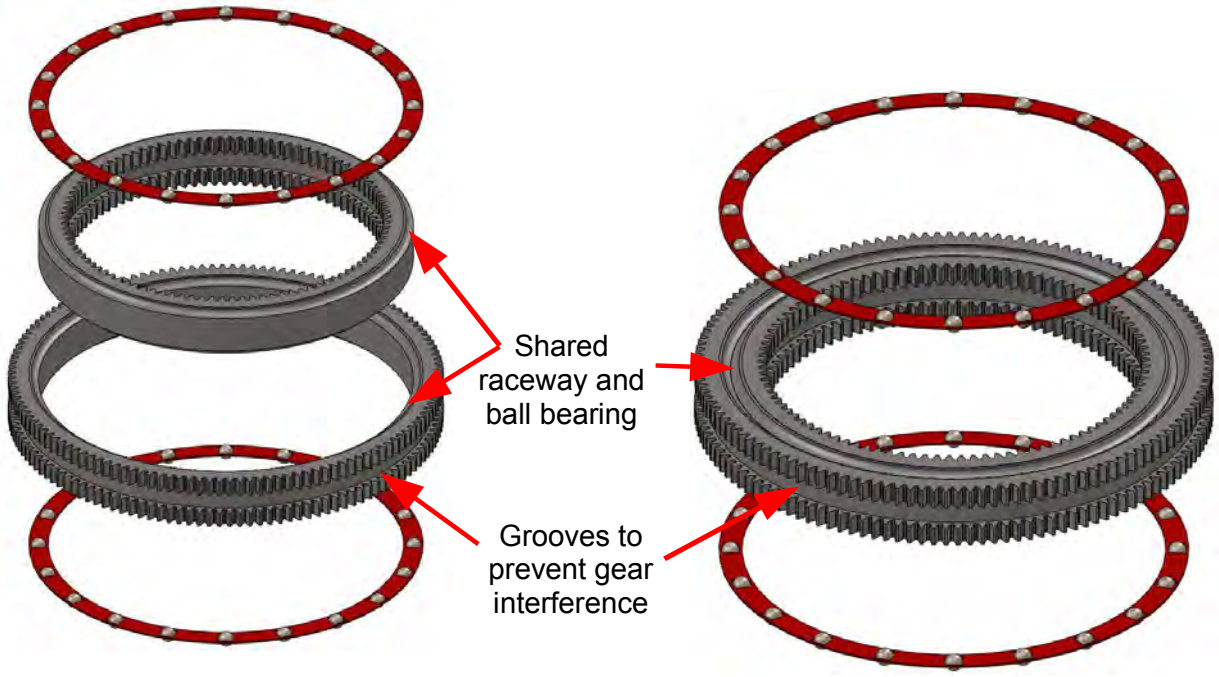


Figure 53: Close up view of consecutive internal and external gears

Each of the gears in the concentric drive had grooves cut in them which served to prevent any interference with other gears in the drive during rotation, i.e. those coupled directly to the motors and those that transferred torque to the next stage of the arm. Additionally the concentric gears were staggered vertically, again to prevent gear teeth interference. This is indicated in figures 53 and 54.

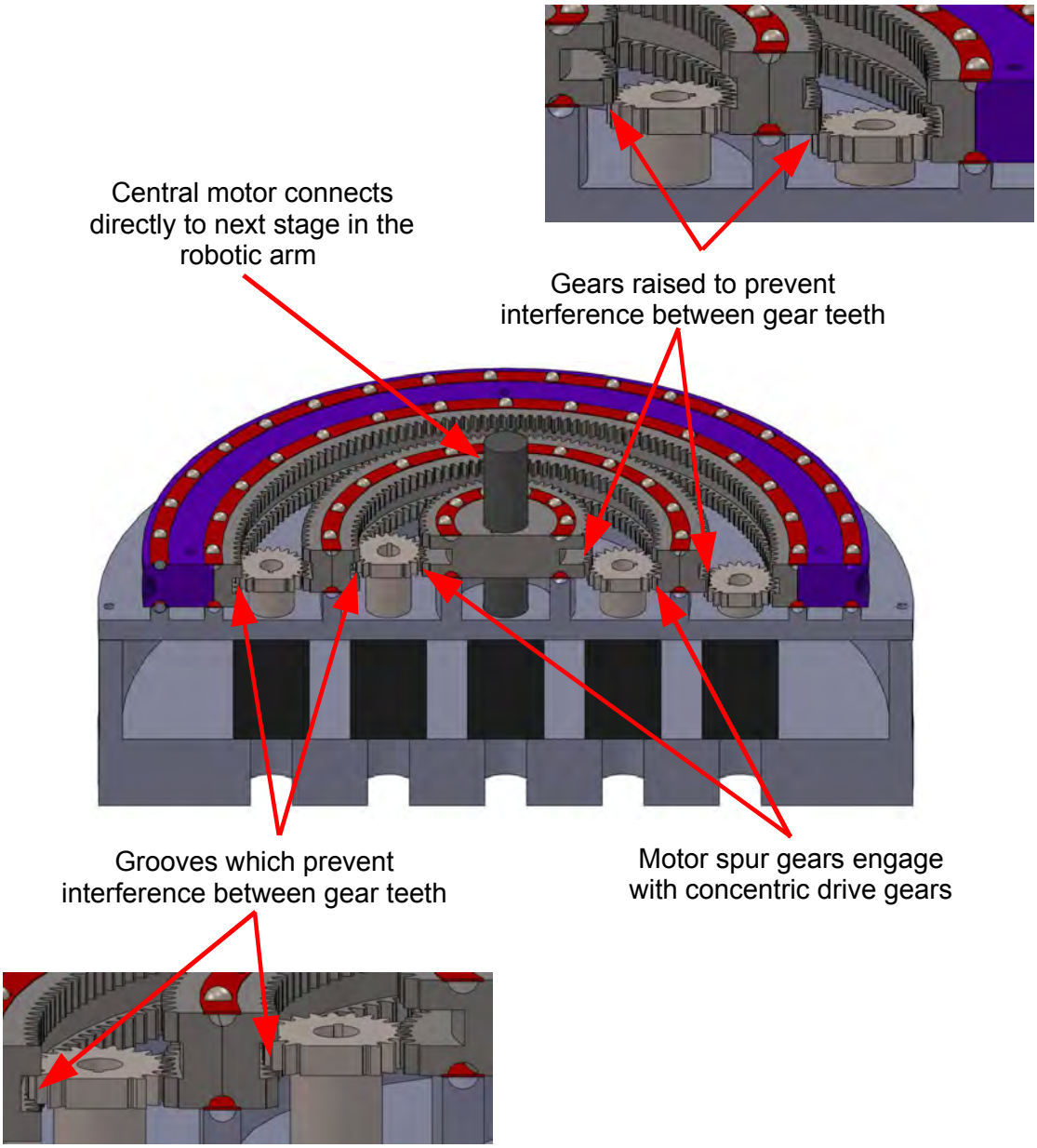


Figure 54: Prevention of gear teeth interference on concentric drive

The inner bearings in the concentric drive did not carry a vertical load, they facilitated the transfer of rotation and torque from the base motors to the designated driver links. The outermost bearing was the only one that carried a vertical load, which was the complete mass of the moving machine.

The top cover of the concentric drive held the gears in place, and it mounted rigidly to the outermost movable section (the worm wheel). The 4 inner gears on the concentric drive then mesh with spur gears mounted on the top cover. The central motor could connect directly to

the next stage in the robotic arm and had no need for a gear set in the concentric drive. Those spur gears were rigidly connected to bevelled gears via a tubular shaft which redirected torque to the remaining 5 axes of interest on the robotic arm. See figures 55, 56 and 58.

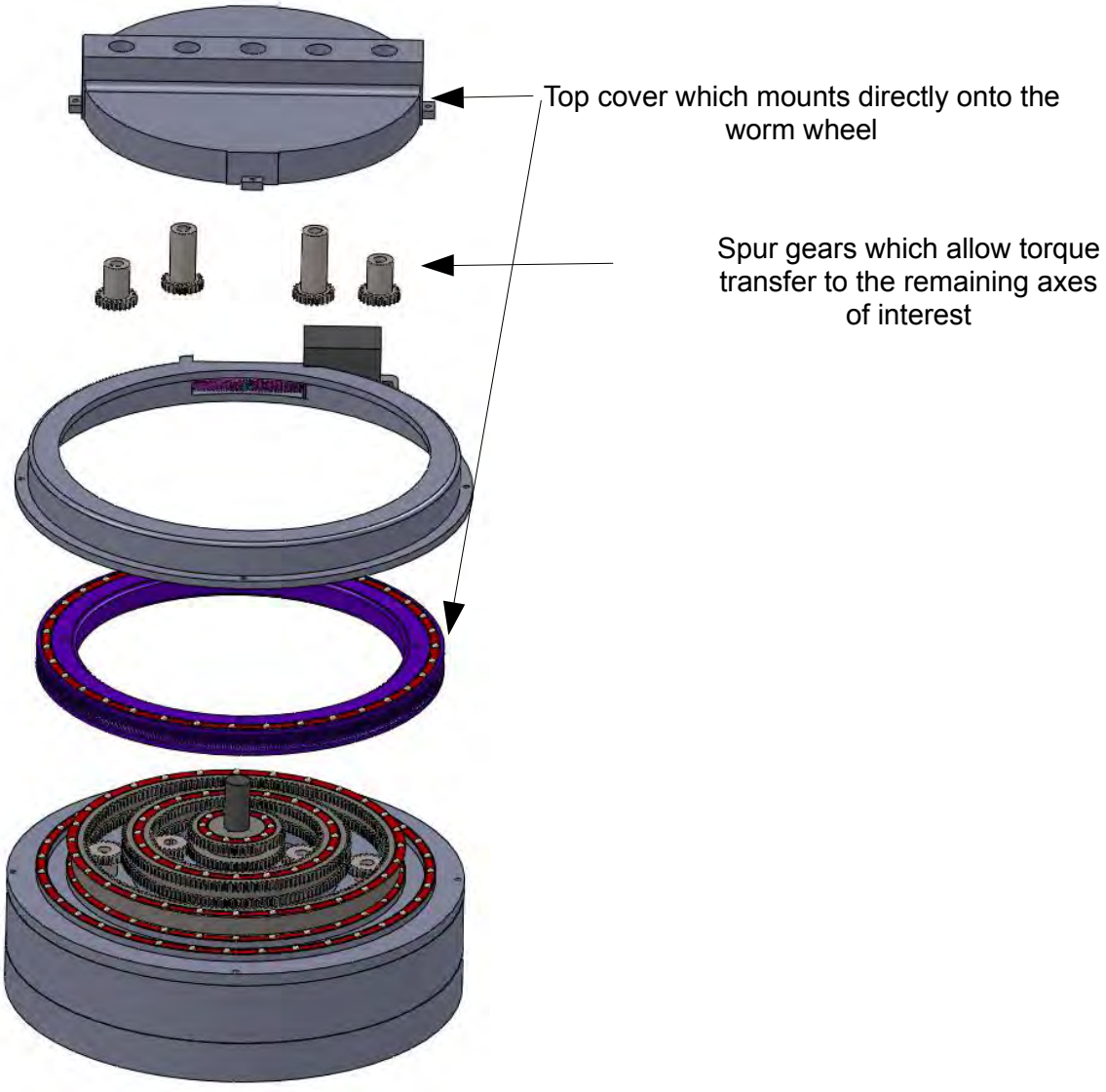


Figure 55: Exploded view of concentric drive showing top cover and worm wheel

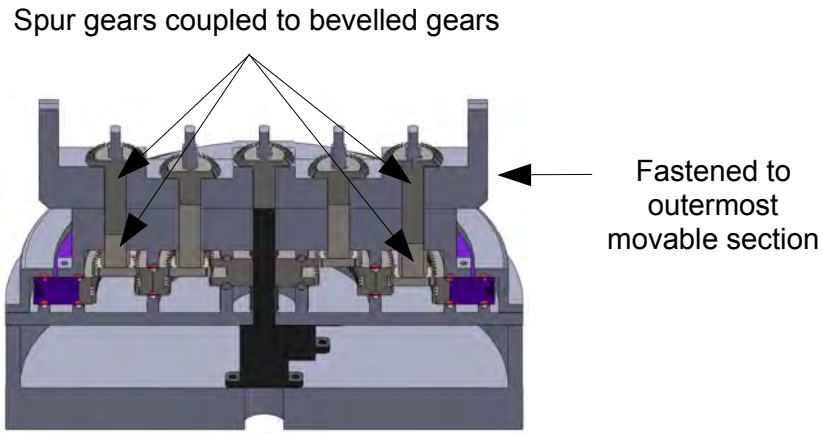


Figure 56: Coupled spur and bevelled gears of next stage

3.2.3. Torque transfer methods

The 2 ways in which torque transfer could be accomplished were through rigid link and non-rigid link rotation transfer mechanisms. Toothed belts or chains would have to be used for the non-rigid link option. The designs that follow focussed on the use of rigid link torque transfer mechanisms. The initial choice for the first design was a parallelogram linkage but there was no simple mechanical solution to prevent the singularity position (when the parallelogram collapsed or adjacent sides became collinear, the exit configuration could be either a parallelogram or a crossed quadrilateral). There were experiments with designs in which extra links were used to create double parallelograms, with a phase offset so that when one collapsed the other prevented the crossed configuration. Another solution was to maintain the crossed configuration, and this required a moving slider-pivot joint between the longer sides of the quadrilateral. The 3 link slider-pivot linkage, as shown in figure 58, was the simplest solution that achieved the required objective.

For the second torque transfer mechanism a geared linkage was used that could transfer torque in any configuration of the proximal and distal arms. This is shown in figures 61 to 63.

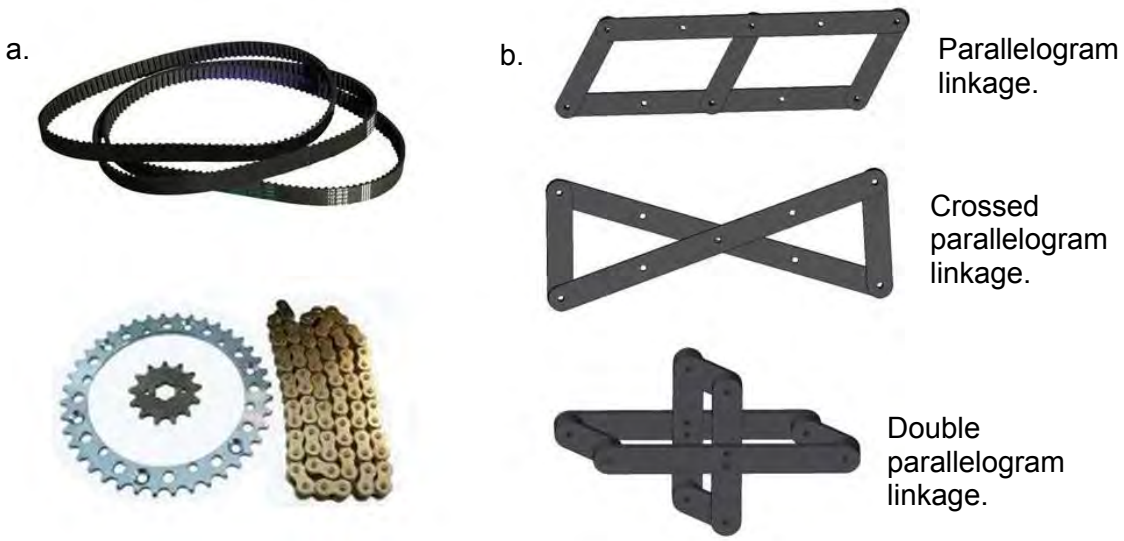


Figure 57: Drive options for the first design

a. Non-rigid link actuation transfer b. Experimental linkages

3.2.3.1. 3 Bar slider-pivot linkage used in design 1

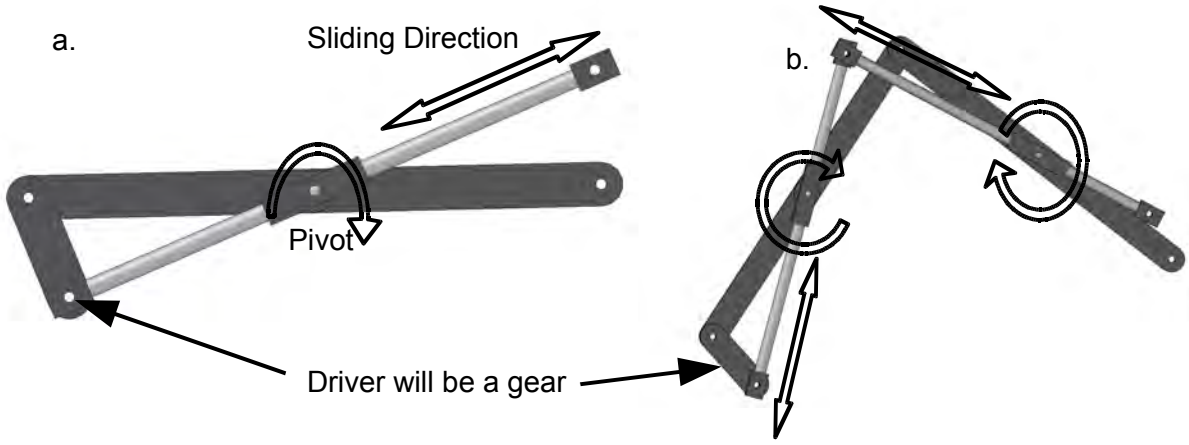


Figure 58: Slider-pivot linkage

a. Primary slider-pivot linkage
b. Primary slider-pivot linkage connected to secondary slider-pivot linkage

To transfer actuation away from the base slider-pivot linkages were used, these are illustrated in figure 58. The orbit of the follower point (linkage output) did not require a 1 to 1 or 1 to -1 match with the driving link (linkage input), which would represent equal radii circles.

The follower however had to match the angular rotation of its driver (not positional magnitude). In other words the follower had to have a 360º rotation for every 360º rotation of its driver. There were angular velocity variations in the output for a constant angular velocity input but those were acceptable for the simpler linkage design, and will be addressed in the simulations section (chapter 5, simulation results) of this thesis.

The slider had a pivot at the midpoint (which allowed the slider to rotate) of the supporting link (that position minimized warping of the follower orbit and maintained a somewhat circular profile). Furthermore the follower on the end of the primary slider-pivot linkage became the driver to secondary stage. The orbit of the follower on the secondary stage was further warped but still circumnavigated the main wrist axis, and matched each degree of rotation of the driver on the primary stage. This will also be addressed in chapter 5.

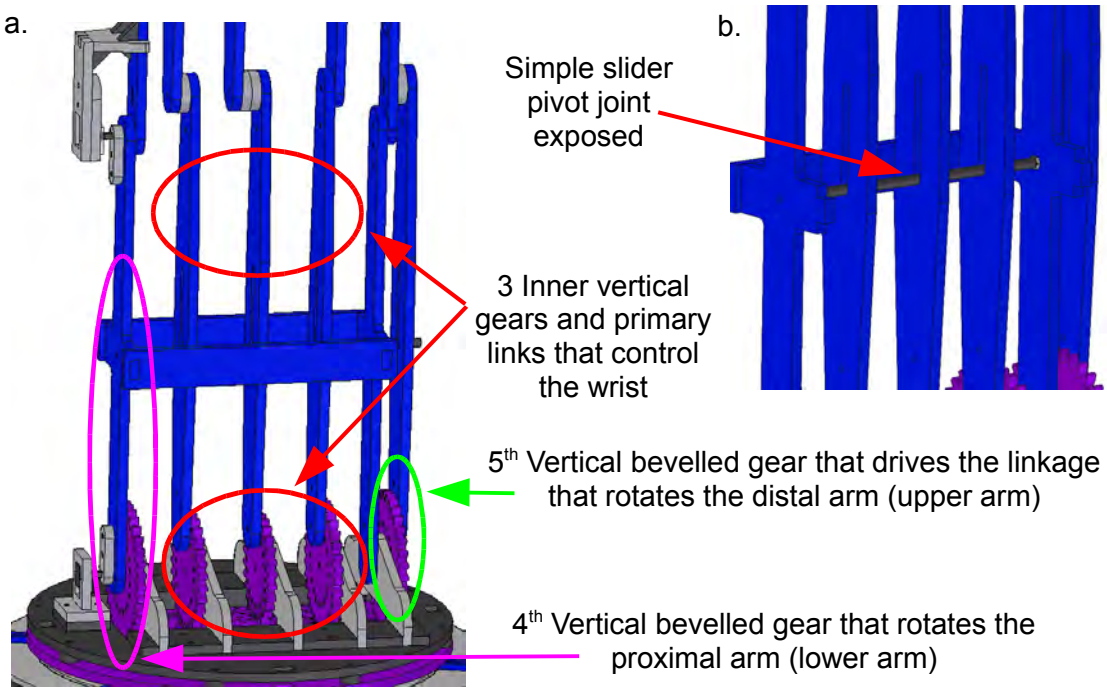


Figure 59: Lower arm

a. Vertical gear connections to links b. Simple slider pivot joint

The 3 inner vertical bevelled gears which were driven by the concentric gear drive, served as the driver links for the primary slider-pivot linkages. Those outputs then drove the

secondary slider-pivot linkages that eventually controlled the orientation of the wrist through the wrist concentric drive (figure 59a, b). The fourth vertical bevelled gear controlled the proximal arm (lower arm) whose midpoint held the slider-pivot axis for the primary links (figure 59). It controlled the elevation of the lower arm (proximal arm) with respect to the horizontal plane. The fifth gear was the input of a slider-pivot linkage whose follower controlled the angle between the proximal and distal arms. The distal arm held the slider-pivot axis for the secondary links (figure 60), which was at its mid-point. This reduced warping of the secondary stage follower orbit.

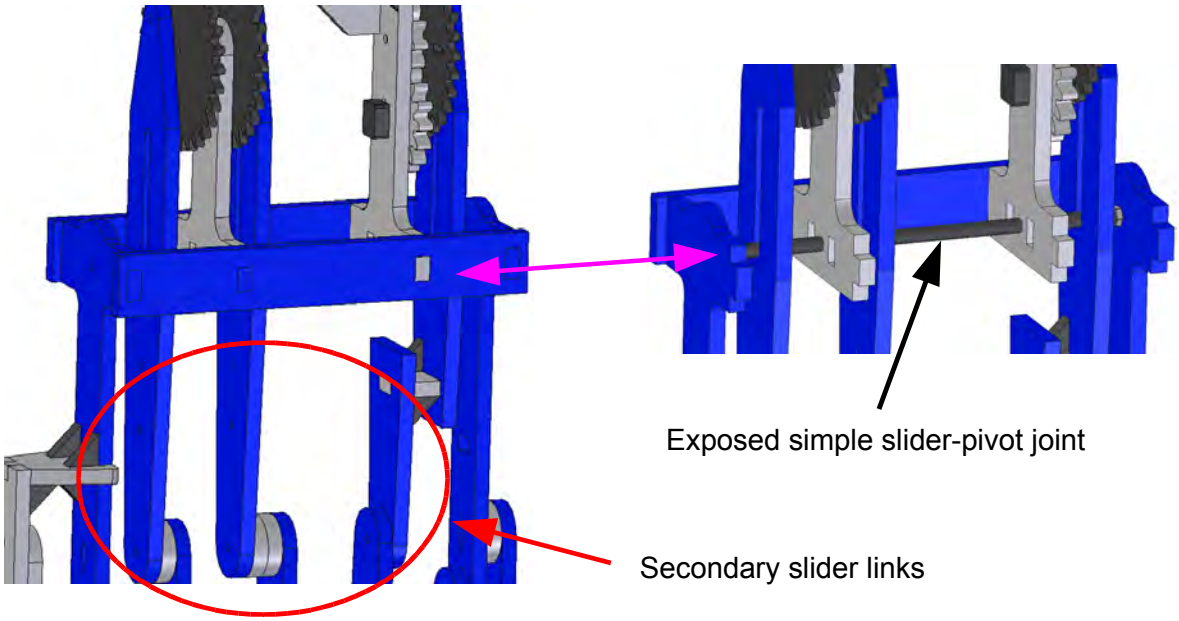


Figure 60: Upper arm

a. Upper arm showing secondary slider bar followers connecting to wrist gear drivers
b. Upper arm showing slider pivot joint

3.2.3.2. Geared torque transfer linkage used with second BCGM

To transfer actuation away from the second BCGM additional bevel geared assemblies were used. These geared mechanisms transferred torque on axes that were parallel to the major links of the robot (the proximal and distal arms), in any robot configuration. This geared solution provided a larger range of motion than universal (Hook joints) or constant velocity

joints, and is shown in figure 61. The gears are labelled G_a, G_b and G_c, with their corresponding axes having labels G_a axis, G_b axis and G_c axis. Axes G_a and G_b are perpendicular to the G_c axis. This geared assembly allowed torque transfer from G_a to G_b for any angle between the G_a and G_b axes.

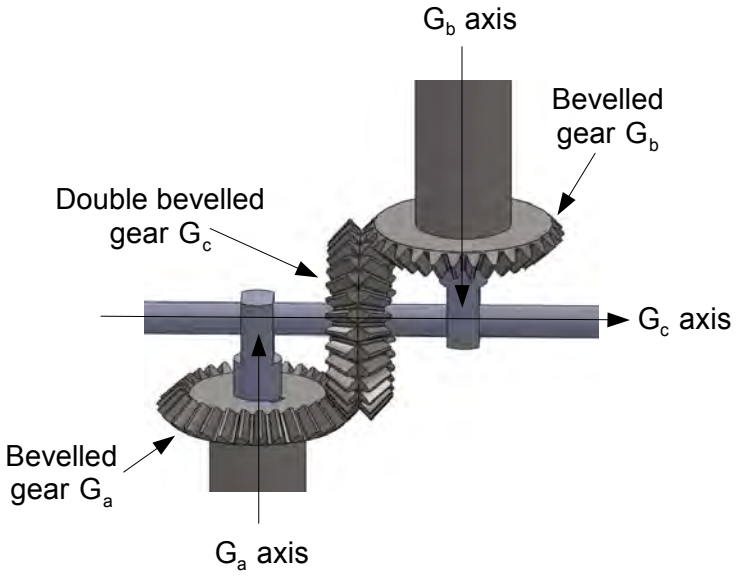


Figure 61: Geared mechanism for torque transfer in any robot configuration

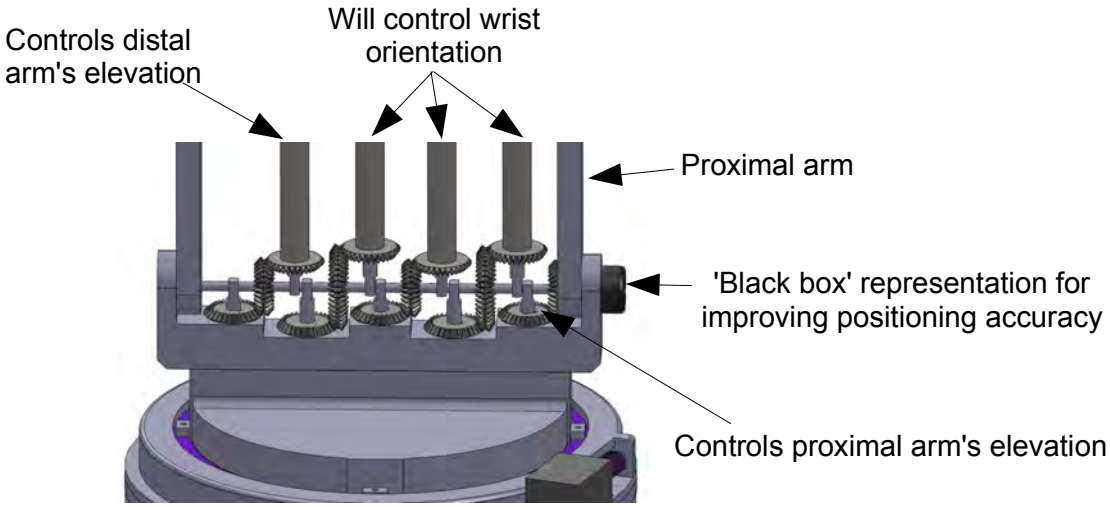


Figure 62: Illustration of components that control links and axes

The 3 inner bevelled gears driven by the BCGM, served as the drivers that eventually controlled the orientation of the wrist through the wrist concentric drive (figure 62). The fourth bevelled gear controlled the elevation of the proximal arm with respect to the horizontal plane. The fifth gear was the input of the geared linkage that controlled the angle between the proximal and distal arms, see figure 63.

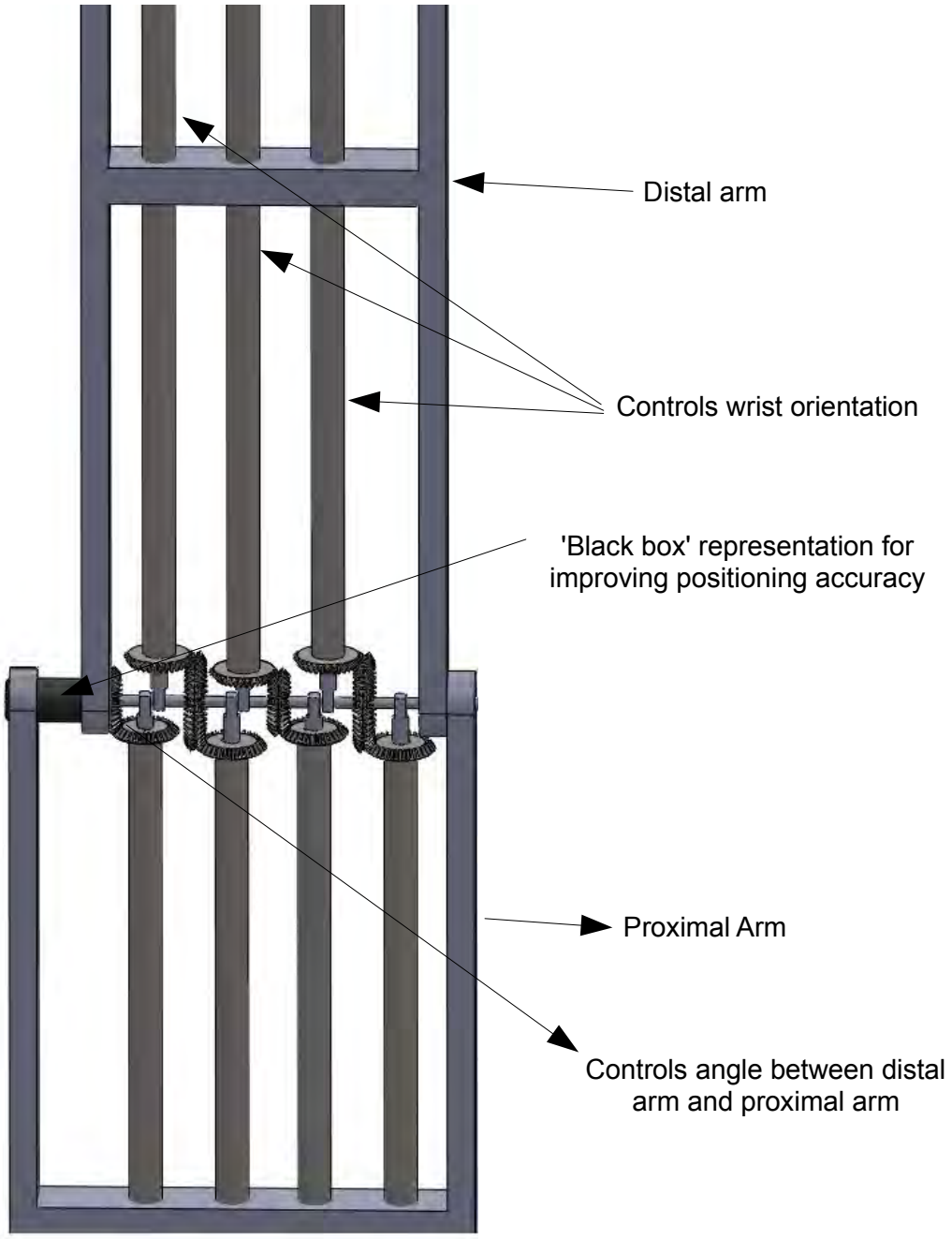


Figure 63: Design 2's proximal and distal arms

In figures 62 and 63 the 'black box' representation for 1 of the methods used to improve positioning accuracy was indicated. For figure 63, if it represented a harmonic drive then the flexspline cup would be connected to the distal arm, the rigid circular spline would be fixed to the proximal arm and the horizontal bevelled gear would be fixed to the wave generator. Similarly for other 'black box' representations, and accuracy improvement methods.

3.2.4. Wrist designs

The wrist designs were meant to be spherical with all 3 axes concurrent at the sphere centre, and as mentioned previously that decoupled wrist position kinematics from wrist orientation kinematics. The wrist designs had similar gearing mechanisms to their respective bases which facilitated the distribution of torque to the designated wrist axes.

3.2.4.1. Wrist for design 1

The follower end points of the 3 inner secondary stage slider-pivot linkages connected to 3 vertical bevelled gears (all 3 axes concentric) respectively (which were mounted on the upper arm). Since the follower did not have a perfect circle orbit around that axis slots were cut into the gears which allowed the follower to move in and out of that perfect circle trajectory (figure 64d). Those vertical concentric bevelled gears then meshed with a concentric drive gearbox which had 4 concentric sections shown in figure 64a. This wrist concentric drive used a similar double ball race bearing illustrated in figures 45 and 46, which allowed each section to move independently of each other. The outer sections held the inner sections in place. Either the outermost or innermost section could be held in place by the upper arm. In this design the innermost section of the wrist gearbox was supported, and is shown in figure 64a. The outermost movable section (or the third section in the concentric gear drive for the wrist) rotated the wrist (this was the fourth robot axis) and had mountings for the inner 2 axes of the 3 DOF wrist. With some additional gearing the fifth robot axis was set at 90º to the fourth axis, and the sixth axis was set at 90º to the fifth. Note that the fourth and sixth axes were not necessarily perpendicular to each other, also that a spherical wrist was desired. The actual design in figure 64 was not spherical due to the positioning of sensors for the scaled model

that would be built, and the requirements for a compact design. Since the robot was only controlled in the forward kinematics sense this did not present a big problem.

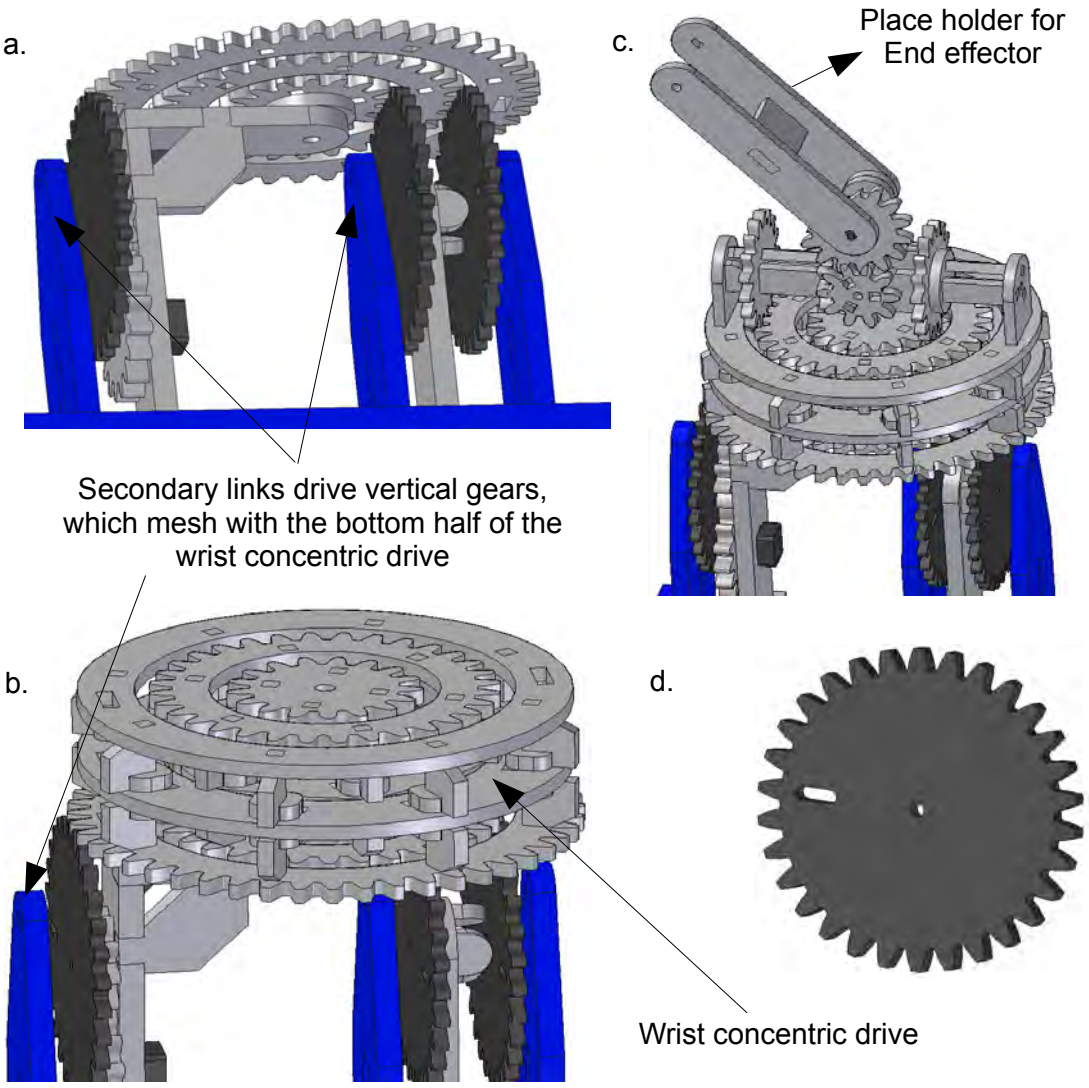


Figure 64: Design 1's 3 DOF wrist

a. Gear meshing at bottom of wrist concentric gearbox
b. Top view of wrist concentric gearbox
c. Complete wrist
d. Secondary follower gear with slot

3.2.4.2. Wrist for design 2

The spur gears at the end of the 3 inner geared linkages then meshed with the wrist concentric drive gears. It had 4 concentric sections as shown in figure 65a. This concentric drive again used the double ball race bearing design illustrated in figures 51 to 54, which allowed each section to move independently of each other. It also had top and bottom covers that held the sections in place. This was a proper 3 DOF spherical wrist. In this design the outermost section of the wrist gearbox was fixed to the distal arm. The innermost movable section rotated the wrist (or controlled the azmuth with respect to the distal arm, this was the fourth robot axis or wrist axis 1). The second section controlled the elevation of the end-effector link with respect to the distal link (wrist axis 2). The last section rotated the end effector about the sixth axis (wrist axis 3) defined by the previous 2 wrist axes.

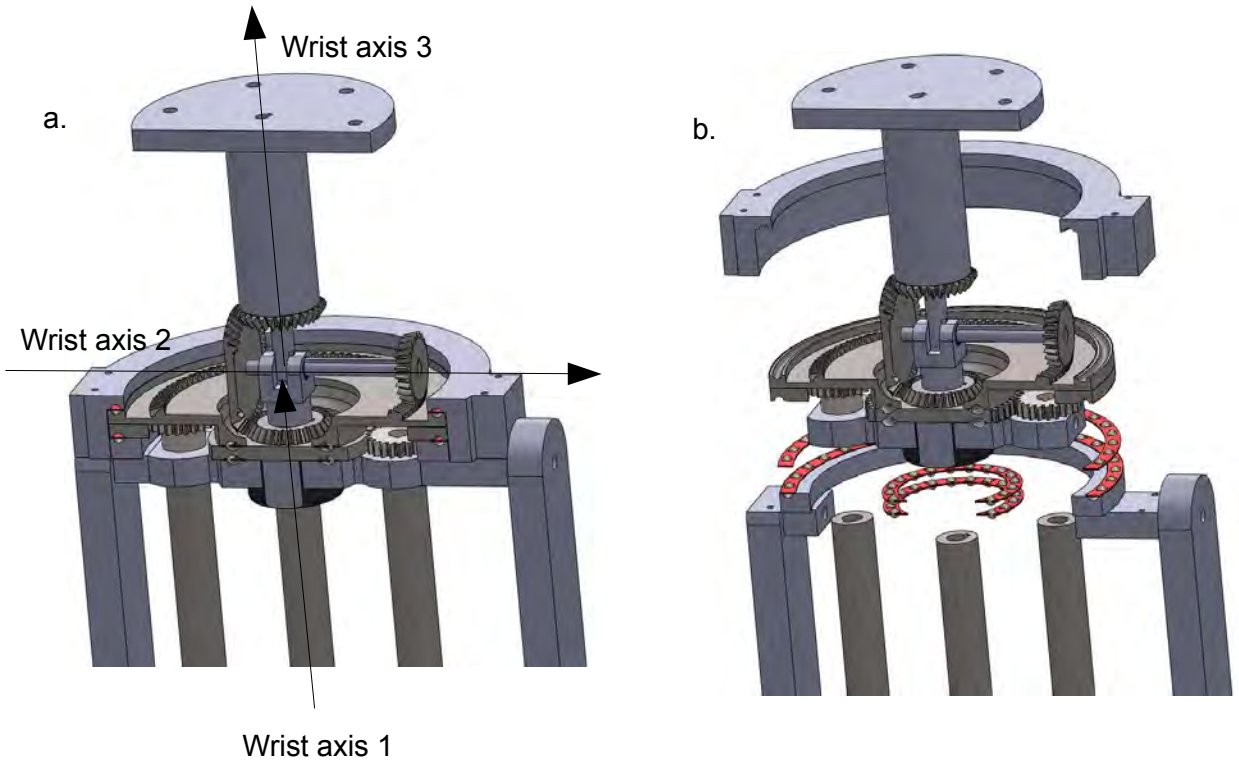
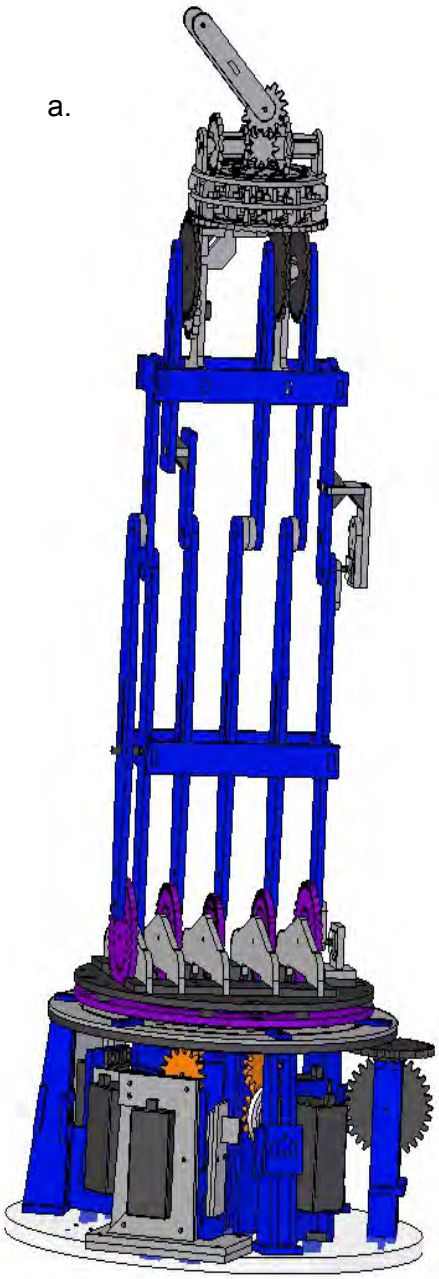


Figure 65: Design 2's 3 DOF spherical wrist

a. Wrist indicating axes b. Exploded view of wrist

3.2.5. Complete designs

a.



b.

Figure 66: 6 DOF large workspace, low inertia machine designs

a. Complete first design concept b. Complete second design concept

These were the 2 concept designs in full. A scaled model of the first design was built and controlled in the forward kinematics sense only. A full inverse kinematics simulation was performed for design 2 and rendered in Scilab and SolidWorks. Videos of each are in the attached CD.

3.3. Design classification

These designs had characteristics of both SKMs and PKMs. Firstly in both designs all the driving motors were located at the fixed base, a characteristic of PKMs. The kinematic modelling on the major structure of the machine however followed a simplified serial robot, as will be explained in section 3.5.2 machine level kinematics, and hence the error propagation was also serial. If a graph theory approach was used to classify the designs then the graph for both looked more like an inverted tree, call this a root, with the ending of each root being the fixed motor. This was a significant difference from tree graphs (which were serial) that have the ends of each branch free. The inter-connecting links of the first design were also reminiscent of an inter-connected chains PKM. Furthermore the wrist and BCGMs, connected to the inter-connected links (parallel) serially, which gave it a hybrid nature. The argument follows similarly for the second design. Thus since the designs exhibited characteristics of both architectures a hybrid classification was settled on.

3.4. Prior art search

Kanayama (2010) [84] described an actuated concentric double axis mechanism. The invention used 2 sets of pulleys, belts and a mating pair of bevelled gears to force a 90º arrangement of the driven axes. This design had the disadvantage in that the platform holding the second driven axis (numbered 10 in figure 67) could not make a full 360º rotation (i.e. axis 1, or the vertical axis) as it would collide with the belt mechanism driving the second axis (numbered 16 in the figure). Additionally this mechanism was restricted to only 2 driven axes.

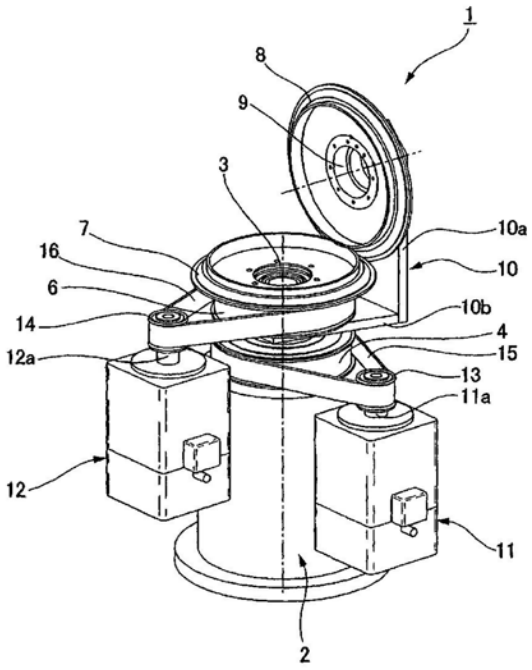


Figure 67: Concentric double axis mechanism
from Kanayama (2010) [84]

A US patent from *Yamazoe (2003)* [85], titled “Horizontal multi-joint industrial robot”, described a planar robot that used belts and pulleys (stated as driving force transmitting mechanism) to transfer rotational torque from motors at a stationary base to designated joints. It could have an additional fourth DOF that could rotate a tool or adjust the height of that tool relative to the base. This design modified the traditional SCARA robot by reducing its inertia making it more agile, and eliminating the need for electrical wiring along the arm. The patent specifically mentioned that the axes at the base had to be concentric.

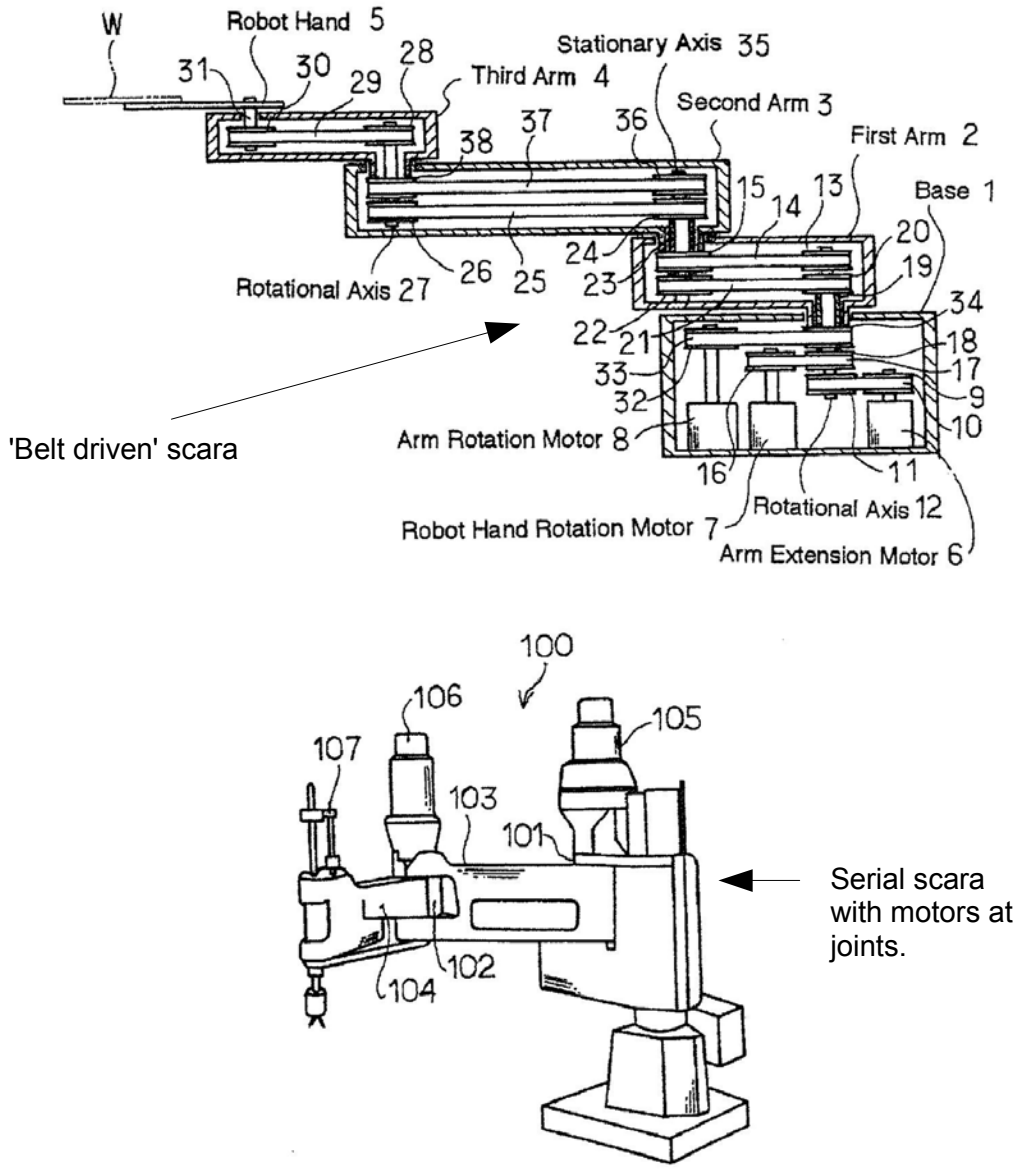


Figure 68: Horizontal multi-joint industrial robot
from Yamazoe (2003) [85]

A European patent titled “Multi-articulated industrial robot with several degrees of freedom of movement”, from *Bisiach (1989)* [86] described a robot that used hollow coaxial shafts and concentric geared mechanisms. This design had 7 DOFs, 3 DOFs to orient the tool and 4 DOFs to position it.

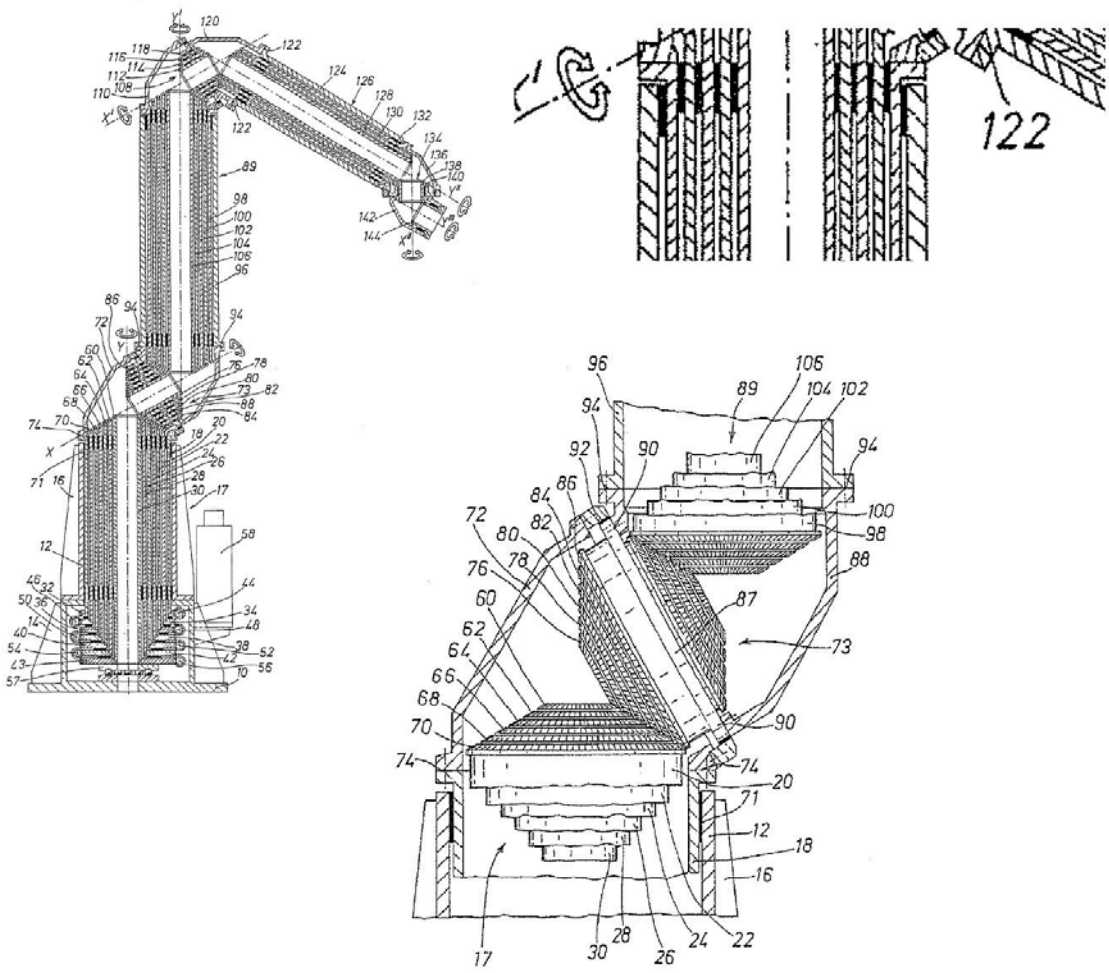


Figure 69: Multi-articulated industrial robot
from Bisiach (1989) [86]

Another German patent, “Robot arm with multiple links has drive transmitted to links from individual motors through coaxial shafts”, from *Pozzi (1989)* [87] also described a robot that used coaxial shafts and concentric geared mechanisms. This design also had 7 DOFs, with 3 for the orientation of a wrist and 4 to position the wrist centre. The gripper motor was also positioned at the base, but that did not constitute a robot DOF.

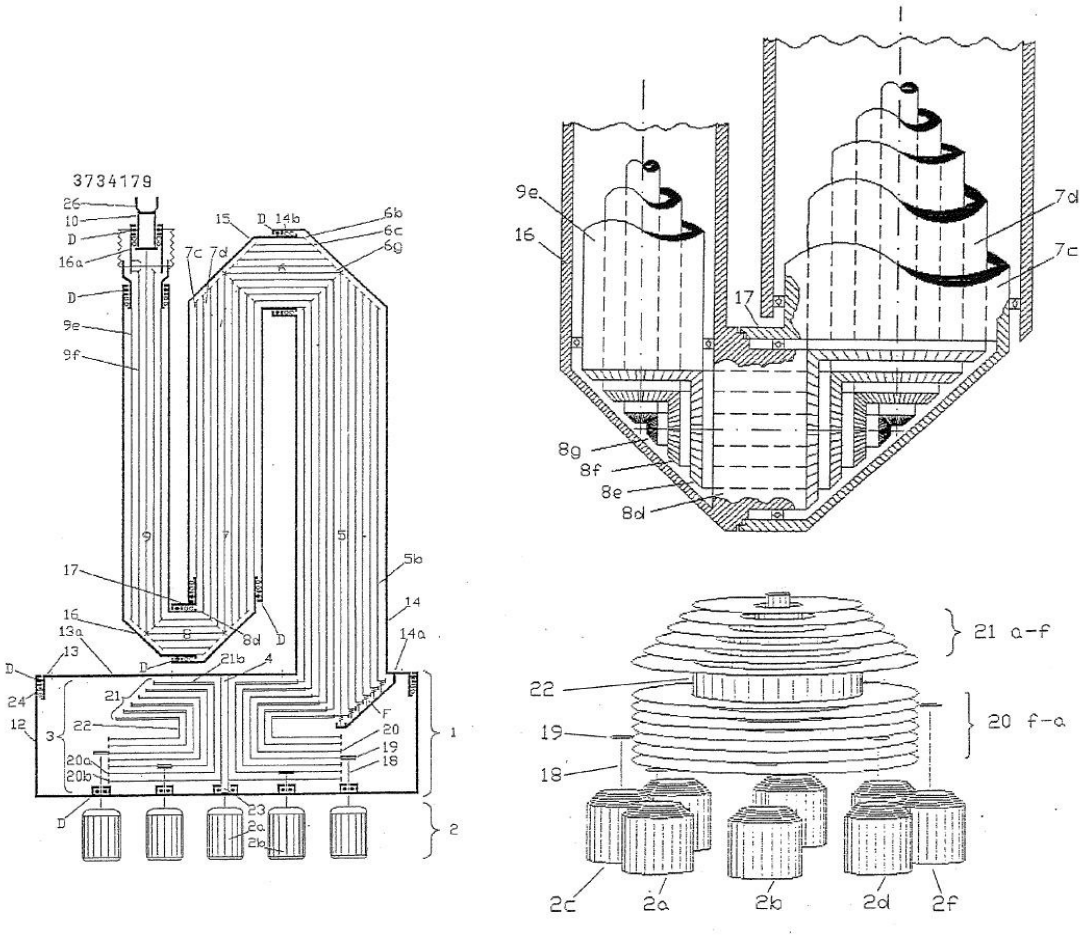


Figure 70: Robot arm with multiple links and coaxial shafts
from Pozzi (1989) [87]

The last 2 patents were most applicable as they had more than 6 DOFs and could achieve the same workspace but with greater dexterity (these were redundant robots). However the seventh motor and associated links and connections increased manufacturing and running costs.

Both designs were forced to use a 2 DOF elbow joint to achieve the same motion trajectory, since the geared mechanisms at the elbow were set at 45º. This was to avoid an awkward location of the elbow joint which could lie to the side of a vertical plane passing through the wrist and base origin instead of lying on that plane). The designs which are the subject of this thesis (now called subject designs) have a 2-1-3 structure, that is 2 DOFs at the shoulder, 1 DOF at the elbow and 3 DOFs at the wrist. The patents mentioned were both 2-2-3 designs.

For these designs to work the driven axes had to be brought up in a concentric manner, and

the only way to do this was through concentric hollow tubes. Comparatively these designs also used a greater number of unique parts which would increase manufacturing cost.

The tubes also used more material and had a greater moment of inertia (increased with tube radius), which meant that comparatively they would consume more energy, and would also be slower using the same motor specifications. Furthermore since everything had to be concentric special custom bearings had to be used, and the assembly of the concentric units would be difficult. In the subject designs the torque transfer links come up from the concentric base parallel to one another (either in parallel planes as with the first design or parallel lines in the second design).

From the drawings the concentric geared systems in these designs used concentric tubes, again difficult to manufacture and assemble. The subject designs used parts that bolt together and support outer geared sections.

Lastly these designs have no room to incorporate high accuracy speed reducers (harmonic and cycloidal drives), or the locking type mechanisms at the main axes to improve end-effector placement accuracy. Hence the subject designs could be more accurate.

Summary of improvements over current patents

1. The designs described in this thesis were more practical and cost effective to manufacture:

    - They used less material and 1 less actuator.
    - Moved less mass (also had a better mass distribution) and had less actuators which reduced energy consumption.
    - Concentric tube design would be difficult to assemble in practice.
    - Used fewer unique components.
    - Was easier to maintain and operate.

2. The torque transfer links that were used were either in parallel planes or parallel lines, they were not concentric.

3. The 7 DOF robot designs were forced to use 2 DOF elbow joints to have a similar motion

trajectory of a typical 6 DOF robot.

4. The subject designs could achieve higher accuracy.

3.5. System modelling

The system modelling that follows will describe the kinematics and force transfer for the primary 3 bar slider-pivot linkage used (the discussion could be extended for the secondary 3 bar slider-pivot linkage, but that was omitted). The machine level kinematics used the same simplified geometric model for both designs, and the reason for this will be explained in the section of that title. The second design used rotating members for the torque transfer linkages, and the COMs of those links moved in unison with the major link holding it in place, hence their kinematics could be derived from the machine level kinematics with some offsets. It was thus omitted.

3.5.1. 3 Bar linkage

The modelling of the 3 bar linkage described here, together with the kinematic solutions were used in the simulations to determine the best position for the slider-pivot joint. Each link was reduced to its most simplified geometry, a straight line connecting the rotary joints at its ends. The rotary joints were illustrated by bold dots, and the slider-pivot joint could only illustrate the rotational part of that joint. The sliding action was implied by the geometry, it was not illustrated explicitly. Figure 71 depicts this simplification, superimposed on the side view of a 3D illustration.

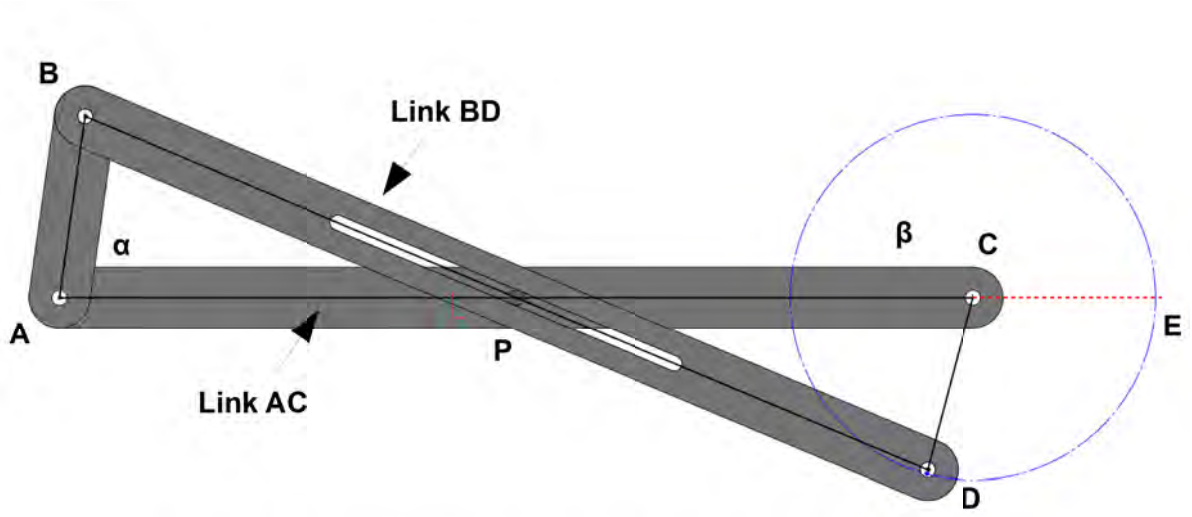


Figure 71: Geometric simplification of 3 bar slider-pivot linkage

3.5.1.1. Forward kinematics

The following notation was used to describe the kinematic solutions. Referring to figure 71, consecutive joint lettering indicated the length between said joints or referred to the link unless otherwise stated, e.g. AB denoted the length of link AB. The order of the letters was irrelevant, AB was the same as BA. Angles were always referred to in an anti-clockwise sense, hence angle PAB which will now be called α, extended from link PA (partial section of link CA or AC), in an anti-clockwise direction to link BA (or AB), and angle ECD which will now be called β, extended from link EC to DC in an anti-clockwise direction.

In this particular case link AC is parallel to the x-axis (for convenience the linkage lies on the xz plane, and A is the origin). Actually the input angle α and the output angle β were with respect to the positive x-axis. If link AC were at some angle θ to the x-axis then the orbit traced by D as α varies from 0° to 360° was the same as the orbit traced by D when link AC was parallel to the x-axis rotated by θ in an anti-clockwise direction about the y-axis. The shape of the orbit does not change with respect to link AC. Hence for simulations of this linkage it was sufficient to have link AC in its current state, parallel to the x-axis.

The xz coordinates of B and P were given by $(AB.\mathbf{cos}\alpha;\ AB.\mathbf{sin}\alpha)$ and $(AP;\ 0)$ respectively. To obtain the coordinates of D, the gradient of BD was first needed given by

$\frac{AB.\sin\alpha}{AB.\cos\alpha - AP} = m$. There were limits imposed on AP but AP would always be greater than $AB.\cos\alpha$, and the gradient of BD was always defined. Additionally for an undefined gradient BD would have to be parallel to the z-axis. The coordinates of D' (not shown), which was on a line through joint A parallel to link BD, was given by $(t, m.t)$ and t satisfied the distance criterion $BD^2 = t^2 + (m.t)^2 = t^2(1+m^2)$. Hence $t = \sqrt{\frac{BD^2}{(1+m^2)}} = \frac{BD}{\sqrt{(1+m^2)}}$ and was positive since its position was to the right of the origin A (the positive x-axis). To get the coordinates of D, the line AD' was translated to BD by adding the coordinates of B to both A and D'. Hence the coordinates of D were $\left(AB.\cos\alpha + \frac{BD}{\sqrt{1+m^2}};\ AB.\sin\alpha + \frac{m.BD}{\sqrt{1+m^2}}\right) = (q;\ r)$. The coordinates of D was used together with that of C to generate an output angle β. The line DC was translated such that C was positioned at the origin, this implied subtracting $(AC;\ 0)$ from both sets of coordinates, and resulted in line D"A (not shown). The coordinates of D" were $\left(AB.\cos\alpha + \frac{BD}{\sqrt{1+m^2}} - AC;\ AB.\sin\alpha + \frac{m.BD}{\sqrt{1+m^2}}\right) = (q - AC;\ r)$. β was found by using the arctan function and by noting the quadrant in which D" was, hence

$$\beta = \arctan\left(\frac{AB.\sin\alpha + \frac{m.BD}{\sqrt{1+m^2}}}{AB.\cos\alpha + \frac{BD}{\sqrt{1+m^2}} - AC}\right) = \arctan\left(\frac{AB.\sin\alpha\,.\sqrt{1+m^2} + m.BD}{AB.\cos\alpha\,.\sqrt{1+m^2} + BD - AC.\sqrt{1+m^2}}\right).$$

$$\beta = \arctan\left(\frac{r}{q - AC}\right)$$

3.5.1.2. Inverse kinematics

For the inverse kinematics the output angle was specified, and the input angle that places the linkage in that configuration was required, in other words β was given (now the input angle) and α had to be calculated (now the output angle). Specifying the coordinates for D, would only result in a solution if D was somewhere on that orbit and hence was a bad choice for calculation. There were no singularity points with regard to the output angle, its range was a full 360º. Singularities would be gaps in the range of motion, or points that the linkage could

not reach. No closed form solution was found for the inverse problem. β was known (and hence $\mathbf{tan}\beta$) and α had to be calculated from the formula $\mathbf{tan}\beta = \frac{AB.\mathbf{sin}\alpha.\sqrt{1+m^2}+m.BD}{AB.\mathbf{cos}\alpha.\sqrt{1+m^2}+BD-AC.\sqrt{1+m^2}}$ where $m = \frac{AB.\mathbf{sin}\alpha}{AB.\mathbf{cos}\alpha - AP}$ (these were derived for the forward kinematics solution). The best that could be done was a numerical solution, which came at a cost of reduced accuracy and increased computation time. There are a number of approaches for the numerical solution, the simplest will be described here. Since memory (data storage) is cheap, a lookup table was stored so that a very rough solution could be obtained from an indexed solution set. The index consisted of 361 entries and ranged from 0 to 360 (whole numbers only). This index represented the angle β rounded to the nearest whole number. Each indexed input angle β had a corresponding output angle α, that would place the linkage in that configuration. Those corresponding sets were found using a forward kinematics search, and then stored. The fractional part of β was then used to make another approximation, it was used to determine the fractional part that would be added to the output angle α to obtain the starting approximation. This starting approximation was then fed as an input angle to the forward kinematics solution to get the actual output of that driving angle. If the error between this and the original input angle was less than some defined tolerance then the solution was acceptable. If not then an adjustment was made and the algorithm repeated until the error was less than the defined tolerance. The flowchart for this algorithm is shown in figure 72.

The fact that the inverse kinematics could not be solved explicitly had consequences for all the modelling that follows. It meant that the inverse force, torque and dynamics could not be solved explicitly as each of these relied on the inverse kinematic solution of the linkage configuration. The algorithm for a numerical solution mentioned above would have to be used for all calculations and there would be performance and accuracy issues.

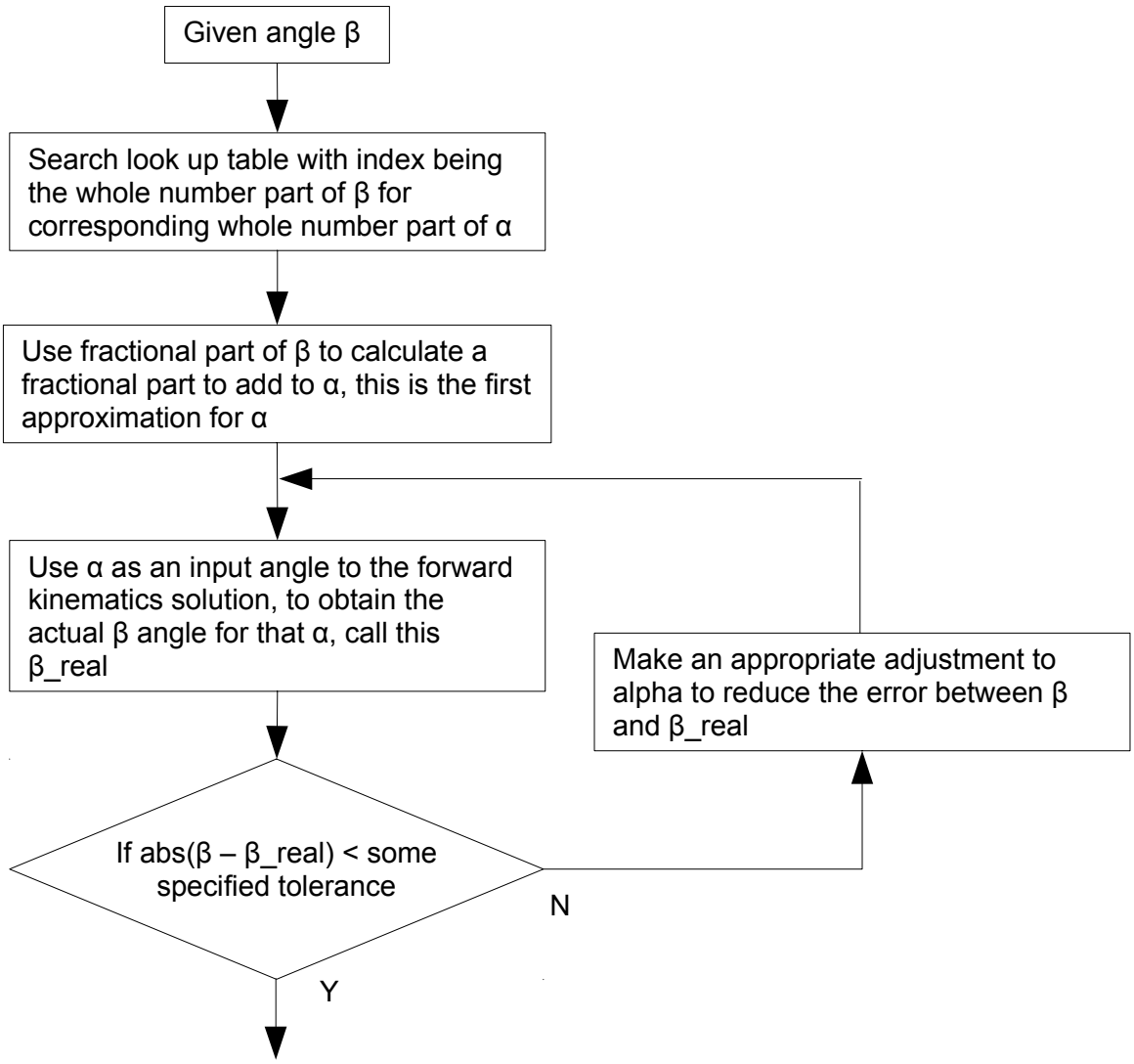


Figure 72: Flowchart solution for the inverse kinematics of the 3 bar linkage

3.5.1.3. Torque/force transfer

This discussion for force transfer ignored the effects of link masses and losses. Joint P was a slider-pivot joint, it allowed link BD to slide which changed the distance BP. Link BD could also rotate (or pivot) about P. Figure 73 depicts the force modelling through the linkage, with a driving input torque **T** (given value). Force $\mathbf{F_1}$ was perpendicular to link BA, and was derived from the torque equation i.e. $\mathbf{F_1} = \mathbf{T}/\mathrm{AB}$. In order to gauge how that force was transferred through the linkage to point D, it was split into components that were parallel (in

this case collinear) and perpendicular to link BD. $\mathbf{F}_{1Per}$ was the component of $\mathbf{F}_1$ that was perpendicular to BD and $\mathbf{F}_{1Col}$ was the component that was collinear to BD, similarly for $\mathbf{F}_{2Per}$ and $\mathbf{F}_{2Col}$. The reason behind breaking the force into parallel and perpendicular components about BD was that there were no cross contributions, meaning that the perpendicular force components on one side of pivot P could not contribute to the parallel components on the other side of P, and vice versa. Additionally the action of the parallel and perpendicular components were quite clear, and easily understood from a visual and computational standpoint.

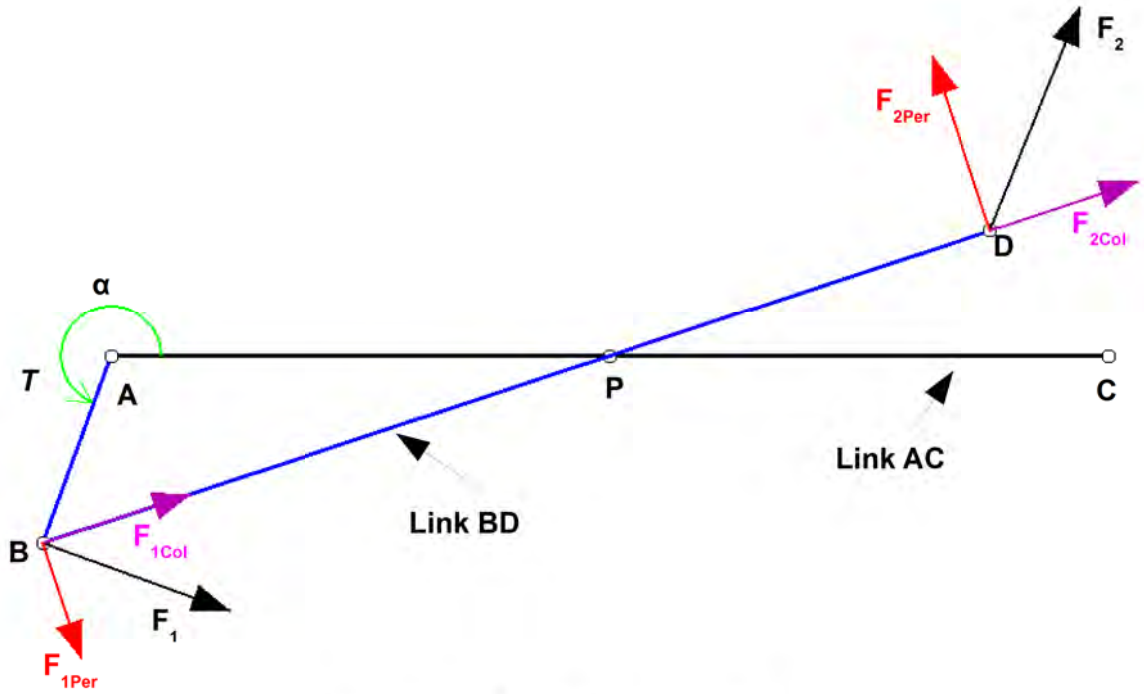


Figure 73: 3 Bar linkage force transfer

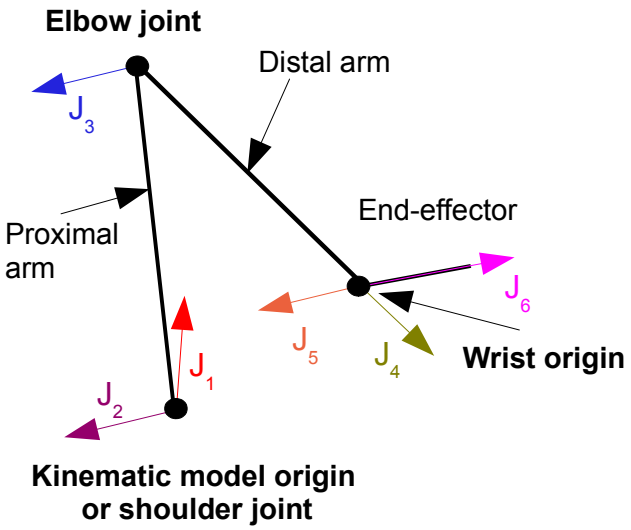
$\mathbf{F}_2$ was the resultant output force and was the vector sum of $\mathbf{F}_{2Per}$ and $\mathbf{F}_{1Col}$. $\mathbf{F}_{1Col}$ was transmitted from B through link BD to D without any modification, i.e. $\mathbf{F}_{1Col} = \mathbf{F}_{2Col}$. This was from the principle of transmissibility, which stated that a force could act at any point on its line of action on a body without altering the net effect caused to that body, from *Meriam and Kraige (2007)* [88]. $\mathbf{F}_{1Per}$ on the other hand was altered due to the pivoting and sliding action of P. The direction was changed due to the rotating action of P and the magnitude depended on the length ratio of BP and PD. In other words the torque at P created by $\mathbf{F}_{1Per}$ acting at B

had to equal the torque at P by $\mathbf{F}_{2Per}$ acting at D, so $\mathbf{F}_{1Per}.BP = -\mathbf{F}_{2Per}.DP$ or $\mathbf{F}_{2Per} = -\mathbf{F}_{1Per}.DP/BP$. Hence the output force contribution due to the torque **T** was given by $\mathbf{F}_2 = \mathbf{F}_{2Per}+\mathbf{F}_{2Col} = -\mathbf{F}_{1Per}.DP/BP+\mathbf{F}_{1Col}$. The torque generated around C could then be calculated as the cross product of that force $\mathbf{F}_2$ and vector CD.

3.5.2. Machine level kinematics

Bonev (2002) [13] highlighted the effectiveness of geometric methods for the design and analysis of parallel mechanisms, and stated that geometry brought insight into the principles of motion better than algebraic or numerical methods and developed creativity and intuition. With regard to spatial parallel mechanisms, the need was more pronounced than for planar ones as the added dimension increased the complexity of the algebraic expressions for motion. A geometric analysis could reduce the size of those expressions, as the essential geometric interpretation of the motion was almost impossible if based purely on the analysis of algebraic expressions.

The same could be said of serial and hybrid mechanisms, hence a mixed geometric algebraic approach was used to describe the kinematics of this machine. The machine level kinematics was simplified by only considering the main links in the 2 designs, the kinematics of all driving links were ignored as those links were used to control the angle of each of the 6 main axes for each robot pose. Sensors could be placed at the 3 main joints of the spherical wrist, as well as the elbow and the shoulder joints (2 DOF), and the control system could monitor those joint angles. The kinematic model geometry thus followed a serial model as shown in figure 74, and was applicable to both machine designs.



J_i stands for joint axis i. J_1 is vertical; J_2 is perpendicular J_1; J_3 is parallel J_2; J_4 is perpendicular J_3 and collinear to the distal arm representation; J_5 is perpendicular J_4; J_6 is perpendicular J_5; J_4, J_5 and J_6 are concurrent at the wrist origin.

Figure 74: Simplified geometric model of machine designs

3.5.2.1. Forward kinematics

The forward kinematics defined the value of the joint angles between the robot's significant links and then required the calculation of the end effector's spatial position and orientation. The forward kinematics was usually a 1 to 1 or many to 1 type function, however there were some mechanisms which allowed an infinite number of solutions for a particular input to the direct/forward kinematics. This situation was referred to as 'self motion' and resulted in finite mobility for some points of the workspace. It was also called a type 2 or redundant output (RO) singularity, from *Bonev (2002)* [13]. The serial robot and model used excluded this type of singularity. The Denavit-Hartenberg method of coordinate frame matrix multiplication was used to evaluate the forward kinematics. For a concise treatment on the method see *Craig (2005)* [89]. The model parameters are listed in table 2, where α_{i-1} and θ_i were measured in degrees and $\mathbf{a}_{i-1}$ and $\mathbf{d}_i$ were measured in mm.

Table 2: Denavit-Hartenberg robot model parameters (degrees, mm)

i	α_{i-1}	a_{i-1} [1]	d_i	θ_i
1	0	0	0	θ_1
2	90	0	0	θ_2
3	0	1000	0	θ_3
4	270	1066.29	0	θ_4
5	270	0	0	θ_5
6	180	181.29	0	θ_6

The above parameters were used in the following general matrix transform ${}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and gave the 6 transform matrices.

$${}^{0}_{1}T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad {}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad {}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 1000 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$${}^{3}_{4}T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 1066.29 \\ 0 & 0 & 1 & 0 \\ -s\theta_4 & -c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad {}^{4}_{5}T = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s\theta_5 & -c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$${}^{5}_{6}T = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & 181.29 \\ -s\theta_6 & -c\theta_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position and orientation of the end-effector with respect to the base frame (${}^{0}_{6}T$) was obtained from the multiplication ${}^{0}_{1}T . {}^{1}_{2}T . {}^{2}_{3}T . {}^{3}_{4}T . {}^{4}_{5}T . {}^{5}_{6}T$.

1 Note that the subscript values are 1 less than the index, so for index 3 a_{i-1} was $a_2 = 1000$ and for index 4 a_{i-1} was $a_3 = 1066.29$.

3.5.2.2. Inverse kinematics

The inverse kinematics problem (IKP) was to compute the joint angles from the end-effector coordinates and orientation. Generally there were several valid sets of joint angles for each end-effector pose. When solutions to the IKP of a chain degenerates or coincides, the pose was classified as a type 1 singularity, also referred to as a redundant input (RI) singularity. The position and orientation were given by $P=\begin{bmatrix} x \\ y \\ z \end{bmatrix}$, and $R=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ respectively.

Since the robot design facilitated a decoupling of the end-effector position and orientation, these will be solved separately. The wrist was spherical with all wrist axes concurrent, and the point of concurrency was the wrist origin, sphere centre or wrist centre (O_{wc}). The first step was to evaluate this point, which was computed by translating backwards along the x-axis (negative x direction) of the given wrist orientation R, by an amount of a_5 starting at coordinate P, i.e. $O_{wc} = R.\begin{bmatrix} -a_5 \\ 0 \\ 0 \end{bmatrix} + P = \begin{bmatrix} x_{wc} \\ y_{wc} \\ z_{wc} \end{bmatrix}$. It was essential to test whether O_{wc} could physically reach that point, by computing the distance to the global origin O_g, which should be less than $a_2 + a_3$. If the result was greater, the given point was outside the workspace. If it was equal to $a_2 + a_3$ then the point was on the workspace boundary (a RI singularity), at this point the proximal and distal arms were parallel and the robot arm was fully outstretched. If the result was less than $a_2 + a_3$, the computed O_{wc} was valid and the inverse kinematics for the position could be solved, i.e. the first 3 angles that position the wrist at that point could be calculated.

Next angle 1 (θ_1) was calculated, which was the angle the movable part of the robot arm rotated about the vertical. This was done by projecting O_{wc} onto the XY-plane (call the vector $_{xy}\hat{O}_{wc}$, which used the x_{wc} and y_{wc} values, the z coordinate was 0) and calculating the angle it made with the reference X-axis of the global reference frame. If the length of $_{xy}\hat{O}_{wc}$ was 0, then O_{wc} was on the fixed frame's Z-axis, and there were an infinite number of values for θ_1 which made it a singular position. If the length of $_{xy}\hat{O}_{wc}$ was greater than 0, θ_1

could be solved by first evaluating its sine and cosine and then using the inverse trigonometric functions to get the angle. Since both the sine and cosine values of the angle were available, there would always be a unique solution in the range 0º to 360º. Hence $\mathbf{sin}(\theta_1) = \frac{y_{wc}}{\|{}_{xy}\hat{O}_{wc}\|}$ and $\mathbf{cos}(\theta_1) = \frac{x_{wc}}{\|{}_{xy}\hat{O}_{wc}\|}$, the arc-sine and arc-cosine functions were applied to each, and the common solution from the results was selected as the correct answer.

The elbow joint coordinates could then be calculated, but first the calculation would be simplified by working on a plane instead of 3D space. Using θ_1 which was just calculated, the wrist origin O_{wc} in the global reference frame was rotated about the reference frame's Z-axis by negative θ_1, to lie on the XZ-plane (${}_{xz}\hat{O}_{wc}$, the resulting y-coordinate was 0 and ${}_{xz}z_{wc} = z_{wc}$), i.e. ${}_{xz}\hat{O}_{wc} = \begin{bmatrix} {}_{xz}x_{wc} \\ 0 \\ {}_{xz}z_{wc} \end{bmatrix} = \begin{bmatrix} c(-\theta_1) & -s(-\theta_1) & 0 \\ s(-\theta_1) & c(-\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{wc} \\ y_{wc} \\ z_{wc} \end{bmatrix}$, in this context s and c are the short forms for sine and cosine respectively and will be used from this point on. Once the coordinates were found on the XZ-plane, angle 2 (θ_2) and angle 3 (θ_3) could be calculated. The angle between the proximal and distal link representations, as well as the elevation of the proximal link were unchanged, which was the critical aspect. Once that was done, the XZ-plane elbow joint (${}_{xz}E_{ej}$) coordinates was rotated back by positive θ_1 to obtain the correct 3D elbow joint coordinates (E_{ej}).

The elbow joint coordinates lie at the intersection of 2 circles, one centred at the base origin O_g and the other at the XZ wrist coordinates ${}_{xz}O_{wc}$, with radii $a_2=1000$ and $a_3=1066.29$ respectively. The circles generally intersect at 2 points ${}_{xz}E_{ej1}$ and ${}_{xz}E_{ej2}$ shown in figure 75a. The case where the proximal and distal links were collinear was accounted for, and in this interpretation the circles are tangent (${}_{xz}E_{ej1}$ and ${}_{xz}E_{ej2}$ coincide and the arm is fully outstretched).

Another modification to simplify calculations was to rotate vector ${}_{xz}\hat{O}_{wc}$ about the reference frame Y-axis (by angle negative ψ in figure 75a, which was calculated in the same

manner as θ_1), such that it was positioned on the reference frame X-axis and was named ${}_xO_{wc}$. The y and z coordinates of ${}_xO_{wc}$ were both 0, and its x coordinate was given by the length of ${}_{xz}\hat{O}_{wc}$, i.e. $\|{}_{xz}\hat{O}_{wc}\| = \sqrt{{}_{xz}x_{wc}^2 + {}_{xz}z_{wc}^2}$. The 2 possibilities for the elbow joint (${}_{xz}E_{ej1a}$ and ${}_{xz}E_{ej2a}$) were then mirrored about the X-axis (the x values were the same but y values had opposite sign, see figure 75b). In the figure $\theta_{2a'} = -\theta_{2b'}$. To solve for $\theta_{2a'}$, both its sine and cosine had to be calculated.

From a rearrangement of the cosine rule $\mathbf{cos}(\theta_{2a'})$ could be evaluated, since all the lengths of the triangle formed by O_g, ${}_{xz}E_{ej1a}$ and ${}_xO_{wc}$ were known, i.e. $c\theta_{2a'} = \dfrac{a_2^2 + \|{}_x\hat{O}_{wc}\|^2 - a_3^2}{2.a_2.\|{}_x\hat{O}_{wc}\|}$. Additionally using both Heron's formula and the sine area formula to calculate the area of said triangle, $\mathbf{sin}(\theta_{2a'})$ could be calculated. Heron's formula, used the semi-perimeter given by $p = \dfrac{a_2+a_3+\|{}_{xz}\hat{O}_{wc}\|}{2}$ and gave the triangle area as $\sqrt{p\cdot(p-a_2)\cdot(p-a_3)\cdot(p-\|{}_{xz}\hat{O}_{wc}\|)}$ using only the triangle side lengths. The trigonometric formula for triangle area was given by $0.5\cdot a_2\cdot\|{}_{xz}\hat{O}_{wc}\|\cdot\mathbf{sin}(\theta_{2a'})$. Equating and making $\mathbf{sin}(\theta_{2a'})$ the subject of the formula gave $s\theta_{2a'} = \dfrac{\sqrt{p\cdot(p-a_2)\cdot(p-a_3)\cdot(p-\|{}_{xz}\hat{O}_{wc}\|)}}{0.5\cdot a_2\cdot\|{}_{xz}\hat{O}_{wc}\|}$. $\theta_{2a'}$ Was calculated using the arc functions, noting the common solutions, and that $\theta_{2a} = \theta_{2a'}+\psi$ ($\theta_{2b'} = -\theta_{2a'}$ and $\theta_{2b} = -\theta_{2a'}+\psi$). The coordinates of the projected elbow joints were also evaluated from ${}_{xz}E_{ej1a} = \begin{bmatrix} a_2\cdot c\theta_{2a'} \\ 0 \\ a_2\cdot s\theta_{2a'} \end{bmatrix}$ and ${}_{xz}E_{ej2a} = \begin{bmatrix} a_2\cdot c\theta_{2b'} \\ 0 \\ -a_2\cdot s\theta_{2b'} \end{bmatrix}$. θ_{3a} and θ_{3b} ($\theta_{3b} = 360-\theta_{3a}$) which were the angles between the proximal and distal links in a clockwise sense, had not changed in value with either of the rotated projections.

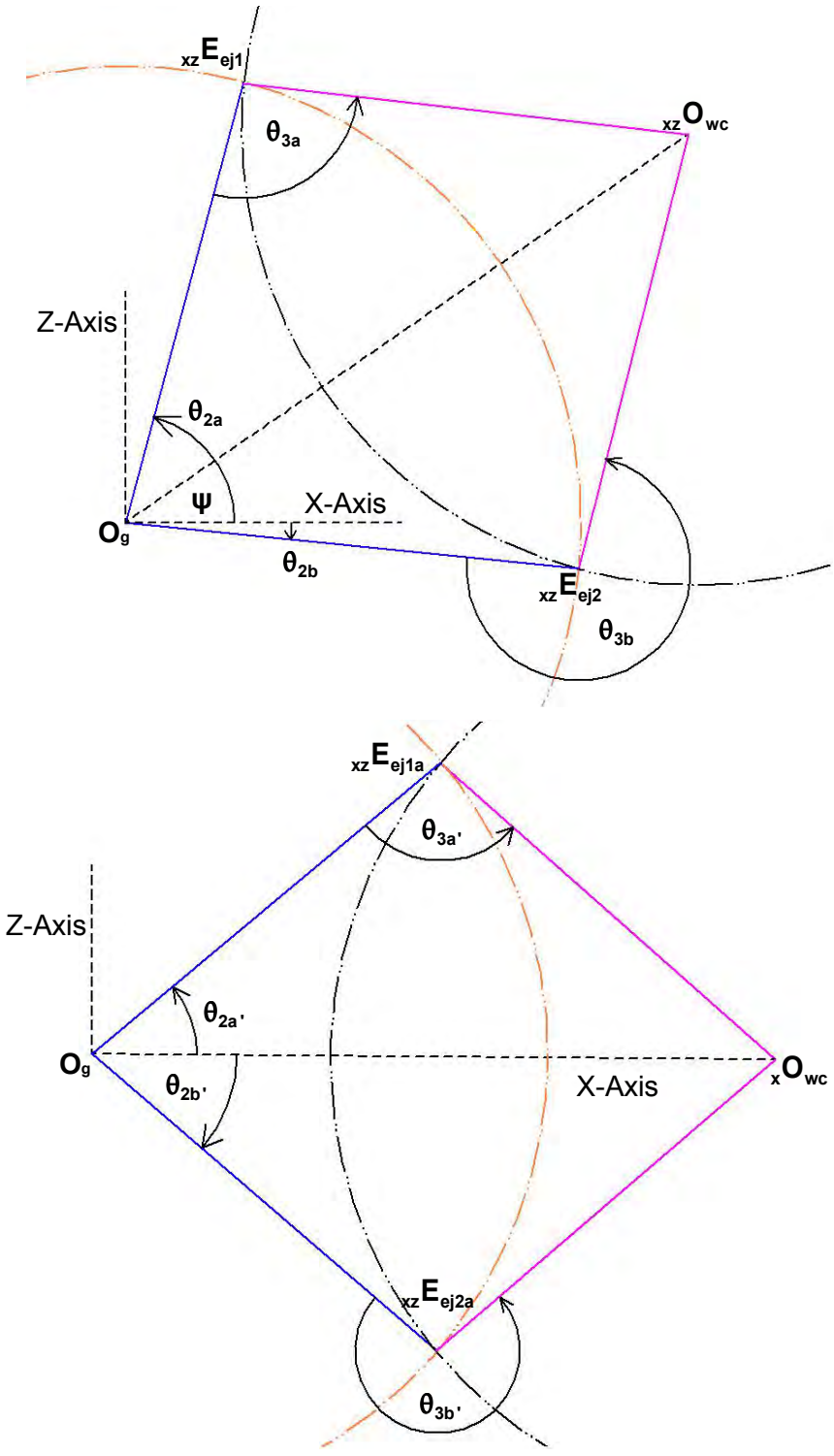


Figure 75: Geometric simplifications to ease calculation of angles 2 and 3

a. Rotated projection of O_{wc} onto XZ-plane, and possible elbow joint locations ${}_{xz}E_{ej1}$, ${}_{xz}E_{ej2}$

b. Rotated projection of ${}_{xz}O_{wc}$ onto X-axis, and possible elbow joint locations ${}_{xz}E_{ej1a}$, ${}_{xz}E_{ej2a}$

θ_{3a} was calculated from its sine and cosine, and in this case the vector cross and dot

products of 2 unit vectors were used, i.e. $s\theta_{3a}=\frac{\sqrt{{}_{xz}\hat{E}_{ej1a}\times\left({}_{x}\hat{O}_{wc}-{}_{xz}\hat{E}_{ej1a}\right)}\cdot{}_{xz}\hat{E}_{ej1a}\times\left({}_{x}\hat{O}_{wc}-{}_{xz}\hat{E}_{ej1a}\right)}{\|{}_{xz}\hat{E}_{ej1a}\|\cdot\|\left({}_{x}\hat{O}_{wc}-{}_{xz}\hat{E}_{ej1a}\right)\|}$

and $c\theta_{3a}=\frac{{}_{xz}\hat{E}_{ej1a}\cdot\left({}_{x}\hat{O}_{wc}-{}_{xz}\hat{E}_{ej1a}\right)}{\|{}_{xz}\hat{E}_{ej1a}\|\cdot\|\left({}_{x}\hat{O}_{wc}-{}_{xz}\hat{E}_{ej1a}\right)\|}$.

The last thing to do for the position kinematics was to obtain the elbow coordinates, in 3D space. This was found by rotating ${}_{xz}E_{ej1a}$ and ${}_{xz}E_{ej2a}$ by positive ψ about the reference frame Y-axis to obtain ${}_{xz}E_{ej1}$ and ${}_{xz}E_{ej2}$ and then rotating the results by positive θ_1 about the reference frame Z-axis to get E_{ej1} and E_{ej2} in 3D space, i.e.

$$E_{ej1}=\begin{bmatrix}c\theta_1 & -s\theta_1 & 0\\ s\theta_1 & c\theta_1 & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}c\psi & 0 & s\psi\\ 0 & 1 & 0\\ -s\psi & 0 & c\psi\end{bmatrix}\cdot\begin{bmatrix}a_2\cdot c\theta_{2a'}\\ 0\\ a_2\cdot s\theta_{2a'}\end{bmatrix}$$ and

$$E_{ej2}=\begin{bmatrix}c\theta_1 & -s\theta_1 & 0\\ s\theta_1 & c\theta_1 & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}c\psi & 0 & s\psi\\ 0 & 1 & 0\\ -s\psi & 0 & c\psi\end{bmatrix}\cdot\begin{bmatrix}a_2\cdot c\theta_{2a'}\\ 0\\ -a_2\cdot s\theta_{2a'}\end{bmatrix}.$$

As it stood there were 2 solutions for the position kinematics, $(\theta_1;\theta_{2a};\theta_{3a})$ and $(\theta_1;\theta_{2b};\theta_{3b})$, for a given 3D location. Two additional solutions were possible, where the first angle was replaced by θ_1+180° for both sets. The proximal and distal links then have to be rotated so that the elbow and wrist points coincide with the original positions, so θ_2 would be replaced by $180^\circ-\theta_2$, and θ_3 would be replaced by $360^\circ-\theta_3$. The first 2 modified angles could be verified by checking the elbow joint coordinates which was obtained by multiplying the first 3 transforms using the modified angles and comparing it with the same multiplication using the original transforms. The third angle could be verified by doing the exact same thing using instead the first 4 transforms and temporarily setting θ_4 to 0, so that ${}^{3}_{4}T=\begin{bmatrix}1 & 0 & 0 & a_3\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$.

$${}^{0}_{3}T={}^{0}_{1}T\cdot{}^{1}_{2}T\cdot{}^{2}_{3}T=\begin{bmatrix}c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3}-c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & -c_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3}-c_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & s_{\theta1} & c_{\theta1}\cdot c_{\theta2}\cdot a_2\\ s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3}-s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & -s_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3}-s_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & -c_{\theta1} & s_{\theta1}\cdot c_{\theta2}\cdot a_2\\ s_{\theta2}\cdot c_{\theta3}+c_{\theta2}\cdot s_{\theta3} & -s_{\theta2}\cdot s_{\theta3}+c_{\theta2}\cdot c_{\theta3} & 0 & s_{\theta2}\cdot a_2\\ 0 & 0 & 0 & 1\end{bmatrix}\quad\ldots(1)$$

$$ {}^{0}_{1}T_b = \begin{bmatrix} c(\theta_1+180) & -s(\theta_1+180) & 0 & 0 \\ s(\theta_1+180) & c(\theta_1+180) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -c\theta_1 & s\theta_1 & 0 & 0 \\ -s\theta_1 & -c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots \quad (2) $$

$$ {}^{1}_{2}T_b = \begin{bmatrix} c(180-\theta_2) & -s(180-\theta_2) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s(180-\theta_2) & c(180-\theta_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & -c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots (3) $$

$$ {}^{2}_{3}T_b = \begin{bmatrix} c(360-\theta_3) & -s(360-\theta_3) & 0 & a_2 \\ s(360-\theta_3) & c(360-\theta_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta_3 & s\theta_3 & 0 & a_2 \\ -s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots \quad (4) $$

$$ {}^{0}_{3}T_b = {}^{0}_{1}T_b \cdot {}^{1}_{2}T_b \cdot {}^{2}_{3}T_b $$

$$ = \begin{bmatrix} c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & c_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} + c_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & -s_{\theta1} & c_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & s_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} + s_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & c_{\theta1} & s_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta2}\cdot c_{\theta3} + c_{\theta2}\cdot s_{\theta3} & s_{\theta2}\cdot s_{\theta3} - c_{\theta2}\cdot c_{\theta3} & 0 & s_{\theta2}\cdot a_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \dots \quad (5) $$

The first 3 elements of the last column of both ${}^{0}_{3}T$ and ${}^{0}_{3}T_b$ are identical, these represent the elbow joint coordinates.

$$ {}^{0}_{4}T = {}^{0}_{1}T \cdot {}^{1}_{2}T \cdot {}^{2}_{3}T \cdot {}^{3}_{4}T $$

$$ = \begin{bmatrix} c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & -c_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & s_{\theta1} & c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3}\cdot a_3 + c_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & -s_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & -c_{\theta1} & s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3}\cdot a_3 + s_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta2}\cdot c_{\theta3} + c_{\theta2}\cdot s_{\theta3} & -s_{\theta2}\cdot s_{\theta3} + c_{\theta2}\cdot c_{\theta3} & 0 & s_{\theta2}\cdot c_{\theta3} + c_{\theta2}\cdot s_{\theta3}\cdot a_3 + s_{\theta2}\cdot a_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

… (6)

$$ {}^{0}_{4}T_b = {}^{0}_{1}T_b \cdot {}^{1}_{2}T_b \cdot {}^{2}_{3}T_b \cdot {}^{3}_{4}T $$

$$ = \begin{bmatrix} c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & c_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} + c_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & -s_{\theta1} & c_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - c_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3}\cdot a_3 + c_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3} & s_{\theta1}\cdot c_{\theta2}\cdot s_{\theta3} + s_{\theta1}\cdot s_{\theta2}\cdot c_{\theta3} & c_{\theta1} & s_{\theta1}\cdot c_{\theta2}\cdot c_{\theta3} - s_{\theta1}\cdot s_{\theta2}\cdot s_{\theta3}\cdot a_3 + s_{\theta1}\cdot c_{\theta2}\cdot a_2 \\ s_{\theta2}\cdot c_{\theta3} + c_{\theta2}\cdot s_{\theta3} & s_{\theta2}\cdot s_{\theta3} - c_{\theta2}\cdot c_{\theta3} & 0 & s_{\theta2}\cdot c_{\theta3} + c_{\theta2}\cdot s_{\theta3}\cdot a_3 + s_{\theta2}\cdot a_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

… (7)

The first 3 elements of the last column of both ${}^0_4T$ and ${}^0_4T_b$ are identical, these represent the wrist origin coordinates. Hence the additional solution sets were $(180^o+\theta_1;180^o-\theta_{2a};360^o-\theta_{3a})$ and $(180^o+\theta_1;180^o-\theta_{2b};360^o-\theta_{3b})$.

The next part was to solve the wrist kinematics, to get the wrist orientation angles. This was accomplished using the first 3 angles calculated (the first 3 transforms could be evaluated), in conjunction with the transform for the forward kinematics, ${}^0_6T={}^0_1T\cdot{}^1_2T\cdot{}^2_3T\cdot{}^3_4T\cdot{}^4_5T\cdot{}^5_6T$, or rather using the 2 possible calculated transforms ${}^0_3T$ and ${}^0_3T_b$, so that ${}^0_6T={}^0_3T\cdot{}^3_4T\cdot{}^4_5T\cdot{}^5_6T$ and ${}^0_6T={}^0_3T_b\cdot{}^3_4T\cdot{}^4_5T\cdot{}^5_6T$. The notation could be simplified by setting ${}^3_4T\cdot{}^4_5T\cdot{}^5_6T$ to ${}^3_6T$. The equations were rearranged to make ${}^3_6T$ the subject in each of the mentioned cases by multiplying ${}^0_6T$ (which was known), by the inverses of ${}^0_3T$ and ${}^0_3T_b$, i.e. ${}^3_6T={}^0_3T^{-1}\cdot{}^0_6T$ and ${}^3_6T={}^0_3T_b^{-1}\cdot{}^0_6T$. The left hand sides of these equations were unknown whereas the right hand sides were known, also the focus could now lie with the rotation matrix portion of the transforms (the upper left 3×3 sub-matrices of ${}^3_6T$, ${}^0_3T^{-1}\cdot{}^0_6T$ and ${}^0_3T_b^{-1}\cdot{}^0_6T$, these are called R, M and N respectively).

The symbolic multiplication of ${}^3_6T$ gave a symbolic representation for R, i.e. $R=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}=\begin{bmatrix} -s_{\theta5}\cdot s_{\theta6} & -s_{\theta5}\cdot c_{\theta6} & c_{\theta5} \\ s_{\theta4}\cdot c_{\theta6}+c_{\theta4}\cdot c_{\theta5}\cdot s_{\theta6} & -s_{\theta4}\cdot s_{\theta6}+c_{\theta4}\cdot c_{\theta5}\cdot c_{\theta6} & c_{\theta4}\cdot s_{\theta5} \\ -c_{\theta4}\cdot c_{\theta6}+s_{\theta4}\cdot c_{\theta5}\cdot s_{\theta6} & c_{\theta4}\cdot s_{\theta6}+s_{\theta4}\cdot c_{\theta5}\cdot c_{\theta6} & s_{\theta4}\cdot s_{\theta5} \end{bmatrix}$ and M, N are also represented symbolically as $M=\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$, $N=\begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix}$. Solutions for θ_4, θ_5 and θ_6 were found by equating the elements of matrix R with the corresponding known elements of matrix M (also for N). Starting with R = M, it could be seen that $r_{13}=c_{\theta5}=m_{13}$. There were 2 possible values for $s_{\theta5}$ and hence for θ_5 (θ_{5a} and θ_{5b}), i.e. $s_{\theta5a}=\sqrt{1-c_{\theta5}^2}=\sqrt{1-m_{13}^2}$ and $s_{\theta5b}=-\sqrt{1-c_{\theta5}^2}=-\sqrt{1-m_{13}^2}$. Corresponding values for $s_{\theta4}$ and $c_{\theta4}$ were found using the fact that $r_{23}=c_{\theta4}\cdot s_{\theta5}=m_{23}$ and $r_{33}=s_{\theta4}\cdot s_{\theta5}=m_{33}$. This implied that $c_{\theta4a}=\frac{m_{23}}{s_{\theta5a}}$ and $s_{\theta4a}=\frac{m_{33}}{s_{\theta5a}}$, and a unique θ_{4a} was found. Similarly values for $s_{\theta6}$ and $c_{\theta6}$ were found using the fact that $r_{11}=-s_{\theta5}\cdot s_{\theta6}=m_{11}$ and $r_{12}=-s_{\theta5}\cdot c_{\theta6}=m_{12}$, i.e. $s_{\theta6a}=\frac{m_{11}}{-s_{\theta5a}}$

and $c_{\theta 6a} = \frac{m_{12}}{-s_{\theta 5a}}$ which resulted in a unique value for θ_{6a}. Using $s_{\theta 5b}$ corresponding solutions for θ_{4b} and θ_{6b} could be found. There were thus 2 solution sets for the orientation angles of matrix M, i.e. $(\theta_{4a}; \theta_{5a}; \theta_{6a})$ and $(\theta_{4b}; \theta_{5b}; \theta_{6b})$. Following the same discussion 2 solution sets could be calculated using matrix R and N with R = N, i.e. $(\theta_{4c}; \theta_{5c}; \theta_{6c})$ and $(\theta_{4d}; \theta_{5d}; \theta_{6d})$.

Combining the solution sets there were 8 possible solutions for the angles θ_1 to θ_6 (or 8 pose configurations). These were:

1. $(\theta_1; \theta_{2a}; \theta_{3a}; \theta_{4a}; \theta_{5a}; \theta_{6a})$
2. $(\theta_1; \theta_{2a}; \theta_{3a}; \theta_{4b}; \theta_{5b}; \theta_{6b})$
3. $(\theta_1; \theta_{2b}; \theta_{3b}; \theta_{4a}; \theta_{5a}; \theta_{6a})$
4. $(\theta_1; \theta_{2b}; \theta_{3b}; \theta_{4b}; \theta_{5b}; \theta_{6b})$
5. $(180^{o}+\theta_1; 180^{o}-\theta_{2a}; 360^{o}-\theta_{3a}; \theta_{4c}; \theta_{5c}; \theta_{6c})$
6. $(180^{o}+\theta_1; 180^{o}-\theta_{2a}; 360^{o}-\theta_{3a}; \theta_{4d}; \theta_{5d}; \theta_{6d})$
7. $(180^{o}+\theta_1; 180^{o}-\theta_{2b}; 360^{o}-\theta_{3b}; \theta_{4c}; \theta_{5c}; \theta_{6c})$
8. $(180^{o}+\theta_1; 180^{o}-\theta_{2b}; 360^{o}-\theta_{3b}; \theta_{4d}; \theta_{5d}; \theta_{6d})$

3.5.3. Machine level dynamics

There are 2 standard methods used to calculate the dynamics of robots i.e., the iterative Newton-Euler and energy-based Lagrangian techniques. According to *Craig (2005)* [89], the Lagrangian technique was more computationally intensive, and for that reason the iterative Newton-Euler algorithm was used here. Additionally it was easier to code, test and implement.

The technique worked by first finding the linear and angular velocities of all points of interest (these were link coordinate frame origins and centres of mass, COMs), which started at the fixed base and moved out to the last link coordinate frame. The linear and angular accelerations were also found, and from those the link's net force and torque were calculated. That was referred to as calculation from an in-to-out direction. Once the last net

force and torque was found on the last coordinate frame, the algorithm reversed and moved in an out-to-in direction, taking into account at that last coordinate frame whatever force or torque one wished the robot to exert on the environment. The forces and torques were then calculated on the ends of the link, from the last coordinate frame on the robot first, down to the base coordinate frame. The joint torques were evaluated using a dot product of the joint vector with the total torque applied at the joint. For convenience the equations constituting this algorithm are reproduced here from *Craig (2005)* [89].

Outward iterations: For i = 0 to 5

$$^{i+1}\omega_{i+1} = {}^{i+1}_{i}R\,{}^{i}\omega_{i} + \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} \quad \ldots (8)$$

$$^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R\,{}^{i}\dot{\omega}_{i} + {}^{i+1}_{i}R\,{}^{i}\omega_{i} \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} \quad \ldots (9)$$

$$^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R\left({}^{i}\dot{\omega}_{i} \times {}^{i}P_{i+1} + {}^{i}\omega_{i} \times \left({}^{i}\omega_{i} \times {}^{i}P_{i+1}\right) + {}^{i}\dot{v}_{i}\right) \quad \ldots (10)$$

$$^{i+1}\dot{v}_{C(i+1)} = {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}P_{C(i+1)} + {}^{i+1}\omega_{i+1} \times \left({}^{i+1}\omega_{i+1} \times {}^{i+1}P_{C(i+1)}\right) + {}^{i+1}\dot{v}_{i+1} \quad \ldots (11)$$

$$^{i+1}F_{i+1} = m_{i+1}\,{}^{i+1}\dot{v}_{C(i+1)} \quad \ldots (12)$$

$$^{i+1}N_{(i+1)} = {}^{C(i+1)}I_{i+1}\,{}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C(i+1)}I_{i+1}\,{}^{i+1}\omega_{i+1} \quad \ldots (13)$$

Inward iterations: For i = 6 to 1

$$^{i}f_{i} = {}^{i}_{i+1}R\,{}^{i+1}f_{i+1} + {}^{i}F_{i} \quad \ldots (14)$$

$$^{i}n_{i} = {}^{i}N_{i} + {}^{i}_{i+1}R\,{}^{i+1}n_{i+1} + {}^{i}P_{C(i)} \times {}^{i}F_{i} + {}^{i}P_{i+1} \times {}^{i}_{i+1}R\,{}^{i+1}f_{i+1} \quad \ldots (15)$$

$$^{i}\tau_{i} = {}^{i}n_{i}\,{}^{i}\hat{Z}_{i} \quad \ldots (16)$$

${}^{i}\omega_{i}$ Represented the angular velocity of the i^{th} frame or link in the i^{th} frame. The leading superscript was the frame in which the vector was expressed and the following subscript represented the frame, link or point in question. Similarly for the velocity vector ${}^{i}v_{i}$ and the time derivatives ${}^{i}\dot{\omega}_{i}$ and ${}^{i}\dot{v}_{i}$, which represented the angular and linear acceleration vectors respectively. $\dot{\theta}_{i}$ And $\ddot{\theta}_{i}$ were the magnitudes of the angular velocity and acceleration of rotational joint i. ${}^{i}P_{i+1}$ Was the vector from the origin of frame i to the origin of frame $i+1$ expressed in frame i. ${}^{i}P_{C(i)}$ Was the vector locating the COM of the i^{th} link from the origin of frame i expressed in the i^{th} frame. ${}^{i}F_{i}$ Was the net force acting on link i, which gave it a

linear acceleration of ${}^{i}\dot{v}_i$ at its COM. ${}^{i}N_i$ Was the net moment acting on link i, which had inertia tensor ${}^{C(i)}I_i$. ${}^{i}\tau_i$ Was the required joint torque for the i^{th} joint expressed in the i^{th} frame.

None of the equations in the algorithm had any reference to gravity terms, but this was taken into account by setting the linear acceleration of the base frame (where $i = 0$) to $-\hat{g}$, a vector that had the same magnitude as gravity but acted in the opposite direction. Effectively it cancelled gravity. The initial conditions were that ${}^{0}\omega_0 = {}^{0}\dot{\omega}_0 = \hat{0}$, ${}^{0}v_0 = \hat{0}$ and ${}^{0}\dot{v}_0 = -\hat{g}$, in the outward iteration phase. For the inward iteration phase ${}^{7}f_7$ represented the external force applied to the robot tool or the force that the robot was expected to exert on its environment. Likewise for the external moment ${}^{7}n_7$.

3.5.3.1. Iterative forward dynamics

The forward dynamics was simple to implement with the algorithm above. Here the task was to calculate the force and moments of the links, as well as the output force and moment applied by the end-effector when given the joint angles, joint rates and joint accelerations. The end-effector position and orientation could be found using the forward kinematics solution. With the given joint angles all rotation matrices ${}^{i+1}_{i}R$ could be evaluated and hence all ${}^{i+1}\omega_{i+1}$ and ${}^{i+1}\dot{\omega}_{i+1}$ could be found. All model parameters were either known or could be calculated including ${}^{i}P_{i+1}$, ${}^{i+1}P_{C(i+1)}$ and ${}^{C(i+1)}I_{i+1}$, which made evaluation of the remaining vectors ${}^{i+1}\dot{v}_{i+1}$, ${}^{i+1}\dot{v}_{C(i+1)}$, ${}^{i+1}F_{i+1}$, ${}^{i+1}N_{i+1}$, ${}^{i}f_i$, ${}^{i}n_i$ and ${}^{i}\tau_i$ possible.

3.5.3.2. Iterative inverse dynamics

The inverse dynamics was precisely the opposite. Given a time stamped path which had position and orientation information as well as the initial joint velocities $\left(\dot{\Theta}_0\right)$ and accelerations $\left(\ddot{\Theta}_0\right)$ at the robot's initial configuration (the applied force vector at the end-effector could also be given), the objective was to calculate all joint torques for each robot configuration/pose in that path. The first step was to use the inverse kinematics solution to solve for the joint angles at each robot pose $\left(\hat{\theta}_i = [\theta_1, \theta_2, \ldots, \theta_6]_i^T\right)$. The change in each joint

angle between successive poses could then be evaluated, call this $\Delta\hat{\theta}_i = \hat{\theta}_{i+1} - \hat{\theta}_i = [\Delta\theta_1, \Delta\theta_2, \ldots, \Delta\theta_6]_i^T$. The time step $\left(\Delta t_i = t_{i+1} - t_i\right)$ was calculated from successive points in the time stamped path. Dividing $\Delta\hat{\theta}_i$ by Δt_i gave the average angular velocity for each joint between successive robot configurations. If a linear relation was used to get from one set of joint velocities to the next (a straight line segment joining consecutive angular velocities, or constant angular acceleration between those 2 points), the set of joint angle velocities at path point i+1 $\left(\dot{\Theta}_{i+1}\right)$ would be given by $\dot{\Theta}_{i+1} = \frac{2\cdot\Delta\hat{\theta}_i}{\Delta t_i} - \dot{\Theta}_i$. This came from the average velocity formula $v_{ave} = \frac{v_i + v_{i+1}}{2}$ or $v_{i+1} = 2\cdot v_{ave} - v_i$ (which was valid only for constant acceleration). The joint angle accelerations at path point i+1 $\left(\ddot{\Theta}_{i+1}\right)$ was then given by $\ddot{\Theta}_{i+1} = \frac{2}{\Delta t_i}\cdot\left(\dot{\Theta}_{i+1} - \dot{\Theta}_i\right) - \ddot{\Theta}_i = \frac{4}{\Delta t_i}\cdot\left(\frac{\Delta\hat{\theta}_i}{\Delta t_i} - \dot{\Theta}_i\right) - \ddot{\Theta}_i$ (from the average acceleration formula). When the joint angles, velocities and accelerations were known, the forces and moments could be calculated using the iterative dynamics algorithm. The joint angle accelerations for the last point in the time stamped path could not be calculated as it depended on the next path point which did not exist. If however the robot came to rest at the end of the path it was set to zero. Hence the forces and moments could be calculated for n configurations of a time stamped path containing n + 1 points.

3.5.3.3. Closed form dynamics

The closed form dynamics of the robot could be formulated using the iterative algorithm, with symbolic terms and proceeding first outward for velocities and accelerations and then inward for forces and moments/torques. However for the 6 DOF manipulator the symbolic result would be very long and complex, and was thus not presented. Additionally trying to code the closed form solution, although computationally efficient, would be tedious and error prone. If necessary these equations could be derived using this iterative algorithm and a symbolic manipulation toolbox in a mathematical package such as Matlab.

3.5.3.4. Simplified dynamic models

3.5.3.4.1. Common base, wrist and link dynamic modelling

To implement a dynamic simulation the models used had to be simplified, there were too many components to account for them all. Additionally the major elements would affect the dynamics much more than the minor elements. This was similar to the machine level kinematics analysis where it wasn't necessary to know the position of every element to calculate the required joint angles. Additional geometric simplifications would aid calculation and generalize analysis. Also it must be stressed that for the dynamics algorithm to work the inertia tensor must be calculated at the links COM.

For all robot dynamic models the wrist was reduced to a solid round bar having a concentrated total mass of m_w at a distance $\frac{l_3}{2}$ from the wrist centre (spherical centre), with radius r_w. l_3 Was a parameter in the kinematics model, i.e. the distance from tool tip to wrist centre. The reference axes for the moments of inertia were taken about the principle axes, and the principle planes divided the part into mirrored halves. As such the inertia tensor reduced to a 3×3 diagonal matrix, containing only the moments of inertia about the principle axes. The products of inertia were all 0. The moments of inertia of a round bar (or cylinder) about its principle axes are $I_{xx}, I_{yy} = \frac{1}{4}mr^2 + \frac{1}{12}ml^2$ and $I_{zz} = \frac{1}{2}mr^2$, directly from *Meriam and Kraige (2007)* [88], where the z-axis here coincides with the round bar axis.

The inertia tensor of the round bar about its principle axes, at its COM was then given by $I_w = \frac{1}{12}\begin{bmatrix} 3m_w r_w^2 + m_w l_3^2 & 0 & 0 \\ 0 & 3m_w r_w^2 + m_w l_3^2 & 0 \\ 0 & 0 & 6m_w r_w^2 \end{bmatrix}$ (all products of inertia are 0 about the principle axes). This tensor was written with respect to the robots sixth coordinate frame for joint axis 6, where the z-axis of that frame was coincident with the round bar axis.

For all models the BCGM was modelled as concentric discs (flattened cylinders) that were coplanar. In the mechanical designs they had collinear rotational axes but were on parallel planes. For the dynamic modelling the distance between planes could be ignored as the

concentric sections did not move out of the planes, see figure 76. Sections (discs) 2 to 5 had holes that were filled by sections (discs) 1 to 4 respectively. Sections 1 and 6 were solid, the fact that disc 6 occupied the same space as discs 1 to 5 could be ignored since this simplification would be used for all dynamic models. All discs had the same thickness l_4, masses $m_{bg(j)}$ and outer radii $r_{bg(j)}$ for $j=1$ to 6. The inertia tensor for discs 1 and 6 was given by $I_{bg(j)} = \frac{m_{bg(j)}}{12}\begin{bmatrix} 3r_{bg(j)}^2+l_4^2 & 0 & 0 \\ 0 & 3r_{bg(j)}^2+l_4^2 & 0 \\ 0 & 0 & 6r_{bg(j)}^2 \end{bmatrix}$, $j=1$ or 6. The inertia tensor for discs 2 to 5 was defined by $I_{bg(j)} = \frac{m_{bg(j)}}{12}\begin{bmatrix} 3\left(r_{bg(j)}^2+r_{bg(j-1)}^2\right)+l_4^2 & 0 & 0 \\ 0 & 3\left(r_{bg(j)}^2+r_{bg(j-1)}^2\right)+l_4^2 & 0 \\ 0 & 0 & 6\left(r_{bg(j)}^2+r_{bg(j-1)}^2\right) \end{bmatrix}$, $j=2$ to 5. The gaps between discs were ignored. Discs 1 to 5 had an angular acceleration defined by the gear ratios between the disc and the motor driving it, call this $\dot{\omega}_{bg(j)}$. The net torque applied to disc j was therefore $I_{bg(j)}(3,3)\cdot\dot{\omega}_{bg(j)}$. This had to be added to the torque calculated for joint j (and the net torques for the transfer rods) to get the actual motor torque required from the stationary actuator.

One of the design goals in each of the design embodiments was to have a symmetric machine. That meant the COMs of all major links would lie on a plane that passed through the machine's $\hat{Z}_1$ axis, and was perpendicular to the $\hat{Z}_2$ axis. The coordinates of the COMs of the main proximal and distal links were then at the mid-points of the links. It was for this reason that the dynamic models were made symmetric even the actual CAD models were not. This objective could be obtained with additional design time. For the H links (main proximal and distal links) solid rectangular bars were used to model the constituent parts of the links. Holes and rounded sections were ignored.

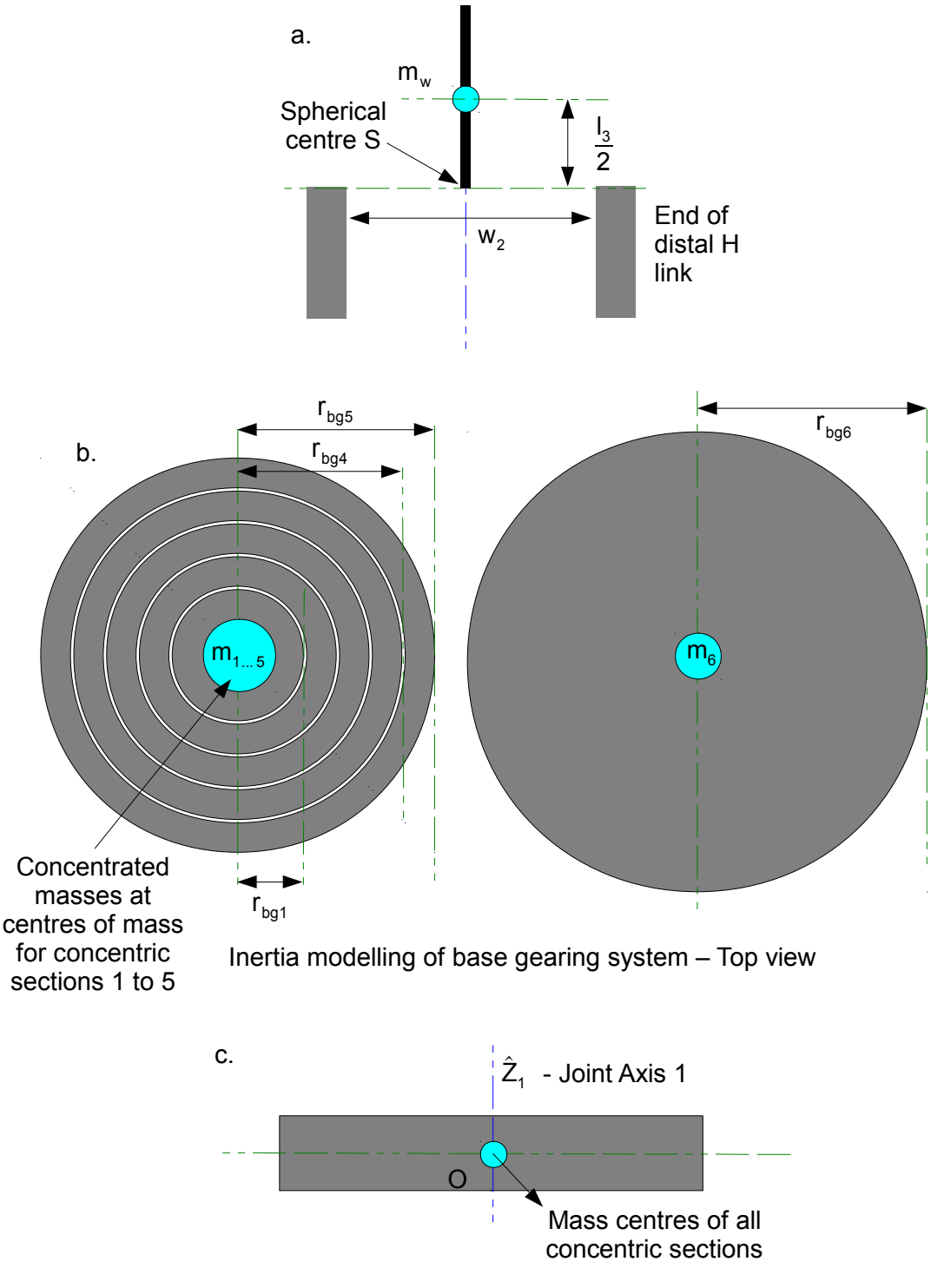


Figure 76: Common dynamic modelling for BCGM and wrist

The reference axes for the moments of inertia were taken about the principle axes, and the principle planes divided the constituent parts into mirrored halves. The principle moments of inertia for the rectangular bar were $I_{xx} = \frac{1}{12} m(a^2 + b^2)$, $I_{yy} = \frac{1}{12} m(a^2 + l^2)$ and

$I_{zz} = \frac{1}{12} m(b^2 + l^2)$, where the major length portion $(a, b < l)$ was in the x direction, from *Meriam and Kraige (2007)* [88]. Using the parallel axis theorem those moments were then translated to the link's COM.

3.5.3.4.2. Design embodiment 1: Proximal and distal link modelling

The proximal H link was modelled using 2 identical cross structures (combined vertical and horizontal rectangular bars) shown in figure 77a, and 2 horizontal bars indicated in figure 77b, c. The vertical bar passing through C in figure 77a had measurements of a_1, b_2 and l_1 for the width, height and length respectively. The inertia tensor about its principle axes was $I_{m(1)} = \frac{m_1}{12}\begin{bmatrix} (a_1^2 + b_2^2) & 0 & 0 \\ 0 & (a_1^2 + l_1^2) & 0 \\ 0 & 0 & (b_2^2 + l_1^2) \end{bmatrix}$. The horizontal bar passing through ACB in figure 77a had measurements of a_1, b_1 and c for the width, height and length respectively. Its inertia tensor was $I_{m(2)} = \frac{m_2}{12}\begin{bmatrix} (a_1^2 + b_1^2) & 0 & 0 \\ 0 & (a_1^2 + c_1^2) & 0 \\ 0 & 0 & (b_1^2 + c_1^2) \end{bmatrix}$. Since those tensors were written about coincident axes, they could be added however the common volume (with mass m_c) had to be subtracted (inherently added twice). The inertia tensor of the cross structure was therefore $I_{m(a)} = I_{m(1)} + I_{m(2)} - \frac{m_c}{12}\begin{bmatrix} (a_1^2 + b_2^2) & 0 & 0 \\ 0 & (a_1^2 + c_1^2) & 0 \\ 0 & 0 & (b_2^2 + c_1^2) \end{bmatrix}$. Using the parallel axis theorem this tensor was translated to the proximal H link's COM $\hat{P}_{PH} = 0.5 \cdot l_1 \cdot {}^2\hat{X}_2$ giving $I_{m(a)\text{ at }P} = I_{m(a)} + (m_1 + m_2 - m_c)\begin{bmatrix} \left(\frac{w_1}{2}\right)^2 & 0 & 0 \\ 0 & \left(\frac{w_1}{2}\right)^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ (as it only needed a translation parallel to the $\hat{Z}_2$ axis). Since there were 2 cross structures the collective contribution at P was $2I_{m(a)\text{ at }P}$ (since they translate the same distance to the H link's COM). The horizontal bar passing through E had an inertia tensor

$I_{m(3)} = \frac{1}{12}\begin{bmatrix} m_3(a_2^2+(w_1-a_1)^2) & 0 & 0 \\ 0 & m_3((w_1-a_1)^2+c_1^2) & 0 \\ 0 & 0 & m_3(a_2^2+c_1^2) \end{bmatrix}$. When it was translated to the proximal link's COM using the parallel axis theorem it gave $I_{m(3)\text{ at P}} = I_{m(3)} + m_3\begin{bmatrix} \left(\frac{b_1}{2}-\frac{a_2}{2}\right)^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \left(\frac{b_1}{2}-\frac{a_2}{2}\right)^2 \end{bmatrix}$ (as it only needed a translation parallel to the $\hat{Y}_2$ axis). Since there were 2 horizontal bars the collective contribution at P was $2I_{m(3)\text{ at P}}$. The inertia tensor and total mass at P thus far was $I_{PH} = 2I_{m(a)\text{ at P}} + 2I_{m(3)\text{ at P}}$ and $m_{PH} = 2(m_1 + m_2 + m_3 - m_c)$ respectively. The torque transfer links still had to be accounted for, which were modelled as rectangular bars, with holes and rounded sections ignored. Each proximal torque transfer bar had measurements of p, q and l_1 for the width, breadth and length respectively. The inertia tensor about its principle axes was given by $I_{Ptt(i)} = \frac{1}{12}\begin{bmatrix} m_4(p^2+q^2) & 0 & 0 \\ 0 & m_4(p^2+l_1^2) & 0 \\ 0 & 0 & m_4(q^2+l_1^2) \end{bmatrix}$.

When the wrist and/or elbow joint angles were changed, the corresponding proximal torque transfer links change position and orientation with respect to the proximal H links. These changes had to reflect in both the COMs and the inertia tensors for the complete proximal link. The inverse kinematics resolved the position and orientation of those torque transfer links with respect to the H link. Their x and y principle axes were no longer aligned to those of the H link, however the z axis direction was unchanged. The orientation information was therefore an angle about the z axis, given by $\theta_{tt(i)}$. A similarity transform was then used to map the principle inertia tensor onto a set of parallel axes (realigning it to the proximal H link) using those orientation angles (a rotation of $-\theta_{tt(i)}$ about the link's z axis i.e. $R = \begin{bmatrix} c\theta_{tt(i)} & s\theta_{tt(i)} & 0 \\ -s\theta_{tt(i)} & c\theta_{tt(i)} & 0 \\ 0 & 0 & 1 \end{bmatrix}$). The similarity transform was given by $I_{Ptt(i)aligned} = R \cdot I_{Ptt(i)} \cdot R^T$. The tensor could then be translated to the complete proximal link's COM $\hat{P}_{PC}$ for addition,

however that new COM location had to be calculated. If the coordinates for proximal torque transfer link i was given by $[x_i\ y_i\ z_i]^T$ relative to $\hat{P}_{PH}$, then $\hat{P}_{PC} = \hat{P}_{PH} + \Delta\hat{P}_{PL} = 0.5 \cdot l_1 \cdot {}^2\hat{X}_2 + \frac{m_4}{m_{PT}}\left[\sum_{i=1}^{4} x_i \quad \sum_{i=1}^{4} y_i \quad \sum_{i=1}^{4} z_i\right]^T$, where $m_{PT} = 4m_4 + m_{PH}$ and was the mass of the complete proximal link (4 torque transfer links and the H link). The general formula implementing the parallel axis theorem was then used to translate $I_{Ptt(i)aligned}$ to $\hat{P}_{PC}$ by vector $\Delta\hat{P}_{Ptt(i)} = \Delta\hat{P}_{PL} - [x_i\ y_i\ z_i]^T$. This was given by $I_{Ptt(i)at\ COM} = I_{Ptt(i)aligned} + m_{P(TOTAL)} \cdot \left[\left(\Delta\hat{P}^T_{Ptt(i)} \cdot \Delta\hat{P}_{Ptt(i)}\right) \cdot I_{3x3} - \Delta\hat{P}_{Ptt(i)} \cdot \Delta\hat{P}^T_{Ptt(i)}\right]$, in which $\Delta\hat{P}^T_{Ptt(i)} \cdot \Delta\hat{P}_{Ptt(i)}$ was the vector dot product, $I_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $\Delta\hat{P}_{Ptt(i)} \cdot \Delta\hat{P}^T_{Ptt(i)}$ was a 3x3 matrix. The H link's tensor also had to be translated to $\hat{P}_{PC}$ by the vector $\Delta\hat{P}_{PL}$ and so $I_{PH\ at\ COM} = I_{PH} + m_{PH} \cdot \left[\left(\Delta\hat{P}^T_{PL} \cdot \Delta\hat{P}_{PL}\right) \cdot I_{3x3} - \Delta\hat{P}_{PL} \cdot \Delta\hat{P}^T_{PL}\right]$. The complete proximal link's tensor was then $I_{PC} = I_{PH\ at\ COM} + \sum_{i=1}^{4} I_{Ptt(i)\ at\ COM}$.

Following these steps for the complete proximal link in conjunction with the illustration on figure 78, the inertia tensor for the complete distal link was found. This will not be shown here.

Table 3: Dynamics parameters for design 1

i	${}^i\hat{P}_{i+1}$	${}^{i+1}\hat{P}_{C(i+1)}$	m_{i+1}	${}^{C(i+1)}I_{i+1}$
0	$0\,{}^0\hat{X}_0$	$0\,{}^1\hat{X}_1$	$m_{bg(6)}$	$I_{bg(6)}$
1	$0\,{}^1\hat{X}_1$	$0.5 l_1\,{}^2\hat{X}_2 + \Delta\,{}^2\hat{P}_{PL}$	$2(m_1 + m_2 + m_3 - m_c) + 4m_4$	I_{PC}
2	$l_1\,{}^2\hat{X}_2$	$0.5 l_2\,{}^3\hat{X}_3 + \Delta\,{}^3\hat{P}_{DL}$	$2(m_5 + m_6 + m_7 - m_c) + 3m_8$	I_{DC}
3	$l_2\,{}^3\hat{X}_3$	$0\,{}^4\hat{X}_4$	0	$0_{3\times3}$
4	$0\,{}^4\hat{X}_4$	$0\,{}^5\hat{X}_5$	0	$0_{3\times3}$
5	$l_3\,{}^5\hat{X}_5$	$-0.5 l_3\,{}^6\hat{Z}_6$	m_w	I_w

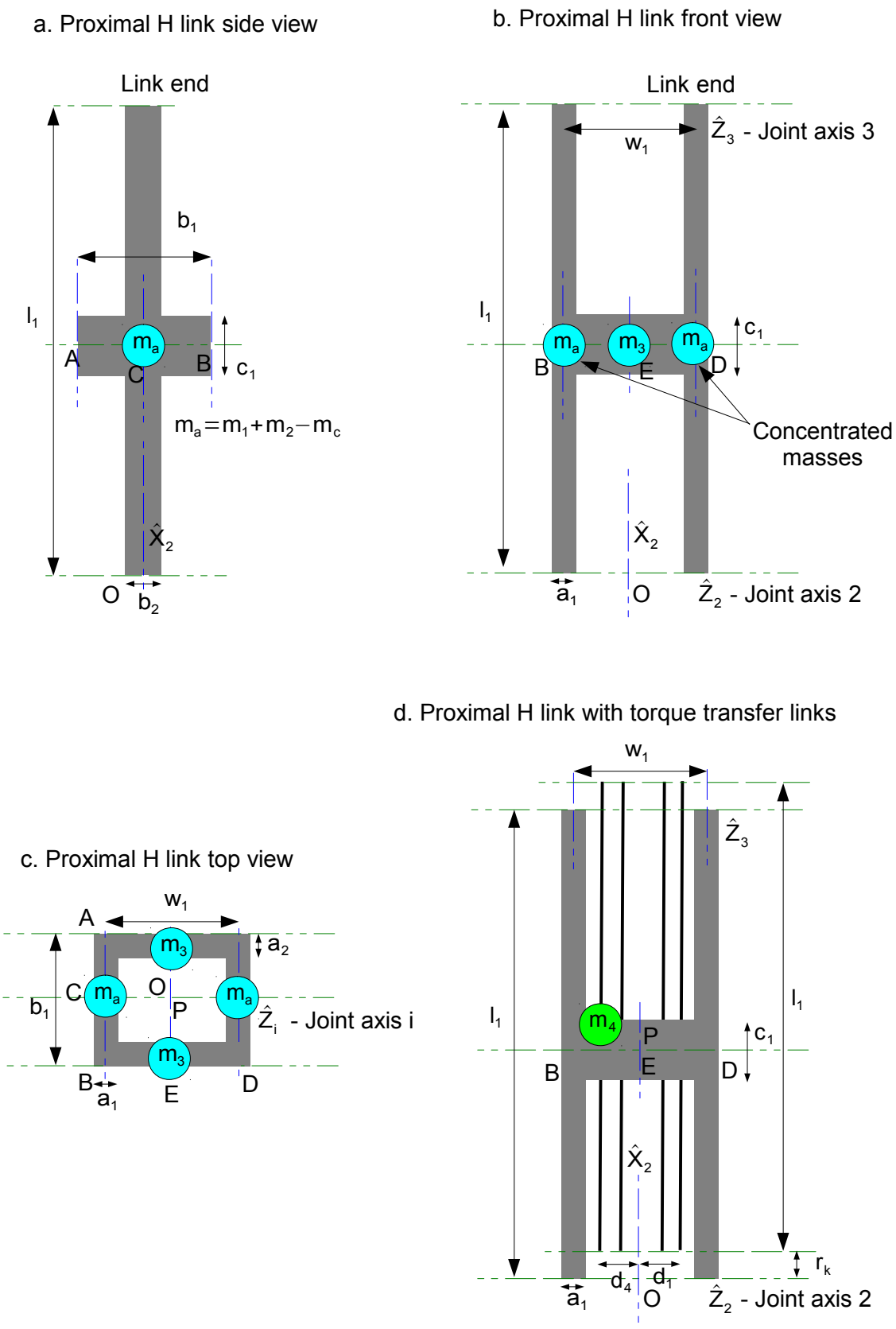


Figure 77: Inertia modelling for design 1's proximal link

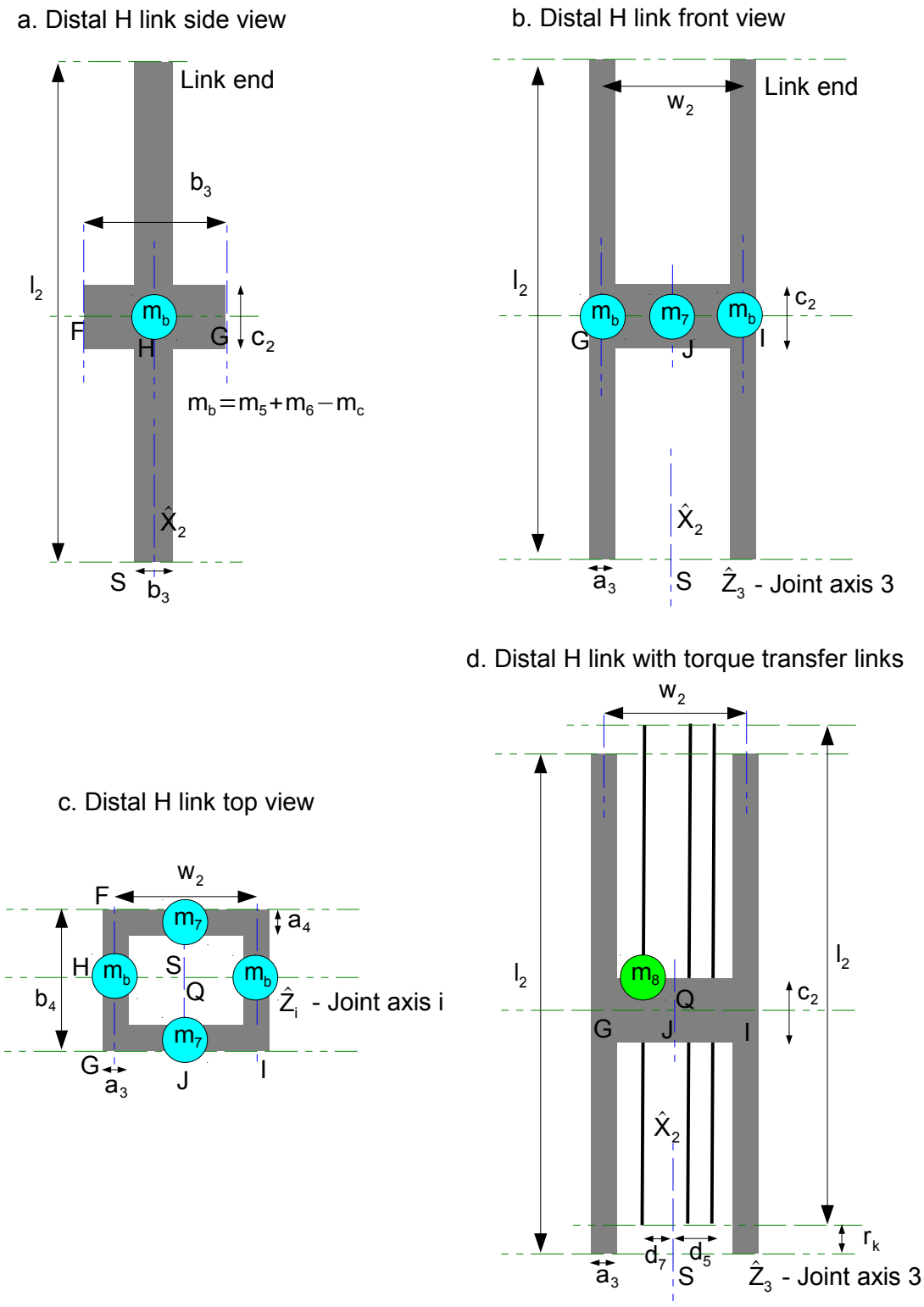


Figure 78: Inertia modelling for design 1's distal link

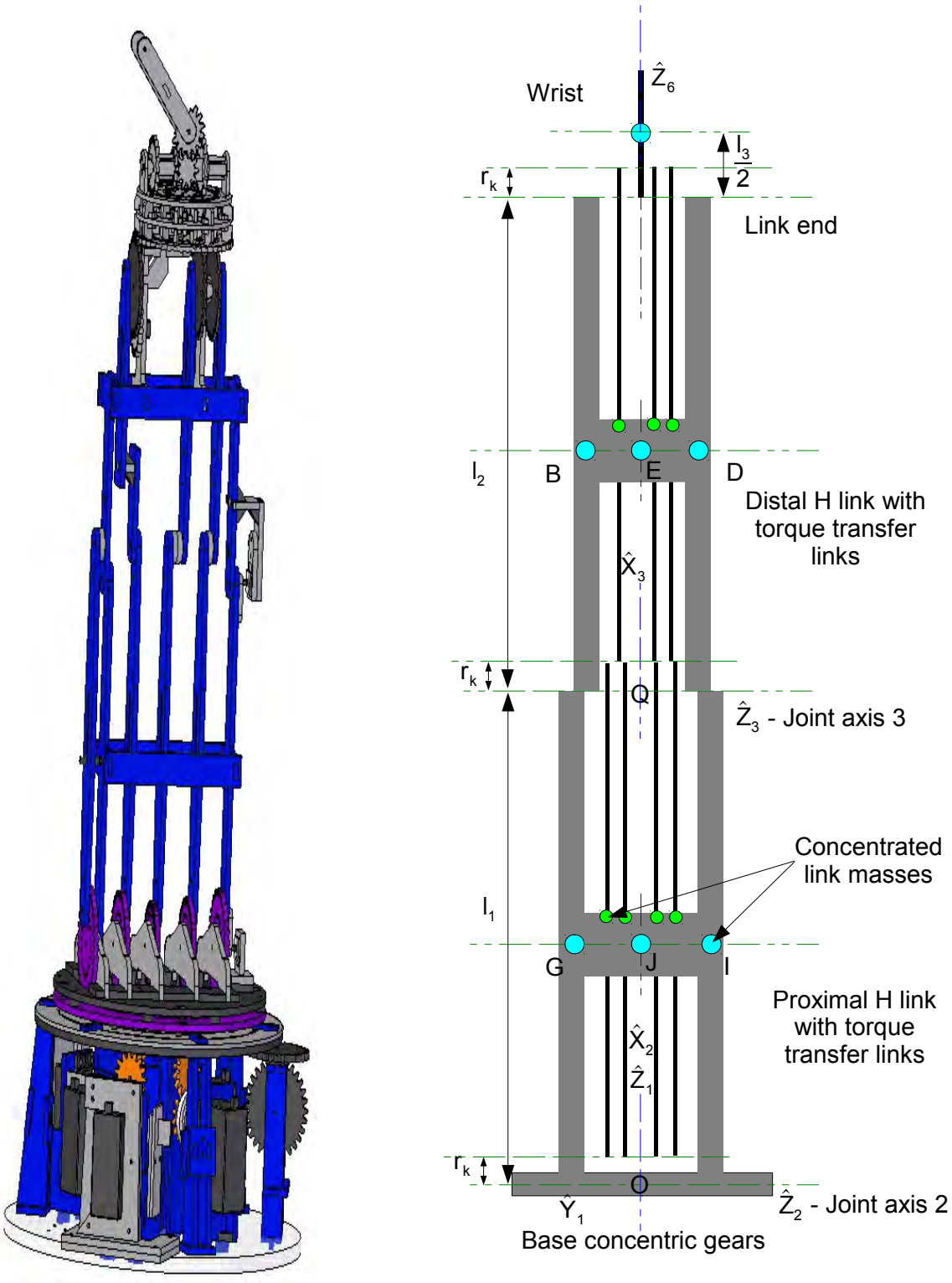


Figure 79: Comparison of CAD model to simplified dynamic model for design 1

3.5.3.4.3. Design embodiment 2: Proximal and distal link modelling

The second design was modelled as a symmetric machine for the reason mentioned previously. In addition since the torque transfer links did not change position with respect to their supporting links, the COMs of the complete proximal ($\hat{P}_{PL}$) and distal ($\hat{P}_{DL}$) links were unchanged with regard to said link's coordinate frame. Those coordinates were at the mid-points of the links even when taking into account the masses of torque transfer links, due to the symmetry. The inertia tensor of the H links (proximal and distal) in this design were derived in the same way as the previous one using rectangular bar moments of inertia for each constituting part of the link (holes and rounded sections were ignored). As shown in figure 80 the inertia tensor of the vertical bar passing through A about its principle axes was given by $I_A = \frac{1}{12}\begin{bmatrix} m_1(a_1^2+b_1^2) & 0 & 0 \\ 0 & m_1(a_1^2+l_1^2) & 0 \\ 0 & 0 & m_1(b_1^2+l_1^2) \end{bmatrix}$, b_1 was the breadth (its direction was into the page in figure 80). The vertical bar passing through B had the same inertia tensor about its principle axes. The tensor of the horizontal bar passing through A, C and B with length (w_1-a_1), was $I_{ACB} = \frac{1}{12}\begin{bmatrix} m_2(b_1^2+(w_1-a_1)^2) & 0 & 0 \\ 0 & m_2(c^2+(w_1-a_1)^2) & 0 \\ 0 & 0 & m_2(b_1^2+c^2) \end{bmatrix}$. The parallel axis theorem was used to transfer all principle moments to a parallel coordinate system with origin at $\hat{P}_{PL}$, $I_{A\text{ at COM}} = \frac{1}{12}\begin{bmatrix} m_1(a_1^2+b_1^2)+12m_1\left(\frac{w_1}{2}\right)^2 & 0 & 0 \\ 0 & m_1(a_1^2+l_1^2)+12m_1\left(\frac{w_1}{2}\right)^2 & 0 \\ 0 & 0 & m_1(b_1^2+l_1^2) \end{bmatrix} =$

$\frac{1}{12}\begin{bmatrix} m_1(a_1^2+b_1^2+3w_1^2) & 0 & 0 \\ 0 & m_1(a_1^2+l_1^2+3w_1^2) & 0 \\ 0 & 0 & m_1(b_1^2+l_1^2) \end{bmatrix}$. I_{ACB} was unchanged as its COM was coincident with the complete proximal link's COM, so $I_{ACB\text{ at COM}} = I_{ACB}$. Hence the inertia tensor of the H link about its COM at C was $I_{PH} = 2I_{A\text{ at COM}} + I_{ACB\text{ at COM}}$. The distal H link's

inertia tensor had the same form with w_2, l_2 and m_4 replacing w_1, l_1 and m_2.

The torque transfer links' inertia tensors were also accounted for, and a cylinder (with radius r_1) was used to model each of those links ignoring the gear heads at each end. The moments of inertia about the link's principle axes were $I_{xx} = \frac{1}{2}mr^2$ and $I_{yy}, I_{zz} = \frac{1}{4}mr^2 + \frac{1}{12}ml^2$. The inertia tensor was then $I_{Ptt(i)}$ = $\frac{1}{12}\begin{bmatrix} 6m_3r_1^2 & 0 & 0 \\ 0 & 3m_3r_1^2+m_3l_1^2 & 0 \\ 0 & 0 & 3m_3r_1^2+m_3l_1^2 \end{bmatrix}$. All tensors were then translated to $\hat{P}_{PL}$ which gave $I_{Ptt(i)\text{ at COM}}$ = $\frac{1}{12}\begin{bmatrix} 6m_3r_1^2+12m_3d_i^2 & 0 & 0 \\ 0 & 3m_3r_1^2+m_3l_1^2+12m_3d_i^2 & 0 \\ 0 & 0 & 3m_3r_1^2+m_3l_1^2 \end{bmatrix}$

$= \frac{1}{12}\begin{bmatrix} m_3(6r_1^2+12d_i^2) & 0 & 0 \\ 0 & m_3(3r_1^2+l_1^2+12d_i^2) & 0 \\ 0 & 0 & m_3(3r_1^2+l_1^2) \end{bmatrix}$, where d_i was the perpendicular distance between the i^{th} (for $i = 1$ to 4) link's principle x-axis and the x-axis of the frame at C. The complete inertia tensor at C ($\hat{P}_{PL}$) was $I_{PC} = 2I_{A\text{ at COM}} + I_{ACB\text{ at COM}} + \sum_{i=1}^{4} I_{Ptt(i)\text{ at COM}}$. The complete inertia tensor at COM F for the distal H link taking into account the torque transfer links had the same form as I_{PC}, i.e. $I_{DC} = 2I_{D\text{ at COM}} + I_{DFE\text{ at COM}} + \sum_{i=5}^{7} I_{Dtt(i)\text{ at COM}}$, with link masses m_5, length l_2 and distances d_5 to d_7 (with $d_6 = 0$).

Furthermore each torque transfer link rotated about its principle x-axis by some angular acceleration $\dot{\omega}$, the net torque on this rod was given by $I_{xx}\dot{\omega} = 0.5mr^2\dot{\omega}$. The motor torque for robot joint axes 3 to 6 was the sum of the link torques in each chain, plus the joint torque and the disc torque for the BCGM.

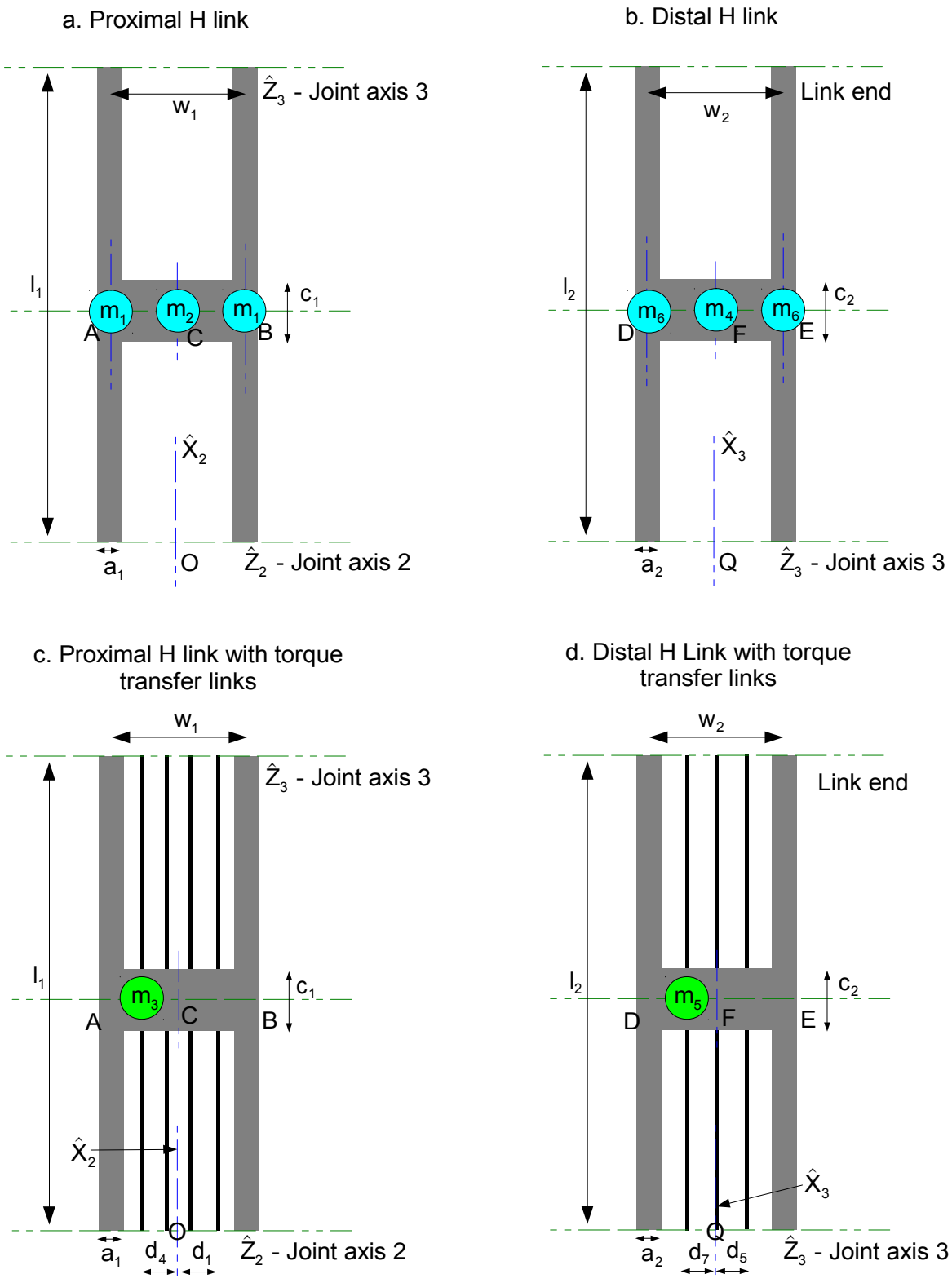


Figure 80: Inertia modelling for proximal and distal links of design 2

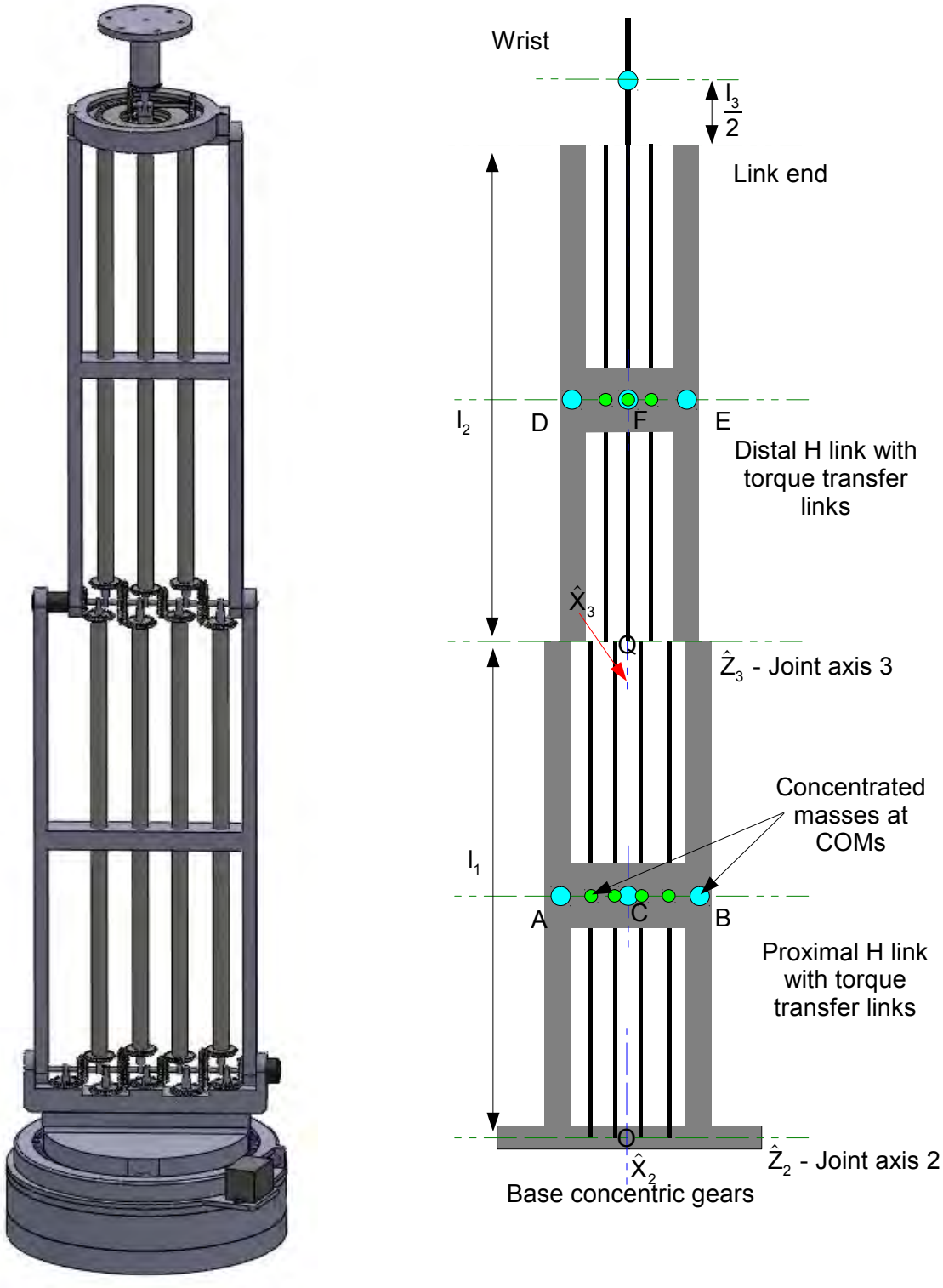


Figure 81: Comparison of CAD model to simplified dynamic model for design 2

Table 4: Dynamics parameters for design 2

i	${}^{i}\mathbf{P}_{i+1}$	${}^{i+1}\mathbf{P}_{C(i+1)}$	$\mathbf{m}_{i+1}$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$0\,{}^{0}\hat{X}_{0}$	$0\,{}^{1}\hat{X}_{1}$	$m_{bg(6)}$	$I_{bg(6)}$
1	$0\,{}^{1}\hat{X}_{1}$	$0.5l_{1}\,{}^{2}\hat{X}_{2}$	$2m_{1}+m_{2}+4m_{3}$	I_{PC}
2	$l_{1}\,{}^{2}\hat{X}_{2}$	$0.5l_{2}\,{}^{3}\hat{X}_{3}$	$2m_{6}+m_{4}+3m_{5}$	I_{DC}
3	$l_{2}\,{}^{3}\hat{X}_{3}$	$0\,{}^{4}\hat{X}_{4}$	0	$0_{3\times3}$
4	$0\,{}^{4}\hat{X}_{4}$	$0\,{}^{5}\hat{X}_{5}$	0	$0_{3\times3}$
5	$l_{3}\,{}^{5}\hat{X}_{5}$	$-0.5l_{3}\,{}^{6}\hat{Z}_{6}$	m_{w}	I_{w}

3.5.3.4.4. Simplified serial arm dynamic model

The serial robot's simplified inertia modelling used cylinders for the proximal and distal structural links, and rectangular blocks for the motors. This model also had the property that when the ${}^{0}\hat{Z}_{1}$, ${}^{0}\hat{Z}_{4}$ and ${}^{0}\hat{Z}_{6}$ axes were parallel, they were collinear. In that configuration the robot pointed vertically up. The dynamic model was made up of 4 movable parts, the rotational base, the proximal arm, the distal arm and the concentrated or simplified wrist. These were compound elements consisting of a structural part and 1 or more motors driving the immediate axis (or axes) that followed. The rotational base was composed of 2 parts the link (modelled as a disc) that attached to the proximal arm and the motor that drove the proximal arm. The complete proximal arm was composed of the structural part (i.e. the cylinder) and the motor that actuated the distal arm. Similarly the distal arm had a structural part (cylinder) and the motors that controlled the wrist (all 3 were modelled as a single block with the breadth parameter 3 times that of a single wrist motor). The wrist was modelled in the same manner as in the previous designs.

The COMs of the rotational base link, proximal and distal arms were first calculated. Those coordinates would be relative to the coordinate frame origins of those parts. The collective mass centre of link 1 (rotational disc and motor 2 at ${}^{1}\hat{P}_{L1C}$), was offset only in the y direction $\left({}^{1}\hat{Y}_{1}\right)$ from the disc COM $\left({}^{1}\hat{P}_{L1}=[0,\,0,\,0]^{T}\right)$, this was given by $\Delta\,{}^{1}\hat{P}_{L1C}=\left[0,\frac{-m_{m2}d_{1}}{m_{m2}+m_{l1}},0\right]^{T}$. The

inertia tensor of the first link's disc was $\mathrm{I}_{rs} = \frac{1}{12}\begin{bmatrix} 3m_{l1}r_{l1}^2+m_{l1}h_{l1}^2 & 0 & 0 \\ 0 & 3m_{l1}r_{l1}^2+m_{l1}h_{l1}^2 & 0 \\ 0 & 0 & 6m_{l1}r_{l1}^2 \end{bmatrix}$, with mass m_{l1}, radius r_{l1} and height h_{l1}. The inertia tensor of the second motor (actuating joint 2) modelled as a rectangular block was $\mathrm{I}_{motor2} = \frac{1}{12}\begin{bmatrix} m_{m2}(w_{m2}^2+l_{m2}^2) & 0 & 0 \\ 0 & 2m_{m2}w_{m2}^2 & 0 \\ 0 & 0 & m_{m2}(w_{m2}^2+l_{m2}^2) \end{bmatrix}$, which had mass m_{m2}, length l_{m2} (parallel to the ${}^1\hat{Y}_1$ direction), width and breadth both equal to w_{m2}. The tensors were translated to link 1's collective COM (along ${}^1\hat{Y}_1$) which gave $\mathrm{I}_{rs\ at\ COM} = \mathrm{I}_{rs}+m_{l1}\begin{bmatrix} (\Delta {}^1\hat{P}_{L1C}(2))^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & (\Delta {}^1\hat{P}_{L1C}(2))^2 \end{bmatrix}$ and $\mathrm{I}_{motor2\ at\ COM} = \mathrm{I}_{motor2} + m_{m2}\begin{bmatrix} \delta^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \delta^2 \end{bmatrix}$, where $\delta = d_1 - \Delta {}^1\hat{P}_{L1C}(2)$. $\Delta {}^1\hat{P}_{L1C}(2)$ was the second element of vector $\Delta {}^1\hat{P}_{L1C}$. The complete inertia tensor for link 1 was $\mathrm{I}_{L1\,C} = \mathrm{I}_{rs\ at\ COM} + \mathrm{I}_{motor2\ at\ COM}$.

The collective COM of the proximal arm $\left({}^2\hat{P}_{PLC}\right)$ relative to the COM of the structural portion of the proximal arm (cylinder at ${}^2\hat{P}_{PL} = [0.5l_1, 0, d_2]^T$) was $\Delta {}^2\hat{P}_{PLC} = \left[\frac{0.5m_{m3}l_1}{m_{m3}+m_{l2}}, 0, \frac{m_{m3}(|d_2|+|d_4|)}{m_{m3}+m_{l2}}\right]^T$, note the absolute values. The inertia tensor for the cylinder part of the proximal arm was $\mathrm{I}_{PLS} = \frac{m_{l2}}{12}\begin{bmatrix} 6(r_{o2}^2+r_{i2}^2) & 0 & 0 \\ 0 & 3(r_{o2}^2+r_{i2}^2)+l_1^2 & 0 \\ 0 & 0 & 3(r_{o2}^2+r_{i2}^2)+l_1^2 \end{bmatrix}$, with mass m_{l2}, outer radius r_{o2}, inner radius r_{i2} and length l_1. The inertia tensor of the third motor (actuating axis 3) modelled as a rectangular block was given by $\mathrm{I}_{motor3} = \frac{m_{m3}}{12}\begin{bmatrix} (w_{m3}^2+l_{m3}^2) & 0 & 0 \\ 0 & (w_{m3}^2+l_{m3}^2) & 0 \\ 0 & 0 & 2w_{m3}^2 \end{bmatrix}$, which had length l_{m3} (parallel to the ${}^3\hat{Z}_3$ direction), width and breadth both equal to w_{m3}.

The inertia tensors were translated to ${}^2\hat{P}_{PLC}$, using the general implementation of the

parallel axis theorem. The proximal arm cylinder's tensor translated by vector $\Delta^2\hat{P}_{PLC}$, thus $I_{PLS\ at\ COM} = I_{PLS} + m_{l2}\cdot\left[\left(\Delta^2\hat{P}_{PLC}^T\cdot\Delta^2\hat{P}_{PLC}\right).I_{3x3} - \Delta^2\hat{P}_{PLC}.\Delta^2\hat{P}_{PLC}^T\right]$. The motor tensor translated by vector $\hat{g}_1 = {}^2\left[l_1, 0, d_4\right]^T - \left({}^2\hat{P}_{PL}+\Delta^2\hat{P}_{PLC}\right)$, thus $I_{motor3\ at\ COM} = I_{motor3} + m_{m3}\cdot\left[\left(\hat{g}_1^T\cdot\hat{g}_1\right).I_{3x3} - \hat{g}_1.\hat{g}_1^T\right]$. The complete inertia tensor for the proximal arm was $I_{PLC} = I_{PLS\ at\ COM}+I_{motor3\ at\ COM}$.

The collective COM for the distal arm $\left({}^3\hat{P}_{DLC}\right)$ relative to the COM of the structural portion of the distal arm (cylinder at ${}^3\hat{P}_{DL} = \left[0.5l_{2,}0,0\right]^T$) was given by $\Delta^3\hat{P}_{DLC} = \left[\frac{-m_{m4}(0.5l_2+|d_3|)}{m_{m4}+m_{l3}},0,0\right]^T$. The inertia tensor for the structural part of the distal link was $I_{DLS} = \frac{m_{l3}}{12}\begin{bmatrix}6(r_{o3}{}^2+r_{i3}{}^2) & 0 & 0\\ 0 & 3(r_{o3}{}^2+r_{i3}{}^2)+l_2{}^2 & 0\\ 0 & 0 & 3(r_{o3}{}^2+r_{i3}{}^2)+l_2{}^2\end{bmatrix}$, with mass m_{l3}, outer radius r_{o3}, inner radius r_{i3} and length l_2. The fourth motor's inertia tensor modelled as a rectangular block was $I_{motor4} = \frac{m_{m4}}{12}\begin{bmatrix}w_{m4}{}^2+b_{m4}{}^2 & 0 & 0\\ 0 & w_{m4}{}^2+l_{m4}{}^2 & 0\\ 0 & 0 & b_{m4}{}^2+l_{m4}{}^2\end{bmatrix}$ (lumped model for 3 wrist motors), with mass m_{m4}, length l_{m4} (parallel to the ${}^3\hat{X}_3$ direction), width w_{m4} and breadth b_{m4}. The tensors were translated to the distal link's collective COM ${}^3\hat{P}_{DLC}$ using the parallel axis theorem. The distal arm's cylinder tensor was translated by vector $\Delta^3\hat{P}_{DLC}$ which gave $I_{DLS\ at\ COM} = I_{DLS} + m_{l3}\begin{bmatrix}0 & 0 & 0\\ 0 & (\Delta^3\hat{P}_{DLC}(1))^2 & 0\\ 0 & 0 & (\Delta^3\hat{P}_{DLC}(1))^2\end{bmatrix}$, where $\Delta^3\hat{P}_{DLC}(1)$ was the first element of vector $\Delta^3\hat{P}_{DLC}$ (other elements were 0). The wrist motors' tensor was translated by vector $\hat{g}_2 = {}^3[d_{3,}0,0]^T-\left({}^3\hat{P}_{DL}+\Delta^3\hat{P}_{DLC}\right)$ which gave $I_{motor4\ at\ COM} = I_{motor4} + m_{m4}\begin{bmatrix}0 & 0 & 0\\ 0 & \hat{g}_2(1)^2 & 0\\ 0 & 0 & \hat{g}_2(1)^2\end{bmatrix}$.

$\hat{g}_2(1)$ Was the first element of vector $\hat{g}_2$, and $\hat{g}_2$ was parallel to ${}^3\hat{X}_3$ since the COMs of both the fourth motor and the distal link cylinder were on ${}^3\hat{X}_3$, so $\hat{g}_2(2)$ and $\hat{g}_2(3)$ were 0. The complete inertia tensor for the distal arm was $I_{DLC} = I_{DLS\ at\ COM}+I_{motor4\ at\ COM}$.

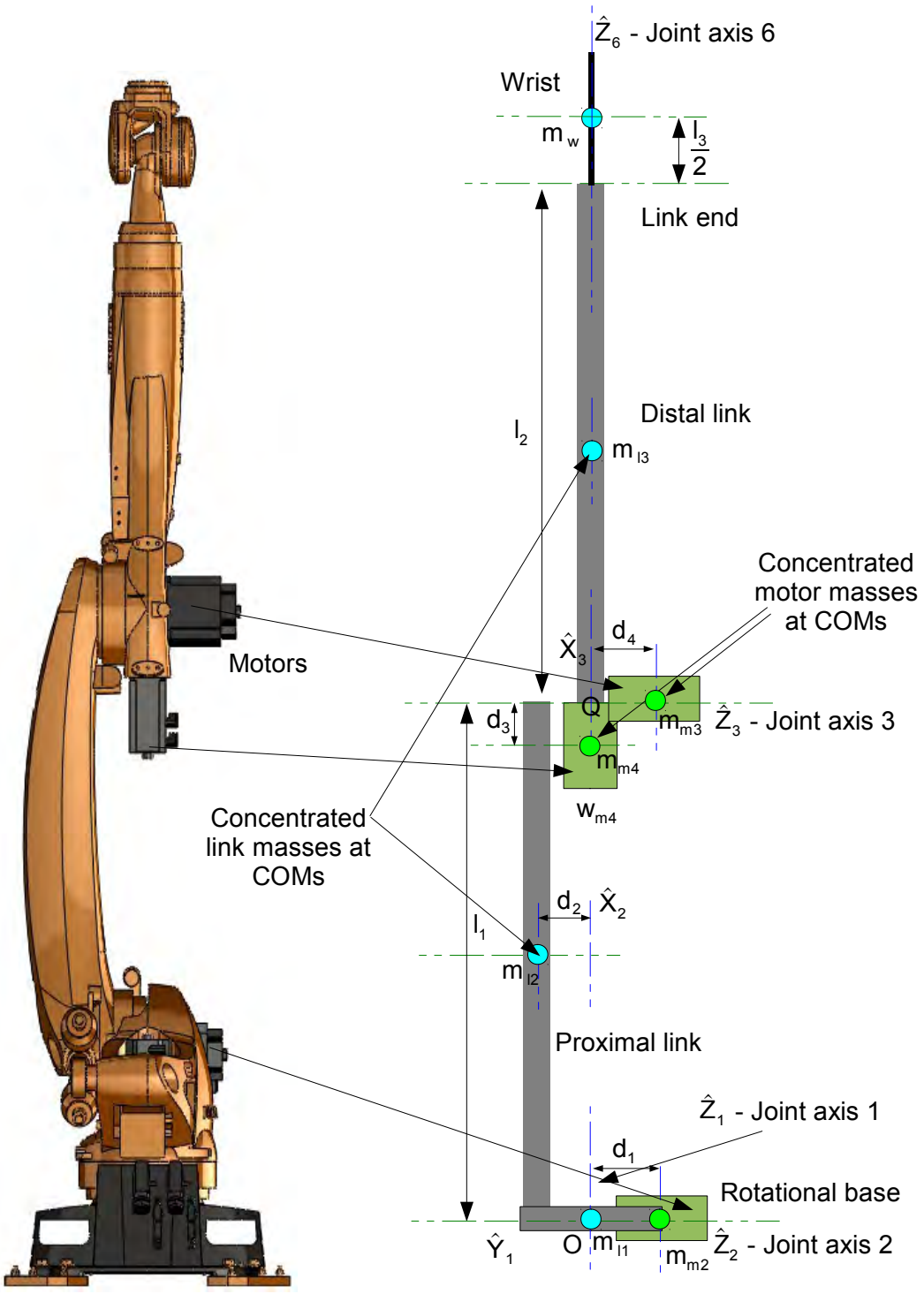


Figure 82: Simplified dynamic model of a serial robot

Table 5: Dynamics parameters for serial robot model

i	${}^{i}\mathbf{P}_{i+1}$	${}^{i+1}\mathbf{P}_{C(i+1)}$	$\mathbf{m}_{i+1}$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$0\,{}^{0}\hat{X}_{0}$	$0\,{}^{1}\hat{X}_{1}$	$m_{m2}+m_{l1}$	I_{L1C}
1	$0\,{}^{1}\hat{X}_{1}$	${}^{2}\hat{P}_{PLS}+\Delta\,{}^{2}\hat{P}_{PLC}$	$m_{m3}+m_{l2}$	I_{PLC}
2	$l_1\,{}^{2}\hat{X}_{2}$	${}^{3}\hat{P}_{DLS}+\Delta\,{}^{3}\hat{P}_{DLC}$	$m_{m4}+m_{l3}$	I_{DLC}
3	$l_2\,{}^{3}\hat{X}_{3}$	$0\,{}^{4}\hat{X}_{4}$	0	$0_{3\times3}$
4	$0\,{}^{4}\hat{X}_{4}$	$0\,{}^{5}\hat{X}_{5}$	0	$0_{3\times3}$
5	$l_3\,{}^{5}\hat{X}_{5}$	$-0.5l_3\,{}^{6}\hat{Z}_{6}$	m_w	I_w

3.5.3.4.5. Simplified dynamic model representing current patents

The designs by *Pozzi (1989)* [87] and *Bisiach (1989)* [86] were very similar, and a single simplified dynamic model representing both was used in the comparative simulations against the subject designs of this thesis and the simplified serial robot. This model will be referred to as the 'Pozzi' model, although it refers to both patents. The simplification maintained the essence of the designs but ignored working detail. As mentioned previously the actual designs had 2 DOF elbow joints, so there were 5 concentric torque transfer cylinders on the proximal link plus the outer structural cylinder; and there were 3 concentric torque transfer cylinders on the distal link plus its outer structural cylinder. For motion comparison however the elbow joint was reduced to a single DOF whose axis was always parallel to the horizontal plane. This permitted reuse of the serial kinematic model already developed. Also the common BCGM and wrist models developed for the thesis subject designs were reused here. The design's actual BCGM for the current patents had 7 concentric sections. For simplicity all concentric components fitted perfectly within one another, radial space between successive cylinders was ignored.

The model for the proximal link then had 4 concentric torque transfer cylinders plus the outer structural cylinder. The outer radii of the cylinders were $r_{p(i)}$ and $r_{d(k)}$ (for $i=1$ to 5 and $k=1$ to 4) for the proximal and distal parts respectively. The inner radii for the proximal tubes were given by $r_{p(i-1)}$ (for $i=1$ to 5), with $r_{p(0)}$ being the inner radius of the first tube. Similarly

for the inner radii of the distal cylinders. The model for the concentric gears at the elbow joint had 4 sections, with outer radii $r_{eg(j)}$ (for $j=1$ to 4), and corresponding inner radii $r_{eg(j-1)}$. The actual elbow concentric gears had 5 sections. Also for the model, the axes of those gears would remain parallel to the axes of the concentric cylinders on the proximal link for all motion.

Essentially this model would produce dynamic results that were better than the most optimised version of the actual robot, since it had less components and less mass to move. With reference to figure 83, the COM of the proximal link's cylinder part was at its mid-point (${}^2\hat{P}_{PL} = [0.5l_1, 0, d_1]^T$), similarly for the distal link (${}^3\hat{P}_{DL} = [0.5l_2, 0, d_3]^T$) and concentric elbow gears (${}^2\hat{P}_{EG} = [l_1, 0, d_2]^T$). Since the cylinders rotate within each other, the COMs of the complete proximal and distal links did not change relative to their respective reference coordinate frames. For that reason the motor responsible for controlling motion about the first axis (${}^0\hat{Z}_1$axis) perceived the complete proximal and distal links as solid bodies. The cylinder parts of each then have outer radii $r_{p(5)}$ and $r_{d(4)}$, and inner radii $r_{p(0)}$ and $r_{d(0)}$ respectively. Similarly for the elbow concentric gearbox. That motor must also account for the torque required to move the supporting base link in the BCGM (with mass $m_{bg(6)}$ and tensor $I_{bg(6)}$). Motor 2 which controlled the elevation of the complete proximal arm perceived it, the elbow's concentric gearbox and distal link as solid bodies. In addition it had to account for its corresponding gear in the BCGM. Motor 3 which controlled the angle between the proximal and distal links, perceived the distal link as a solid body. It also had to account for its corresponding torque transfer cylinder in the proximal link, and gears in both the elbow and base concentric gearboxes. The 3 motors that orient the wrist had to account for their corresponding torque transfer cylinders in both the proximal and distal arms as well as their corresponding gears in the elbow and base concentric gearboxes.

Each of the 5 cylinders in the proximal arm had mass $m_{p(i)}$, outer radius $r_{p(i)}$, inner radius $r_{p(i-1)}$, and length l_1; their inertia tensors were given by

$I_{p(i)} = \frac{m_{p(i)}}{12}\begin{bmatrix} 6\left(r_{p(i)}{}^2+r_{p(i-1)}{}^2\right) & 0 & 0 \\ 0 & 3\left(r_{p(i)}{}^2+r_{p(i-1)}{}^2\right)+l_1{}^2 & 0 \\ 0 & 0 & 3\left(r_{p(i)}{}^2+r_{p(i-1)}{}^2\right)+l_1{}^2 \end{bmatrix}$ (i=1 to 5). Similarly the 4 cylinders in the distal link had masses $m_{d(k)}$ and inertia tensors $I_{d(k)} = \frac{m_{d(k)}}{12}\begin{bmatrix} 6\left(r_{d(k)}{}^2+r_{d(k-1)}{}^2\right) & 0 & 0 \\ 0 & 3\left(r_{d(k)}{}^2+r_{d(k-1)}{}^2\right)+l_2{}^2 & 0 \\ 0 & 0 & 3\left(r_{d(k)}{}^2+r_{d(k-1)}{}^2\right)+l_2{}^2 \end{bmatrix}$ (k=1 to 4). The elbow gearbox had sections with masses $m_{eg(j)}$ and inertia tensors $I_{eg(j)} = \frac{m_{eg(j)}}{12}\begin{bmatrix} 6\left(r_{eg(j)}{}^2+r_{eg(j-1)}{}^2\right) & 0 & 0 \\ 0 & 3\left(r_{eg(j)}{}^2+r_{eg(j-1)}{}^2\right)+h^2 & 0 \\ 0 & 0 & 3\left(r_{eg(j)}{}^2+r_{eg(j-1)}{}^2\right)+h^2 \end{bmatrix}$ (j=1 to 4), where h was the height of the gears. Here h was set as a constant but in reality, like the cylinder length, they must be staggered in order to mesh.

The cylinder part of the proximal arm had mass $m_P = \sum_{i=1}^{5} m_{p(i)}$ and inertia tensor $I_P = \frac{m_P}{12}\begin{bmatrix} 6\left(r_{p(5)}{}^2+r_{p(0)}{}^2\right) & 0 & 0 \\ 0 & 3\left(r_{p(5)}{}^2+r_{p(0)}{}^2\right)+l_1{}^2 & 0 \\ 0 & 0 & 3\left(r_{p(5)}{}^2+r_{p(0)}{}^2\right)+l_1{}^2 \end{bmatrix}$. The elbow gearing was part of the complete proximal link had total mass $m_{EG} = \sum_{j=1}^{4} m_{eg(j)}$ and inertia tensor $I_{EG} = \frac{m_{EG}}{12}\begin{bmatrix} 6\left(r_{eg(4)}{}^2+r_{eg(0)}{}^2\right) & 0 & 0 \\ 0 & 3\left(r_{eg(4)}{}^2+r_{eg(0)}{}^2\right)+h^2 & 0 \\ 0 & 0 & 3\left(r_{eg(4)}{}^2+r_{eg(0)}{}^2\right)+h^2 \end{bmatrix}$. The complete mass of the proximal arm was $m_{PT} = m_P + m_{EG}$. The collective COM for the complete proximal arm $\left({}^2\hat{P}_{PLC}\right)$ relative to the COM of the cylinder part $\left({}^2\hat{P}_{PL}\right)$ was $\Delta^2\hat{P}_{PLC} = \left[\frac{0.5 l_1 m_{EG}}{m_{PT}}, 0, \frac{(d_2-d_1)m_{EG}}{m_{PT}}\right]^T$. Tensors were translated to ${}^2\hat{P}_{PLC}$ using the general formula for tensor translation. The cylinder part translated by vector $\Delta^2\hat{P}_{PLC}$ which gave $I_{P\ at\ COM} = I_P + m_P\cdot\left[\left(\Delta^2\hat{P}_{PLC}^T\cdot\Delta^2\hat{P}_{PLC}\right).I_{3x3} - \Delta^2\hat{P}_{PLC}.\Delta^2\hat{P}_{PLC}^T\right]$. The tensor for the elbow gears translated by vector $\hat{g}_1 = {}^2\hat{P}_{PL} - {}^2\hat{P}_{EG} - \Delta^2\hat{P}_{PLC}$, and was given by

$I_{EG\ at\ COM} = I_{EG} + m_{EG}\cdot\left[\left(\hat{g}_1^T\cdot\hat{g}_1\right).I_{3x3} - \hat{g}_1.\hat{g}_1^T\right]$. Hence the inertia tensor for the complete proximal arm was $I_{PLC} = I_{P\ at\ COM} + I_{EG\ at\ COM}$.

The distal link had a total mass of $m_{DT} = \sum_{i=1}^{4} m_{d(i)}$, and inertia tensor $I_{DT} = \frac{M_{DT}}{12}\begin{bmatrix} 6\left(r_{d(4)}^2 + r_{d(0)}^2\right) & 0 & 0 \\ 0 & 3\left(r_{d(4)}^2 + r_{d(0)}^2\right) + l_2^2 & 0 \\ 0 & 0 & 3\left(r_{d(4)}^2 + r_{d(0)}^2\right) + l_2^2 \end{bmatrix}$ at its COM ${}^3\hat{P}_{DL}$.

Table 6: Dynamics parameters for 'Pozzi' model

i	${}^{i}\mathbf{P}_{i+1}$	${}^{i+1}\mathbf{P}_{C(i+1)}$	$\mathbf{m}_{i+1}$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$0\,{}^0\hat{X}_0$	$0\,{}^1\hat{X}_1$	$m_{bg(6)}$	$I_{bg(6)}$
1	$0\,{}^1\hat{X}_1$	${}^2\hat{P}_{PL} + \Delta\,{}^2\hat{P}_{PLC}$	m_{PT}	I_{PT}
2	$l_1\,{}^2\hat{X}_2$	$\hat{P}_{DL}$	m_{DT}	I_{DT}
3	$l_2\,{}^3\hat{X}_3$	$0\,{}^4\hat{X}_4$	0	$0_{3\times3}$
4	$0\,{}^4\hat{X}_4$	$0\,{}^5\hat{X}_5$	0	$0_{3\times3}$
5	$l_3\,{}^5\hat{X}_5$	$-0.5l_3\,{}^6\hat{Z}_6$	m_w	I_w

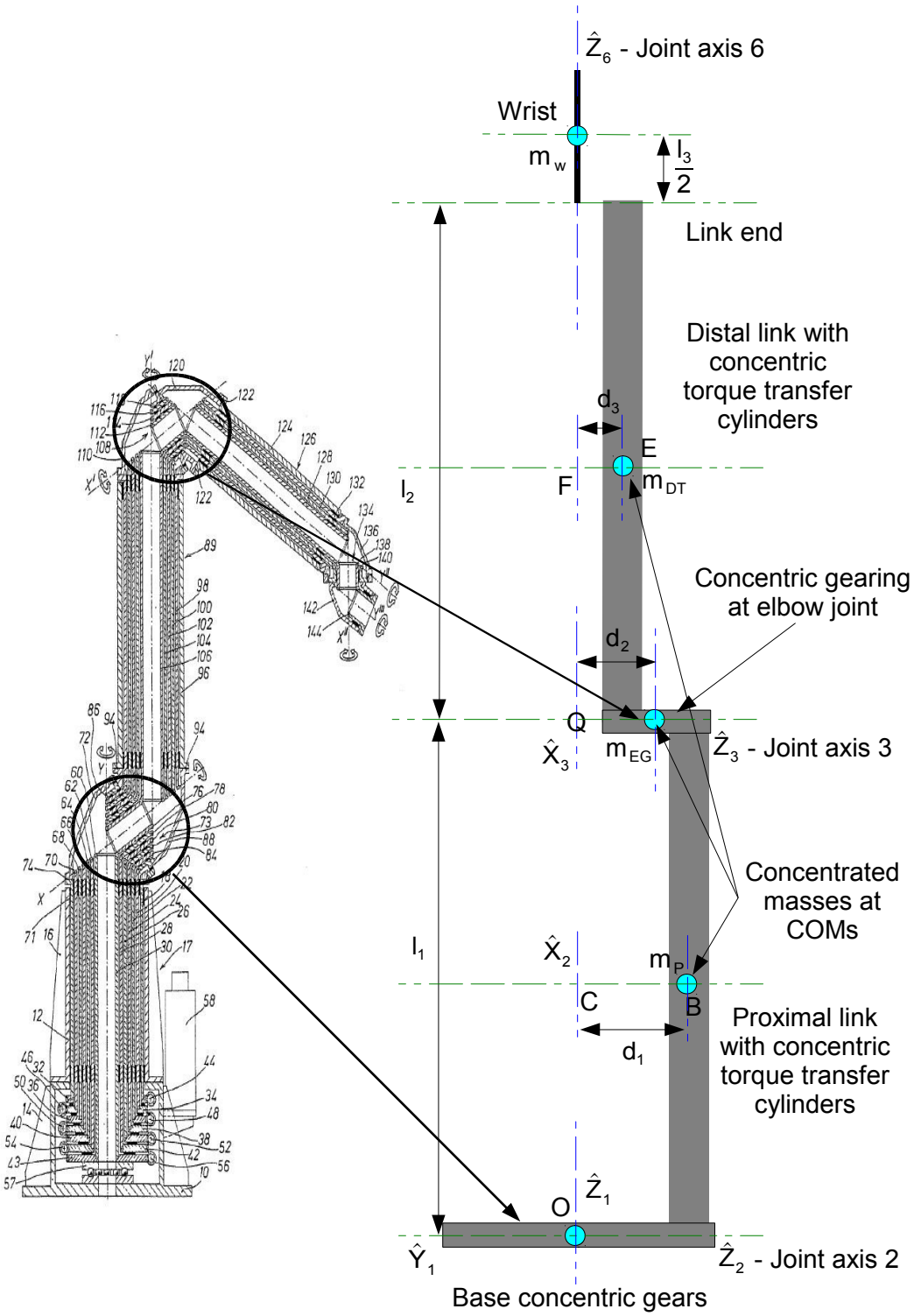


Figure 83: Simplified dynamic model for current patents

3.6. Chapter summary

This chapter discussed 2 low inertia 6 DOF robot design concepts with a large workspace to footprint ratio and high dexterity. Both designs had all their motors fixed at the stationary base, and used multiple links and gears to transfer torque to the correct axes. The low inertia was expected to help reduce the robot's energy usage and with performance metrics such as speed, reach and payload. In the prior art search 2 designs were found that accomplished the same goal, these were from *Bisiach (1989)* [86] and *Pozzi (1989)* [87]. Both patents fixed all actuators at the stationary base and used coaxial tubes and concentric geared mechanisms to control the main robot joints. These designs however had a few significant disadvantages, which included greater costs in manufacture and maintenance, they used more material, needed an extra DOF (an extra motor), had a lower energy efficiency and lower accuracy. Methods were presented which could improve robot accuracy, in spite of the number of gears used. These included harmonic and cycloidal drives; self-locking gears and braking mechanisms. It would be difficult to incorporate these into the *Bisiach (1989)* [86] and *Pozzi (1989)* [87] designs. The thesis subject designs were discussed and illustrated in detail, and the mechanical designs were then modelled mathematically, starting with the 3 bar slider-pivot linkage. A common robot kinematic model was created for both designs, since in both it was possible to align the $\hat{z}_1$, $\hat{z}_4$ and $\hat{z}_6$ axes, and the major link lengths could be the same. The forward and inverse kinematics were solved completely, and used a mixed geometric-algebraic approach which provided greater insight and a better understanding. For each robot pose the inverse kinematics solution gave 8 different sets of joint angle solutions. The dynamics algorithm was then discussed and dynamic models were created for these as well as the serial robot and 'Pozzi' design. The inertia tensors and COMs which were calculated from those simplified models, together with velocities and accelerations, were then used in the dynamics simulations to calculate forces, torques and joint work. These were used to determine the most energy efficient design.

4. Scaled functional model

A scaled functional model was built using perspex as a construction material. The reasons for this were that the material was cheap, a laser cutter was readily available at no cost and changes to the design could be implemented quickly. To drive the links and gears, model helicopter servo motors were used as the actuators. These had to be modified so that the motor (and gearbox) could be placed at the stationary base with its sensor at the joint that it controls (please see appendix D modifying servos). A generic controller board with an Atmel ATMEGA128 micro controller was sufficient for communications and PWM signal generation for servo motor control. A modified PC power supply was used to power everything, and instructions on how to make those changes can be found on the internet.

4.1. Motor control

Six PWM (pulse width modulated) signals that had to be generated to control the 6 servo motors, which in turn controlled the 6 DOFs of the model. The width of these square waves were in the order of 1 to 2ms. A generic controller board (with an ATMega128 micro controller) was used to communicate with the user interface software on a laptop via the RS232 port, and to generate those PWM signals. The signals were generated by the 16 bit timers and 2 16 bit timer compare interrupts. The 16 bit timer register increments on a time interval equal to the inverse of the clock frequency (at 16 MHz, it was a period of 0.0625 micro seconds). When the value stored in the compare register equalled the value in the timer register (hardware comparison) an interrupt was triggered, and the flow of program execution jumped to the timer compare interrupt service routine, where the output of a single PWM pin was raised to a 'high' (logic 1 or 5 volts). Each compare interrupt generated 3 PWM signals, and it kept track of each signal through a counter. On the timer overflow interrupt, i.e. when the timer register reached 65535 (0xFF in hexadecimal, all 16 bits were 1), the overflow interrupt service routine executed. That routine incremented the PWM tracking counter (if it exceeded 2 it was reset to 0) and cast down all PWM pin outputs to a 'low' (logic

0 or 0 volts).

The user interface was written in Matlab, and had 6 vertical bars that the user could select by pressing numbers 1 to 6 on the keyboard, to control a specific servo. The up and down arrow keys increased and decreased the angle respectively, and was used for control in the forward kinematics sense only. The forward control was sufficient to test the scaled model. The selected angle (0º to 200º) was then converted to a value for the micro's 16 bit timer (16000 to 32000) and split into a number of ASCII characters that were transmitted through the RS232 serial port. The micro controller received those characters and built up the value for the respective timer compare interrupt, and stored it in that register.

4.2. Servo drives and control for a real, full-scale robot

The term servo applies to a 'servo motor' but is also a general control term indicating that a feedback loop was used to position something. Industrial servos come in 2 parts the servo drive and the servomechanism (includes a large class of actuators not only motors). The servo drive is an electronic amplifier that both powers and monitors a feedback signal (continually adjusting deviations from a desired reference through negative feedback) in electric servomechanisms. The feedback signal is generated from a sensor either a potentiometer, digital encoder or resolver that is attached at the output of the servomechanism. A resolver uses a second set of rotor and stator coils called the transformer to induce rotor voltages across an air gap. It does not have any electronic components, functions at high temperatures, and is inherently shock-resistant thus making it ideal for usage in harsh environments. The feedback sensor primarily monitors position, but the servo could be configured to control position, speed or torque, depending on what the user wants to control. The servo motor could be a DC, AC, or a brushless DC motor. Some servomechanisms have a mechanical stop on the output shaft limiting rotation whereas other systems allow continuous rotation.

The industrial drive's command signal is usually in the form of a coded pulse (pulse width modulation or pulse coded modulation), which is then amplified controlling electric current to

the servo motor which induces motion. The width of the pulse indicates the desired angle. Other systems use a set of ascii characters that indicate the desired position in direct human readable form. The feedback circuitry most generally implements a PID (proportional, integral and derivative) loop, and the gains for each are programmable allowing the user to tune the feedback control loop.

Servo systems can be used in industrial machine tools, CNC machines, factory automation, packaging applications, robotics, etc., generally across a number of fields where the motor must be able to operate at a range of speeds and to hold its load at a required position when stationary without overheating.

The main differences in control between the model helicopter servo and an industrial servo are:

- The command signal may be a set of ascii codes. This isn't a major problem as the angular commands can be sent directly out a host computer's communication port. If the servo uses PWM then no changes are necessary.
- Industrial servos use PID control that the user can tune (adjust the gain parameters) to the required application. The model helicopter servos use proportional control only, and the single gain parameter is fixed.
- Model servos use potentiometers only as the feedback sensor.

4.3. Scaled model of design 1

All mechanical components i.e. links, gears and structural elements were laser cut from 2D sheets of Perspex, and 2D components were then assembled into 3D structures. The design required bevelled gears, but to work around that obstacle the spur gear teeth were made large enough so that they could mesh at 90º and in this case their pitch circles would be tangent at 90º (the pitch circles lie on 2 separate planes that were perpendicular to each other). This adaptation functioned adequately, in spite of the fact that the gears made point contact instead of line contact. For the scaled model this was sufficient. On submission of

this thesis the model had 4 working DOFs, 3 which position the wrist and 1 to orient it.

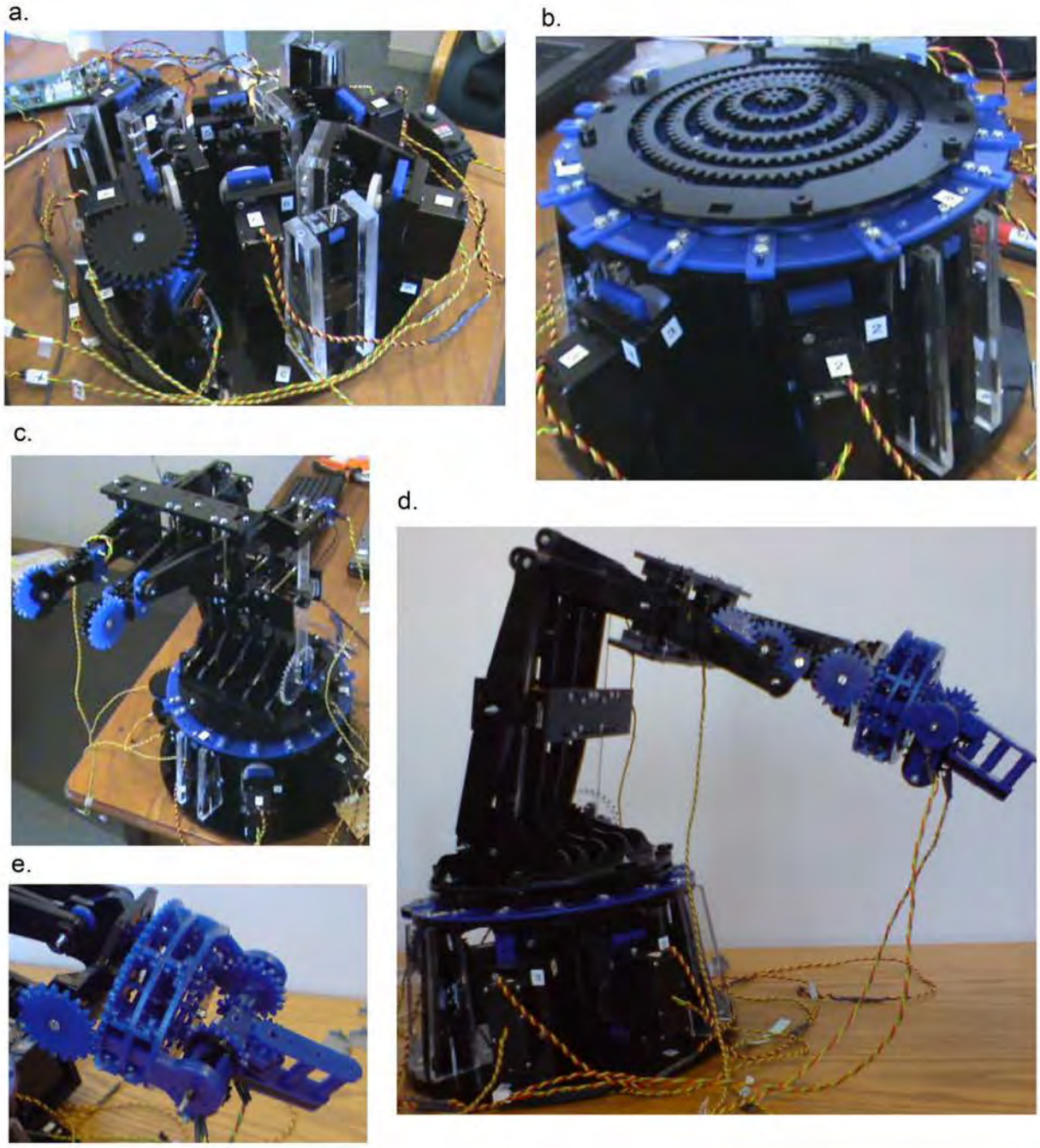


Figure 84: Perspex model of design 1

a. Motors at the base
b. Concentric gears placed on top of motors
c. Arm linkage
d. Complete model
e. Wrist

The last 2 DOFs could not be completed as the improvised Perspex bearings for the concentric geared mechanism at the wrist became problematic. However the physical model

of the 3 DOF wrist worked and oriented the makeshift end-effector as desired. It was not a spherical wrist since it was difficult to position everything (sensors and gears) in a compact space that would allow intersection of all wrist axes. This wasn't a problem as the scaled model was controlled in the forward kinematics sense only, and the inverse kinematics which would be difficult to solve for the non-spherical wrist was not required.

4.4. Chapter summary

This short chapter served to highlight the most important aspects in the build of a scale model for design 1. The model was made using cheap materials and actuators, and served to demonstrate that this arrangement of geared mechanisms and 3 bar slider-pivot linkages could produce a functioning robotic arm. Multiple configurations could be prototyped in this way at low cost to find the most suitable design option.

The Perspex model highlighted some structural problems, and the need for motor axis support at the base. The laser inaccuracy also magnified backlash problems, and since no high accuracy non-back-drivable gearboxes were used, this affected end-effector motion. To combat this the model was driven slowly, and the gears where then scaled minutely to force a closer mesh. Eventually the changes made allowed the scaled model to work, and 4 DOFs were completed. A video of the forward kinematic control is attached in the accompanying CD, which shows how the model worked at each stage in construction.

5. Simulation results

To run the simulation code and to understand what each part was for please see appendix A.

5.1. Design 1: Primary 3 bar slider-pivot linkage

The first thing to determine was the best position for the slider-pivots, on the primary 3 bar slider-pivot linkage. It would also indicate the best position on the secondary 3 bar slider-pivot linkages. A simplifying relationship between the comprising links, AB, AC, BD and the pivot position P, shown in figure 85 was required as all could vary. A normalization (parametrization) about link AB or the driving link was chosen, and the length of AB was set to 1. AC and BD were some factors q and r larger than AB ($AC = q.AB = q$ and $BD = r.AB = r$, as $AB = 1$, where q and r were > 1). Certain errors were then defined with reference to figure 85.

The normalized instantaneous position error of point D was defined as DE/AB (since AB was 1, its just DE). Point E was on the intersection of CD (extended if necessary) and a circle with radius of length AB centred at C. The length DE was positive if E was outside of CD and negative if E was within CD. The average error was defined as the sum of the normalized instantaneous position errors as AB rotated from 0º to 360º (the angle with respect to a fixed horizontal AC, in an anti clockwise sense about A), averaged over the number of angular positions used. The square mean error was defined as the sum of the squares of the normalized instantaneous position errors over the number of angular positions used.

There was both an upper and lower limit on the size of link BD with regard to link AC. If $BD = AC + AB = AB(1+q) = AB.r$, then the orbit of D passed through C. If $r > (q+1)$ then the entire orbit of D was to the right of a vertical line through C, shown in figure 86a. This meant that the link which would be joined at C and controlled by B would oscillate about C at twice the frequency of the driver link AB. The larger r became with respect to $(q+1)$ the smaller the

oscillation angle got as the orbit of D moved further away from C. That would defeat one of the design objectives, which was a large range of motion of the arm's main links. The rotation of link AB controlled the angle of the link connected to joints C and D, there should be a 1 to -1 rotation matching with regard to those links, hence $r<(q+1)$ was a necessary condition. Similarly if $r=(q-1)$, the orbit of D passed through C and if $r<(q-1)$ then the entire orbit of D was to the left of a vertical line thorough C, shown in figure 86b, and the link that connected to joint C controlled by D would oscillate to the left of C. Hence $r>(q-1)$ and the bounds for r with respect to q was $q-1 < r < q+1$.

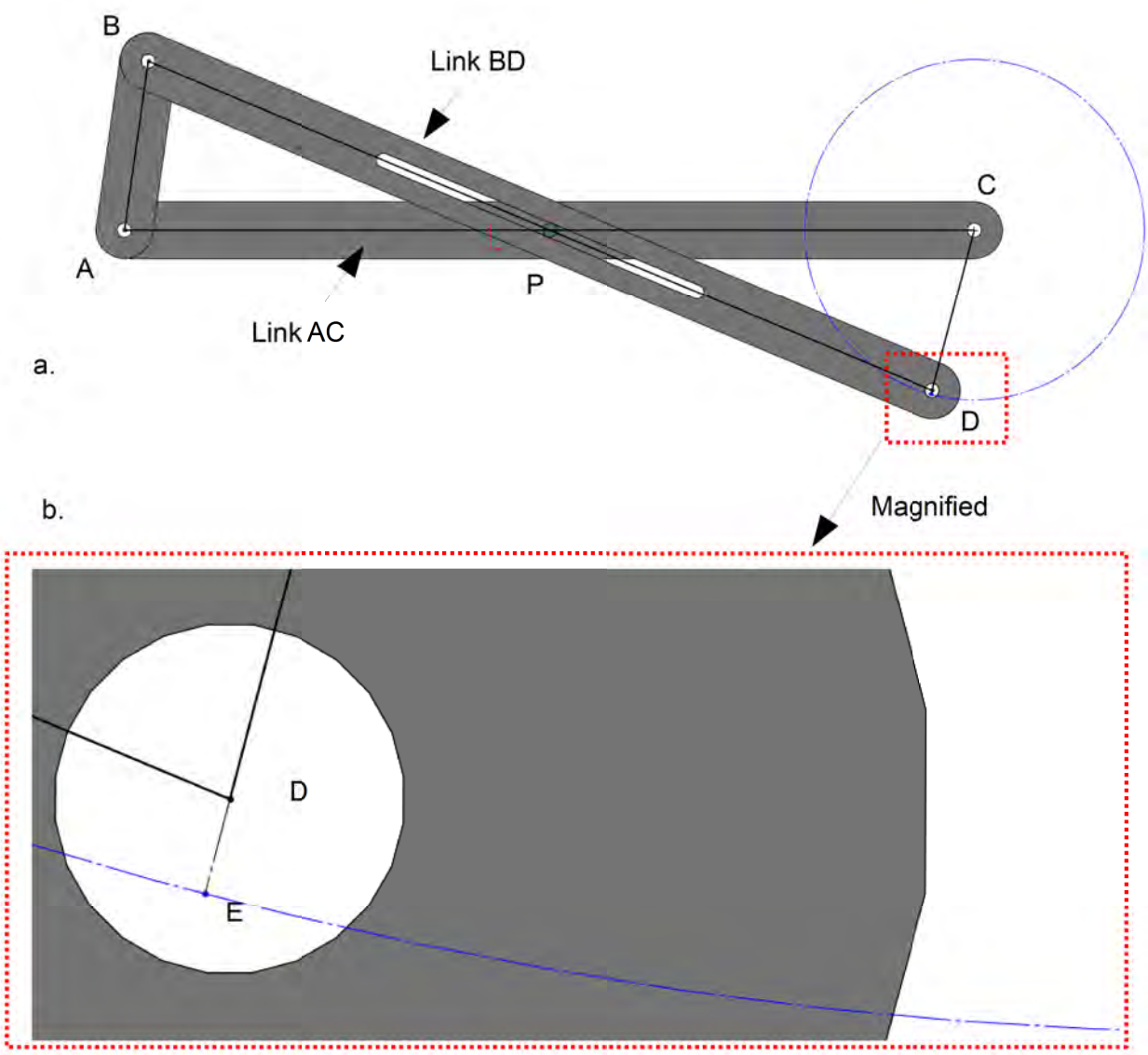


Figure 85: 3 Bar slider pivot linkage

a. Illustration depicting links AB, AC, BD and virtual link CD, and circle of radius AB at C
b. Magnification of joint D, and trajectory error DE

There were also limits imposed on the location of pivot point P with regard to link AC. If length AP was less than AB then the orbit of D circumnavigated A, and again a complete revolution of AB would cause an oscillation of the link joining C and D, in addition to being very large. Also link AB would have to pass through the slider-pivot P, which it cannot. See figure 87a on page 147. If $AP>(AC-CD)$ when angle BAD was 180° (at that angle $AC=AB$) then link AB could not make a full 360° rotation as link BD would then move past the forced limitation imposed by P. Hence $AB < AP < (AC-CD)$. See figure 87b.

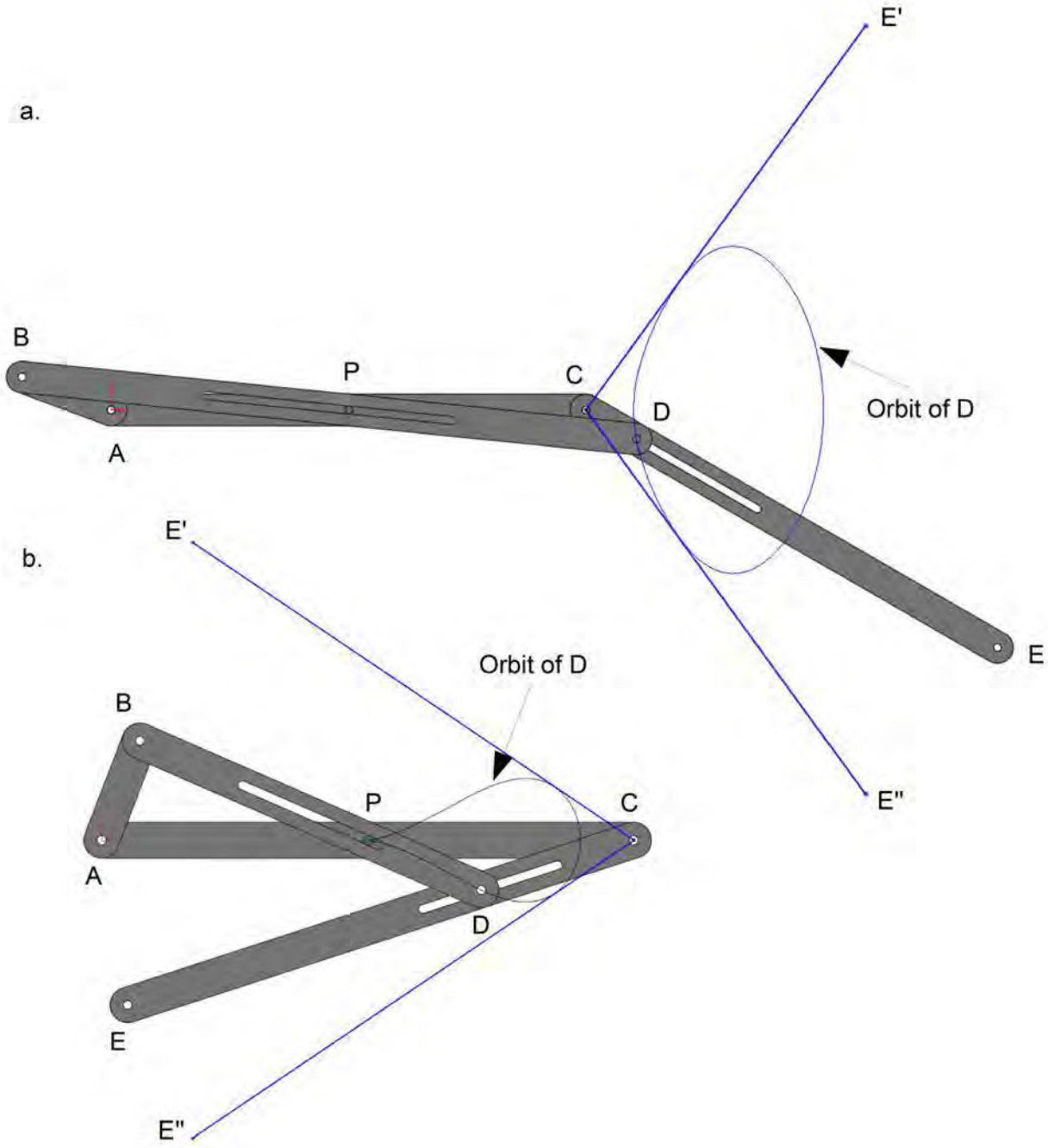
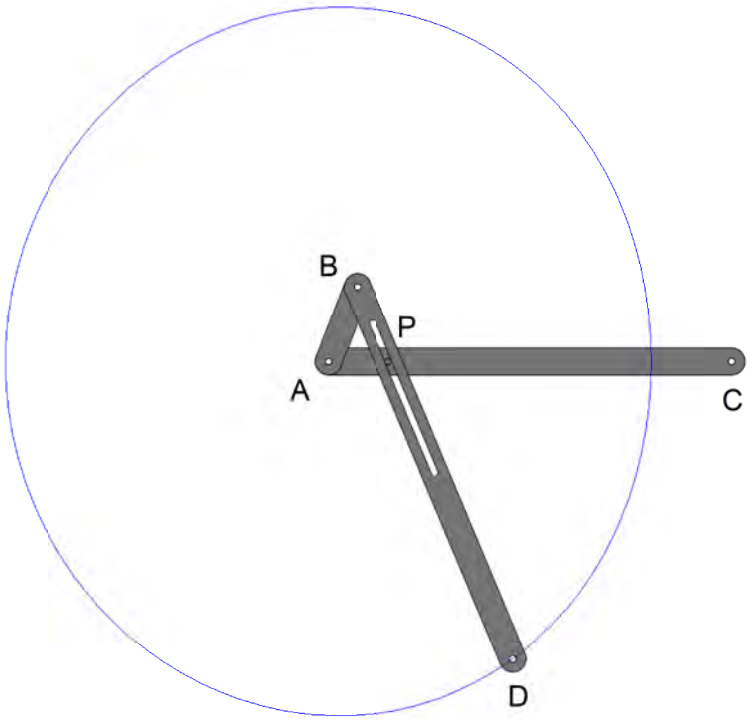


Figure 86: Orbit of D indicating limited range of motion

a. CE limited motion when r > (q + 1)

b. CE limited motion when r < (q - 1)

a.



b.

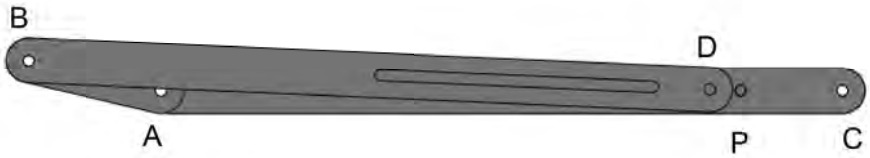


Figure 87: Outer bounds of slider-pivot position

a. When length AP < AB
b. When (AC - CD) < AP (here AB cannot make a full 360° rotation)

The position of the pivot, and magnitudes of q and r determine the maximum normalized instantaneous position error. Varying r against q did not make the orbit of D more regular, so r and q were made equal. A 3D surface plot was used to visualize and compare the orbits/trajectories produced for output point D (on a complete rotation of the input link) as the pivot position was varied between its limits. The XY data was stacked vertically with the Z axis indicating pivot position AP. Additionally the orbits produced for 5 selected pivot positions was shown on a separate 2D plot. The results for $q = r = 8$ are shown in figure 88.

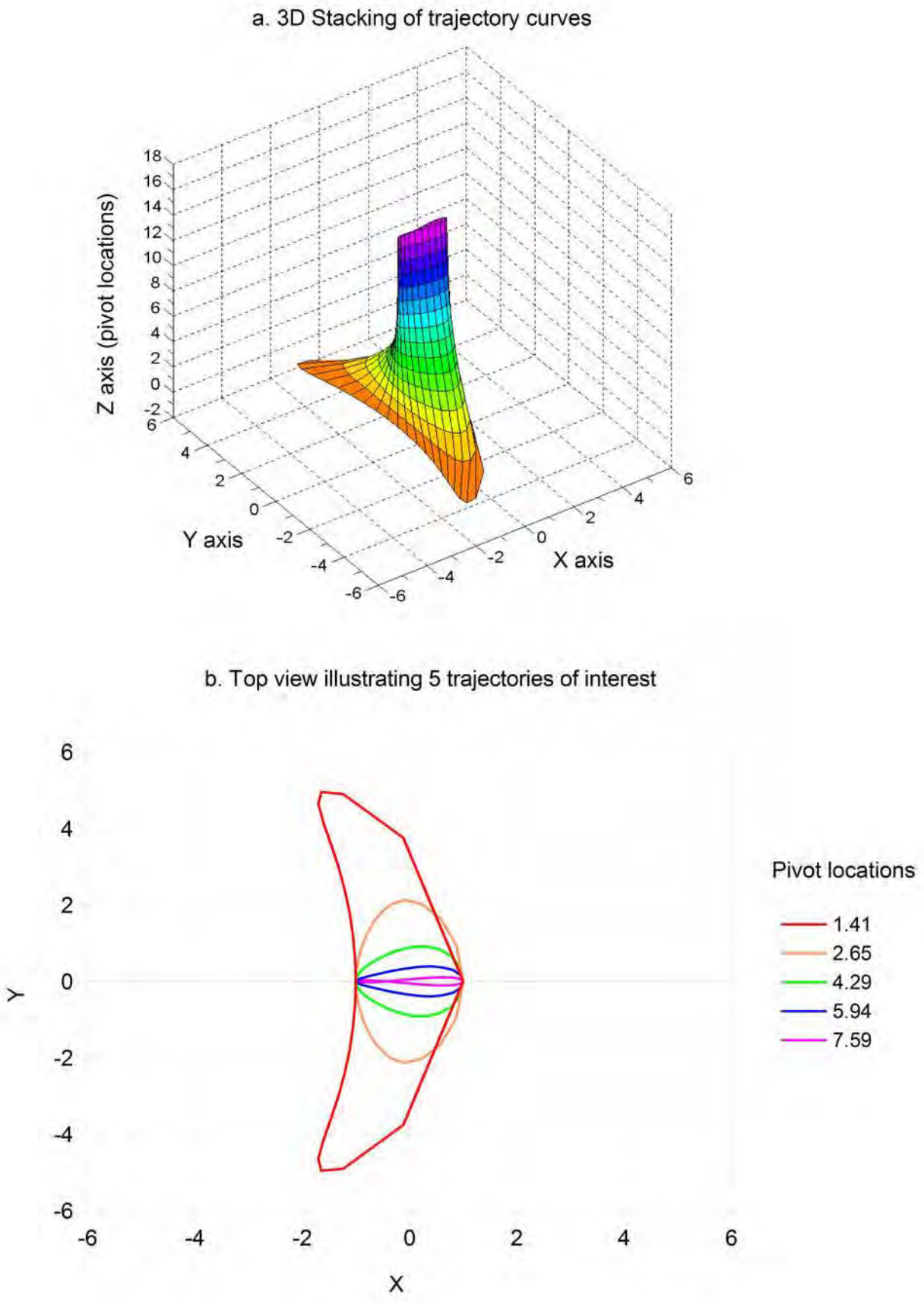


Figure 88(a,b): Output trajectories of 3 bar slider-pivot linkage

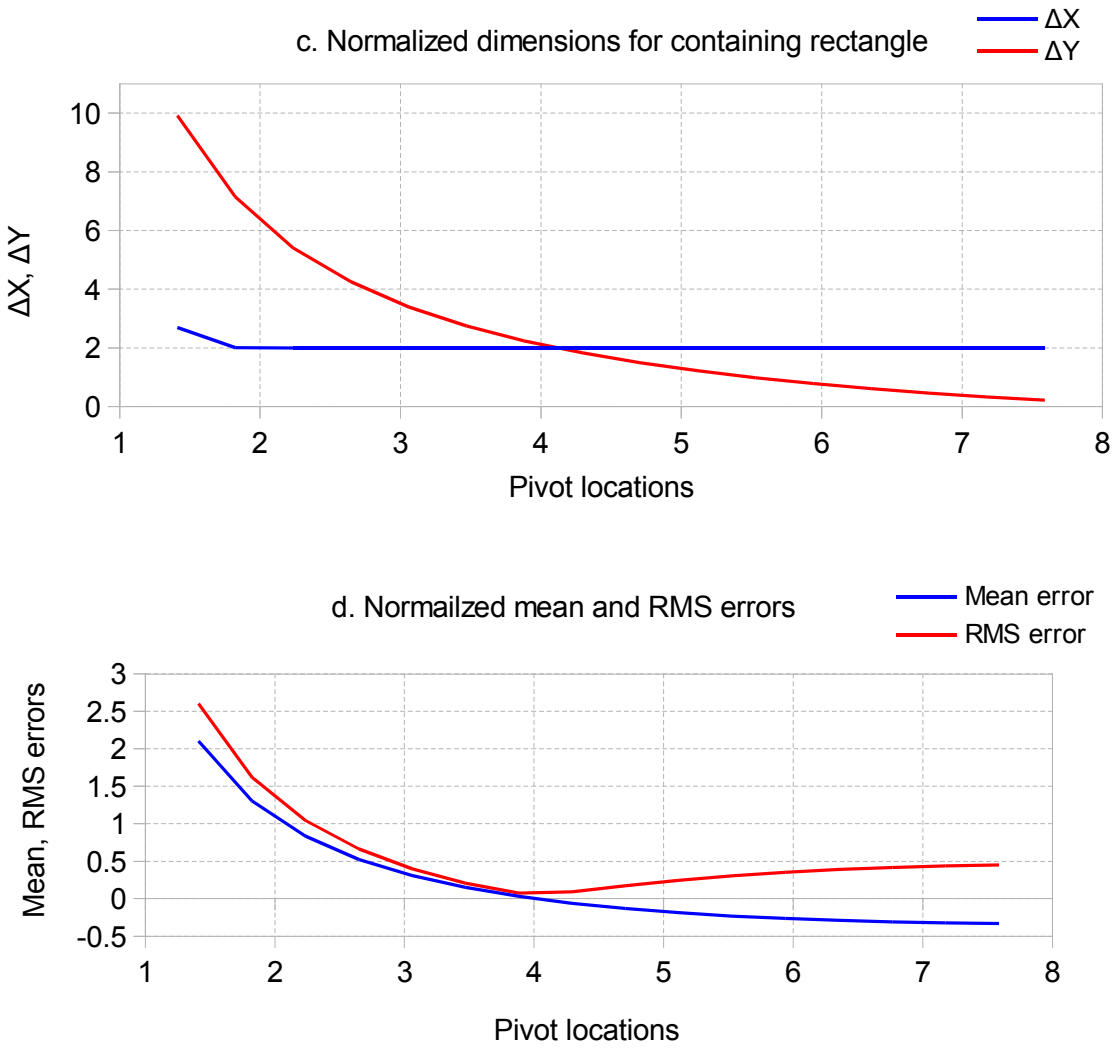


Figure 88(c,d): Output trajectories of 3 bar slider-pivot linkages

From figure 88(a, b) it could be seen that when the pivot was close to the axis of the driver link, the output trajectory got larger and when it was further away from that axis it got smaller. Figure 88(c, d) illustrates this in another way, through dimensions of a containing rectangle and mean and RMS errors. The containing rectangle was derived by taking the maximum x and y values for each trajectory and subtracting its respective minimum. Once the orbit became circular shaped the Δx value remained constant at twice the driver link length. The value of Δy decreased in a hyperbolic fashion and got very close to 0 as the pivot location approached the link end point. For the graphs illustrated the pivot position was pushed past its physical limit where $AP > (AC - CD)$. $\Delta x = \Delta y$ Just after the mid-point of the supporting link.

Also the average error was 0 at the midpoint 4 and the RMS error was at its minimum at that point. A second criterion for the output was that it track the input in as much a linear fashion as possible. For that reason the pivot had to be at the supporting link's mid-point. The orbit at that pivot location was examined closer, and was compared to the perfect circle orbit of its driver link. Figure 89a shows the output trajectory superimposed on a perfect circle. Figure 89b displays the difference in radial distance between the 2 trajectories, and indicates a sinusoidal pattern. The 4 places at which this graph cuts the 0 axis correspond to the 4 intersections (of the circle and output orbit) in the previous graph. The maximum radial distance was 111% of the input driver link's length, and the minimum radial distance was 92% of that length. The output angle given by DCA in an anti-clockwise sense about CA, sped up and slowed down even for a constant input angular speed. This is indicated in figure 89(c-f). The output angle was plotted against the input angle (CAB in an anti-clockwise sense about CA), and was compared to a perfect circle output angle that tracked its input angle in a perfect 1 to -1 ratio (the negative implies a change in rotational direction). The output angle lead-lag when compared to the perfect circle output is illustrated more clearly in figure 89d. A measure of the output angular speed was given by the difference in output angle for a uniform change in input angle, this is shown in figure 89e. To obtain the actual angular speed this would be multiplied by the input angular speed. The average speed across the input range must equal -1 (output angle degree per input angle degree). Lastly a measure of the output acceleration is shown in figure 89f. The mid-point pivot gave the flattest curves for angular speed and acceleration for the primary 3 bar slider-pivot linkage.

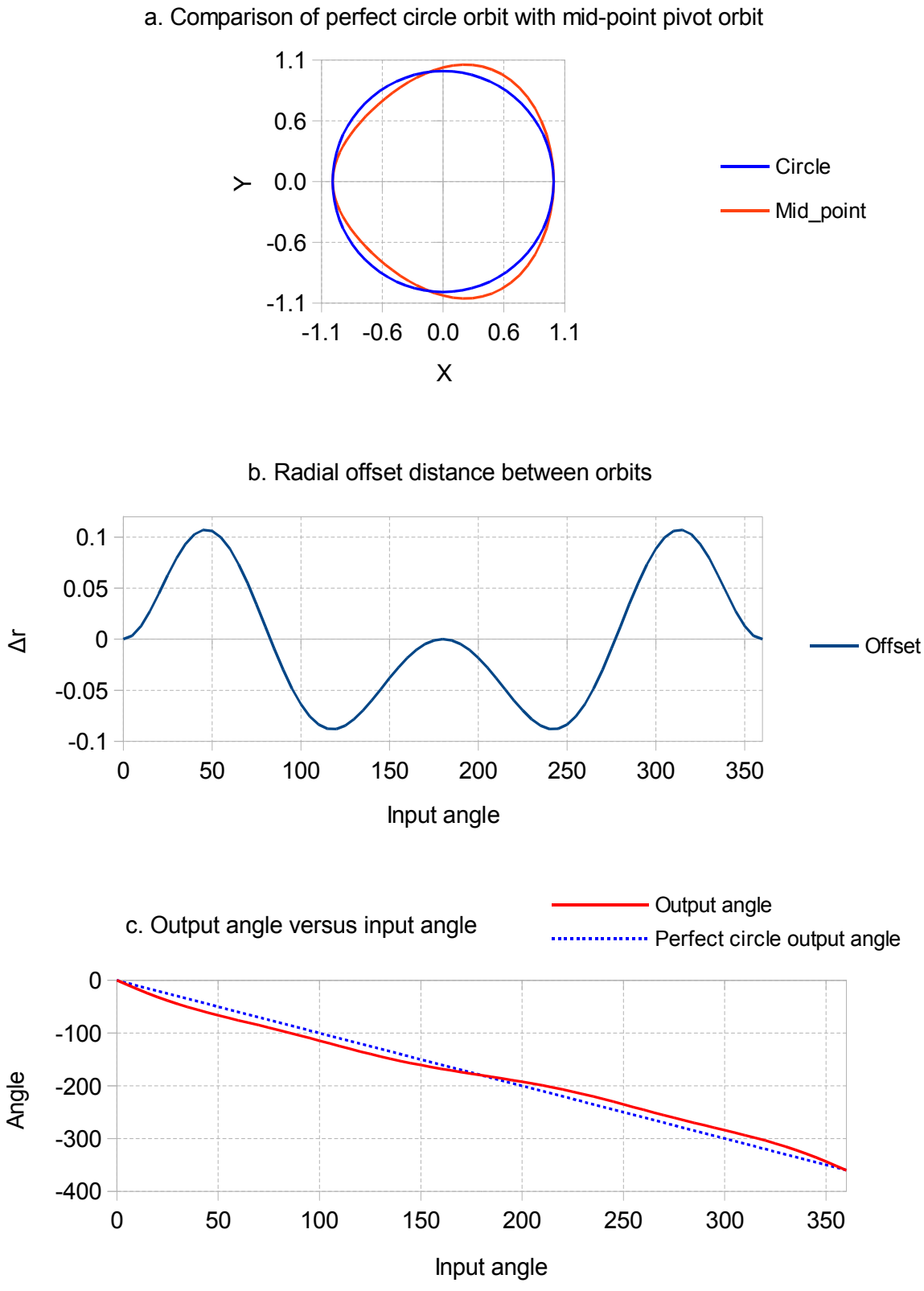
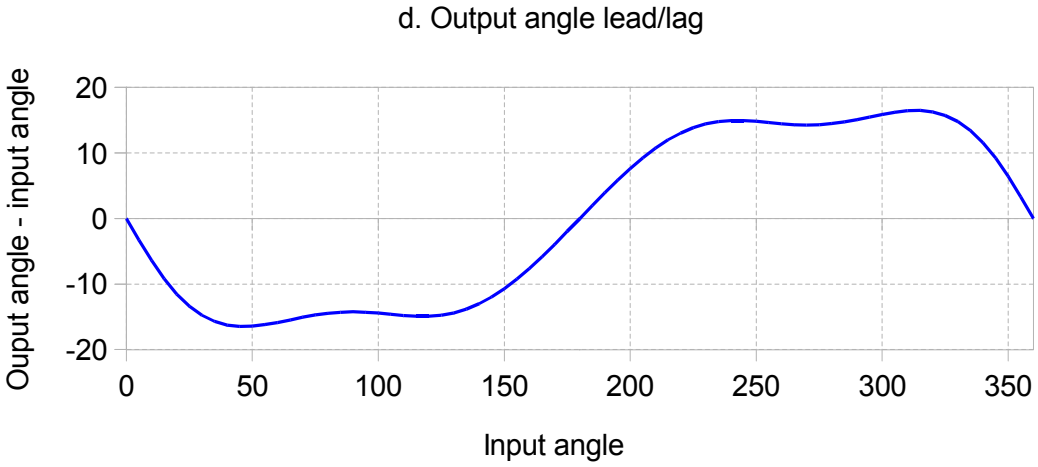
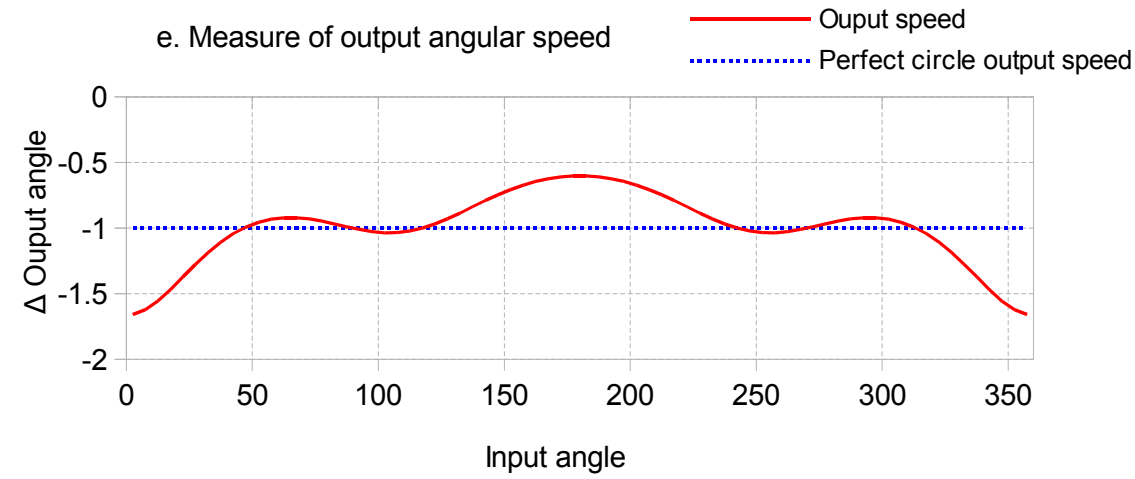


Figure 89(a-c): Mid-point pivot output data





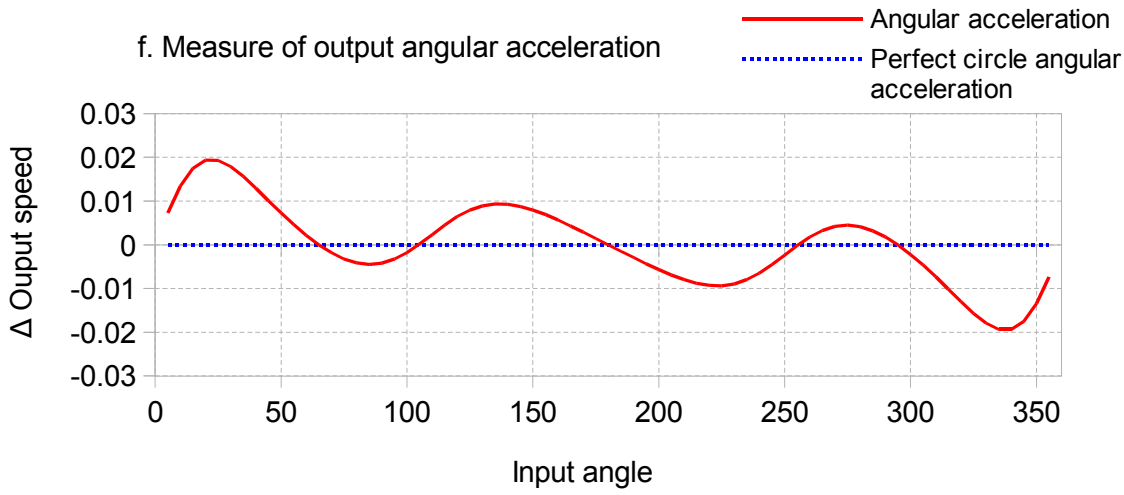


Figure 89(d-f): Mid-point pivot output data

5.2. Design 1: Secondary 3 bar slider-pivot linkage

It followed from the previous discussion that the best location for the secondary 3 bar slider-pivot linkage was also the mid-point of its supporting link. The driver link for the secondary linkage was the follower from the previous stage. Since the input to this stage was not a perfect circle, it was necessary to know what the output of the secondary stage looked like in terms of trajectory, angular speed and acceleration. An additional parameter of interest now was the angle between the supporting links of the primary and secondary 3 bar linkages. This was referred to as angle 3, a labelling reference used in the machine level kinematic modelling of the robot. Also for the graphs that follow the secondary linkage had $q = r = 8$.

The first graph is a 3D illustration of stacked output trajectories of the secondary linkage, for a complete 360º range of the driven input angle, and a complete 360º range for angle 3. It must be noted however that the actual physical range for angle 3 excluded the range between 160º and 200º, as that would cause collisions between proximal and distal links. The height for each trajectory was then set at angle 3. A separate plot illustrates 5 trajectories of interest, and these are shown in figure 90a and b respectively. With respect to the XY axes the maximum and minimum values of the x and y coordinates did not exceed 25% of the input driver link's length on the primary 3 bar linkage. The normalized dimensions for the containing rectangles are shown in figure 90c, and lie within the bounds $1.98 < \Delta y < 2.36$ and $1.99 < \Delta x < 2.21$. Since the curves were normalized these numbers refer to factors of the driver link's length. The normalized mean and RMS errors illustrated next were with respect to a perfect circle output trajectory, with radius equal to the input link length. The average error oscillated close to zero, and the RMS error did not exceed 12%.

The output angle was calculated for the same range of input angle and angle 3, as mentioned previously. The output angle of this stage rotated in the same direction as the input angle due to the double negative, 2 3-bar linkages whose output reversed the direction of its input. Each output angle curve was stacked in 3D with the height set as the value of angle 3 for that trace. Furthermore to get the smooth surface shown in figure 91a on page 157, the angles had to be unwrapped about the [360º, 0º] and the [-360º, 0º] boundaries. If an angle passed 360º, it should not start from 0º again but continued past 360º. Similarly for

negative angles that go below -360º. That ensured continuity and that the derivatives which would follow later for angular speed and acceleration did not spike and give incorrect results. Figure 91b shows a top view of a few selected curves for different values of angle 3. The start angle for each curve however was set to zero so that they could be superimposed. The start angle for each curve is shown in figure 91c. The output angle lead-lag (output angle minus corresponding input angle) with respect to its own start angle was plotted against the input angle and is shown in figures 91d and e, and the maximum lead-lag did not exceed 45º. The curves which serve as measures for angular speed and acceleration are shown in figures 91f to i. Like those curves for the 3 bar linkage this illustrates the non-linear nature of the secondary linkage, as these angular speeds and accelerations were for a constant change of input angle. Of note was the angular speed and acceleration at angle 3 = 90º. In that configuration the output tracked the input perfectly. These output angle speeds and accelerations were the flattest curves that could be achieved, and the supporting link's mid-point was the best choice for pivot position.

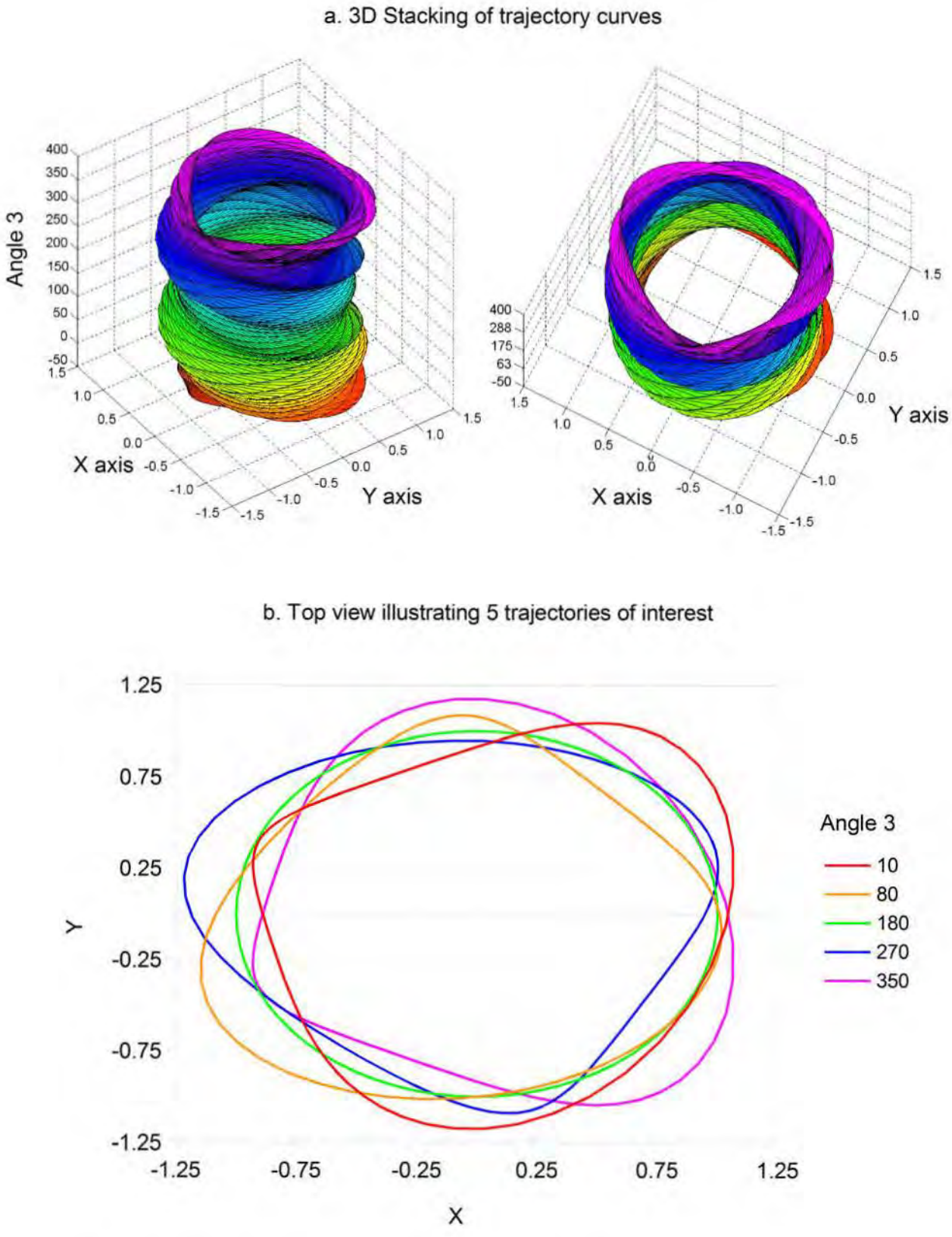


Figure 90(a,b): Output trajectory data for secondary slider-pivot linkage

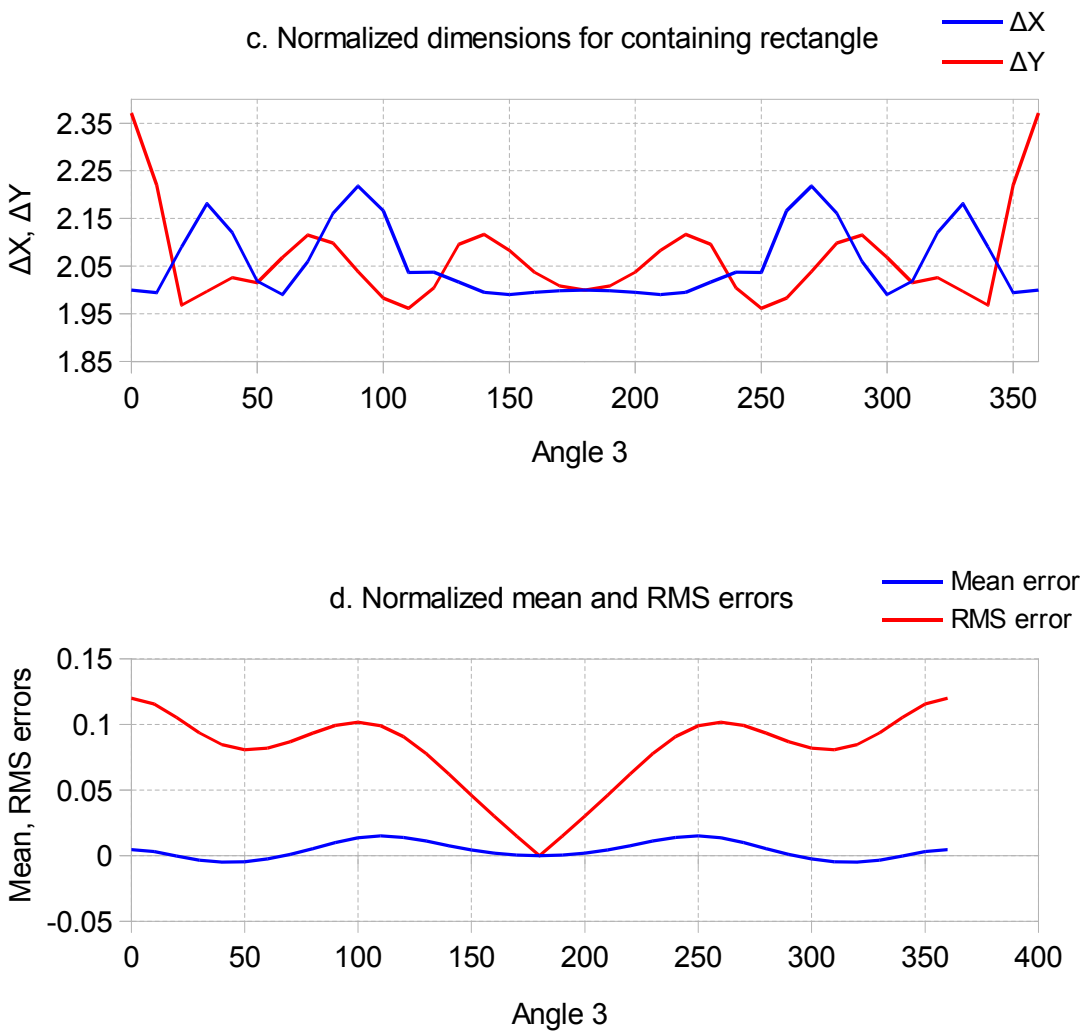
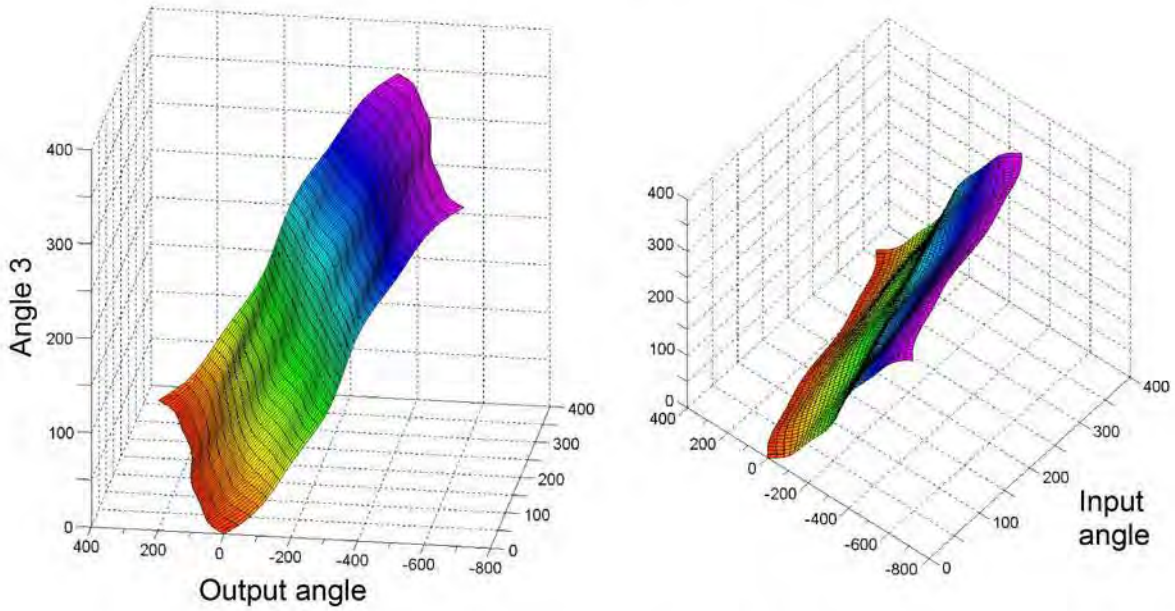


Figure 90(c,b): Output trajectory data for secondary slider-pivot linkage

a. 3D Stacking of output angle



b. Top view illustrating 5 contours of interest

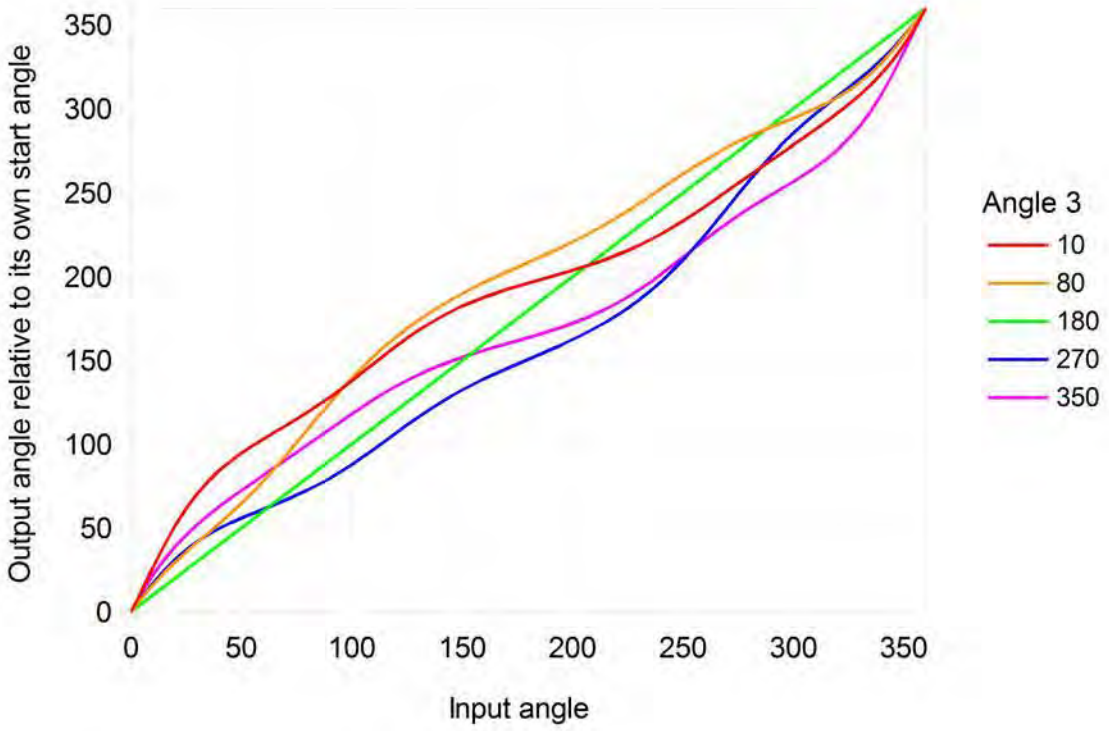


Figure 91(a,b): Output angle data for secondary 3 bar slider-pivot linkage

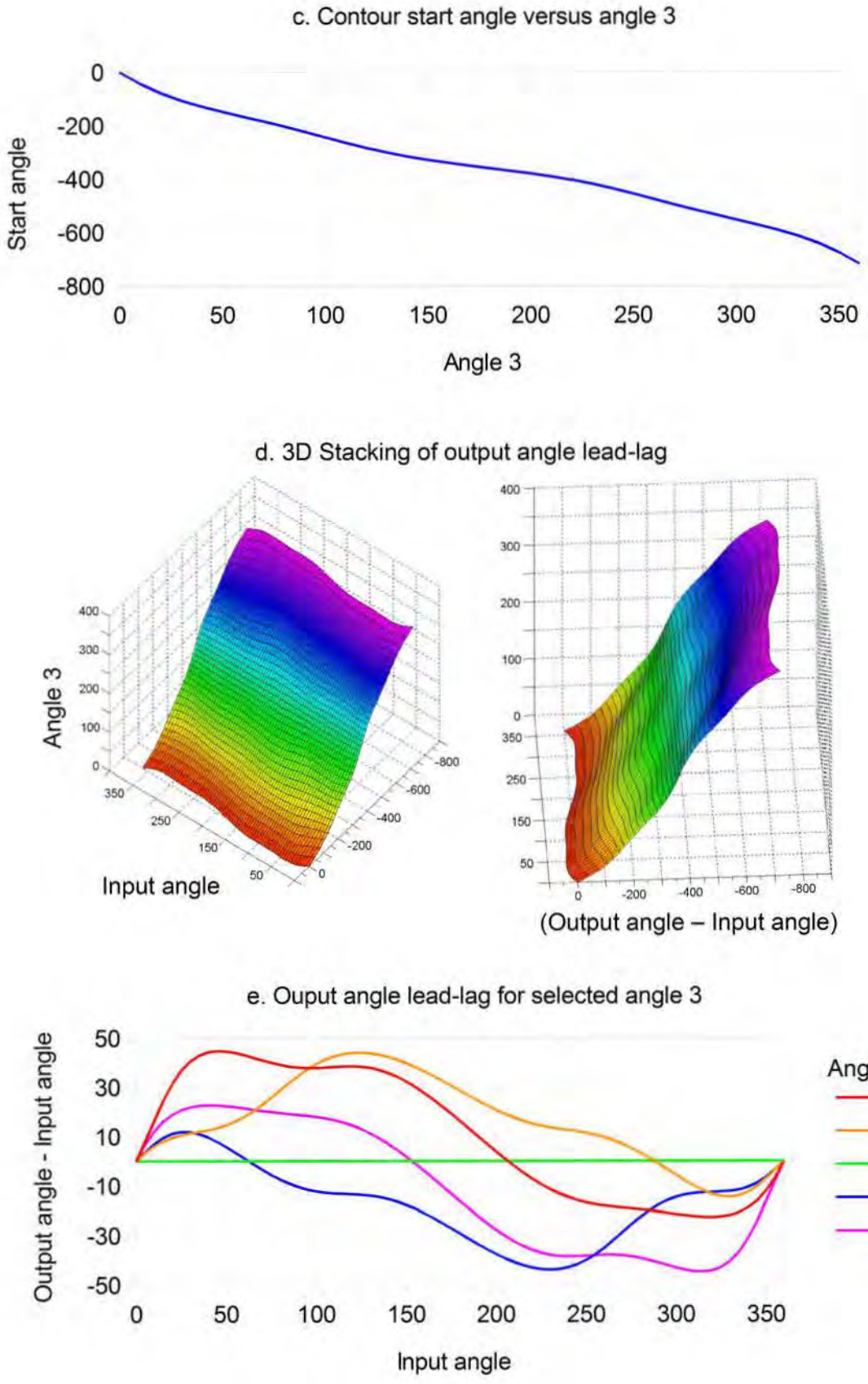


Figure 91(c-e): Output angle data for secondary 3 bar slider-pivot linkage

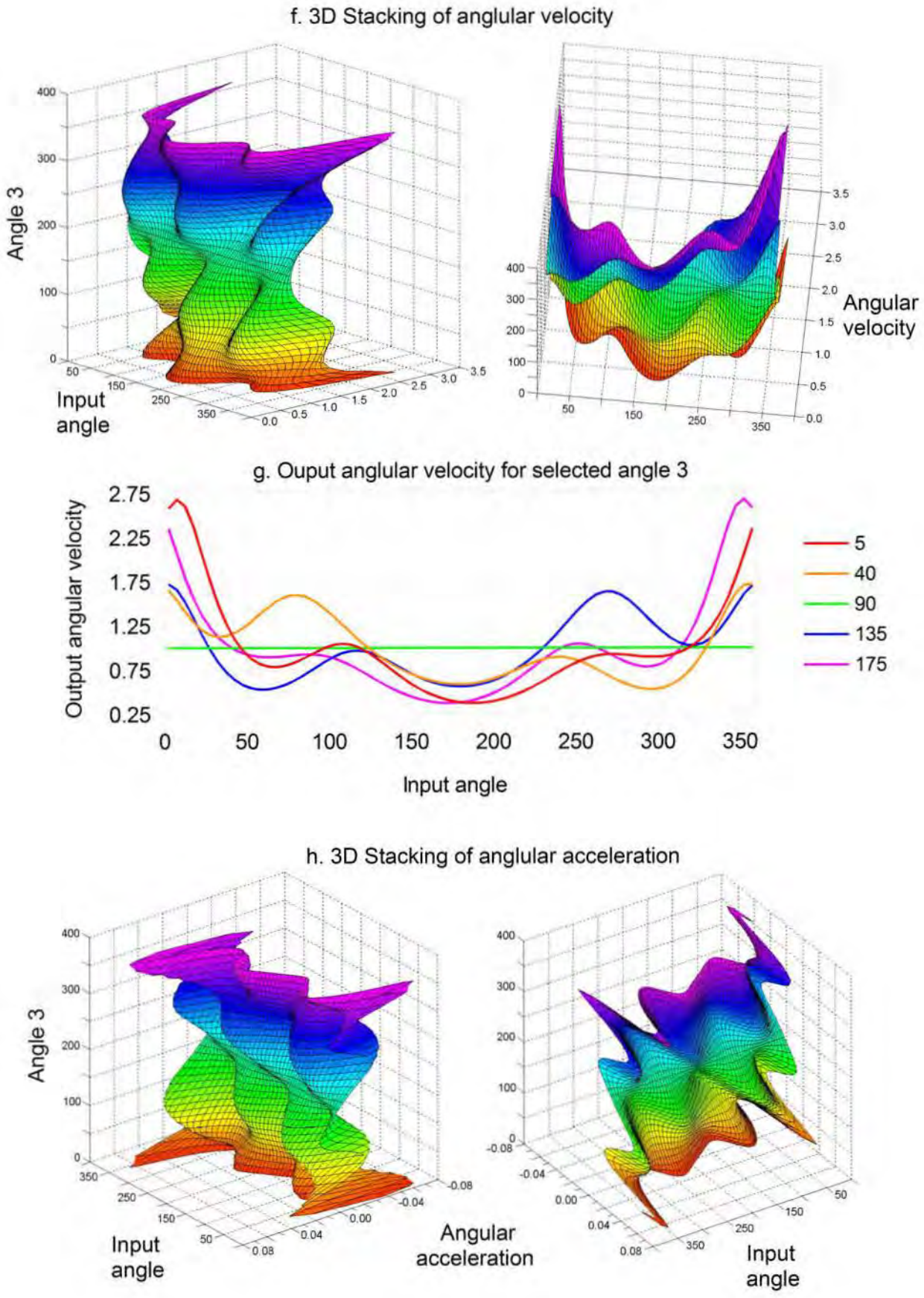


Figure 91(f-h): Output angle data for secondary 3 bar slider-pivot linkage

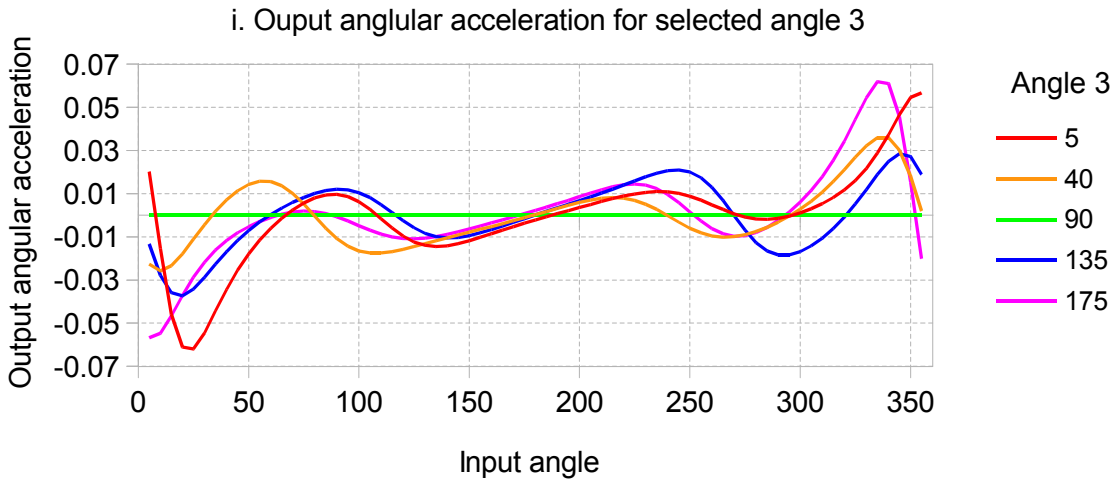


Figure 91(i): Output angle data for secondary 3 bar slider-pivot linkage

5.3. Machine level kinematics simulation

The machine level kinematics simulation could be executed using 3 separate stand alone script files as mentioned in section Error: Reference source not found on page Error: Reference source not found. Figure 92 illustrates the GUI interface for the forward kinematics and figure 93 illustrates the various pre-set paths that the user can select and watch the end-effector trace out. The results for the machine level kinematics were minor but served to point out that both the forward and inverse kinematics scripts verify each other. This was essential for the next part of simulations in which the dynamics of 4 machine models would be compared against each other to determine the most energy efficient. Video simulations can be found in the accompanying CD. A 3D motion simulation was also performed in Solidworks; the time stamped angular position data was exported to Solidworks which then animated a volumetric version of design 2. This video also appears on the CD.

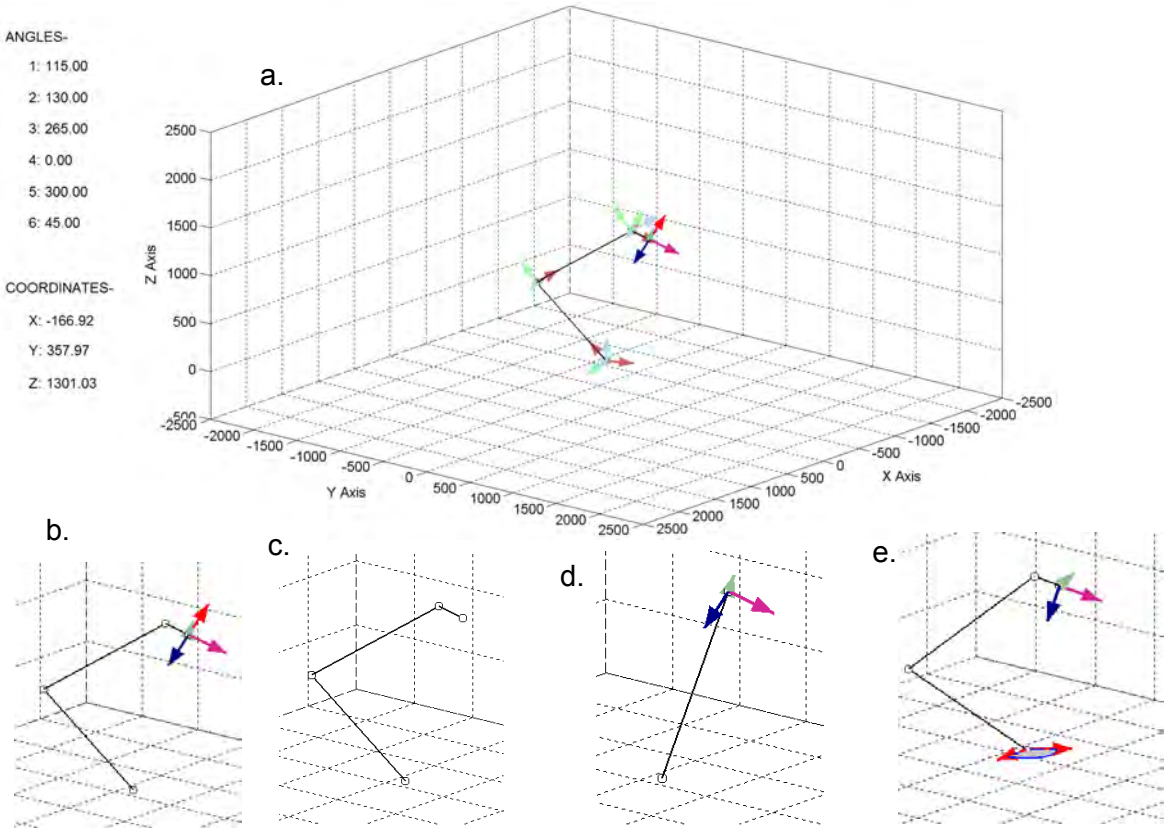


Figure 92: Forward kinematics GUI illustrating multiple views of joint coordinate frames

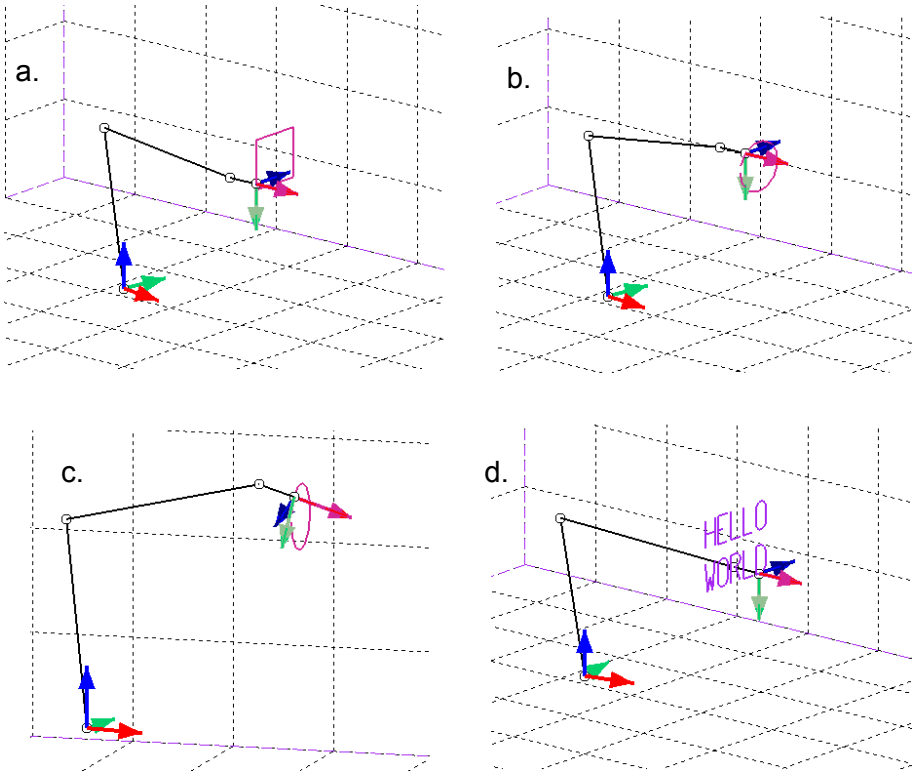


Figure 93: Inverse kinematics implemented on multiple paths

5.4. Machine level dynamics simulation

The purpose of the machine level dynamics simulation was to calculate the joint torques required for motion at a desired speed and acceleration for a given pose, which took into account the motion which lead up to its current pose. As mentioned in the robot dynamics section of the mechanical design chapter, the algorithm used to calculate the manipulator dynamics came from *Craig (2005)* [89]. The algorithm required the following data for implementation, i.e. the joint angles for the given pose, the joint angular velocities and accelerations, the robot model parameters with include inertia tensors, masses, COM locations, velocities and accelerations. The main reason for implementing the dynamics was to determine which of the 4 robot designs was the most energy efficient. As a measure of energy usage the work done at each joint was calculated, and that was given by the multiplication of joint torque and the angular displacement it produced. If there was no angular displacement no work was done even if a joint torque exists. Electrical energy may be consumed in this case but this was ignored. This dynamics simulation only takes into account the joint torques and energy at the 6 major joint axes for each model. Design 1 and design 2 had additional moving parts in their torque transfer linkages, but these could be made of light materials and their speeds could also be set quite low if high accuracy low reduction gearboxes were used at the respective joint axes. Hence their energy consumption should be low in comparison to the energy required at each of the joints they control. To account for this however an immediate 15% penalty was placed on each of the hybrid models, meaning that they would have to consume at least 15% less energy than the serial model to be on par with it, only on an energy front. For these designs to be an actual improvement and present a competitive alternative to the serial robot they must consume at least 50% less energy when taking into account manufacturing cost, set up, maintenance and capital outlay for factory overhaul, etc. Furthermore the only active joints that are of interest were the first 3 which move the most amount of mass to position the end-effector. The last 3 which control the orientation of the end-effector was ignored, since the wrist was modelled in exactly the same way for each of the robot models. The joint torque and hence energy used at those corresponding axes were the same. For the hybrid designs the actual

energy used would be higher as the torque transfer links had be accounted for, even though they would be minimal. So at the outset if one only considered the wrist motors, the serial robot was best in terms of energy consumption since it had fewer torque transfer links. If however the hybrid designs saved more than 50% of the energy when considering the first 3 active joints then they would be better overall. The simulations ignored the strength of links and was purely a view along energy lines, friction was also ignored.

There were a number of parameters that could vary, and the amount of data generated if all were taken into account would be staggering. In this thesis however one parametric configuration was used for each of the hybrid designs, and they were such that the corresponding proximal and distal links in all models had the same mass (which included the torque transfer links at each stage). This meant that each corresponding joint in each of the hybrid machine models moved the same mass. The difference in torque and energy usage between the models would then come from link mass distribution and the geometry of the links. The serial robot model however had additional mass from its wrist, elbow and shoulder motors which it had to move, and for that 2 parametric models were used. In the first light weight motors were used which would make the serial robot similar to the hybrid designs. In the second heavy motors were used which pushed the mass and mass distribution for each robot pose further away from those in the hybrid designs. Heavy motors in this sense was classified as motors that were 35% the mass of all the links that it moved, with the exception of the wrist motor block which was set at 35% of the distal link's mass. The elbow motor (motor 3) was then 35% of the combined mass of the distal link and the wrist motors. The shoulder motor (motor 2) was 35% of the combined mass of the proximal link, the distal link and the wrist motors. These 2 parametric serial models should then be the limits at which the serial robot could exist when considering motor mass only. The motor geometry was modelled as a block with varying edge length, whose density was estimated at 15% steel, 40% ferrite, 40% copper and 5% empty space, i.e. 6709.5 $kg \cdot m^{-3}$. The density was needed to determine motor volume and the side lengths or edges were calculated from parametric relations between the edges. Those relations as well as others which made up the geometric parameters of the models can be seen in table 7 and were with respect to the figures used in

section 3.4.3.4 simplified dynamic models. The table also includes some relevant masses, i.e. motor masses, torque transfer link masses and structural link masses (m_{l1}, m_{l2} and m_{l3} for the movable base, proximal and distal links respectively). The motor mass totalled 9.28kg for the light version of the serial robot, which was an additional 8.2% more than that for designs 1 and 2; and 8.15% more than the Pozzi model (which carried 0.42kg, or 0.37% more mass than designs 1 and 2 due to its elbow gears). For the heavy version of the serial robot the motor mass totalled 80.63kg, which was 71% heavier than designs 1 and 2 (70.8% more than Pozzi). Tables 8 to 12 list the numerical values of the dynamic parameters for each model, i.e. the coordinate frame origins ${}^{i}P_{i+1}$, the link COMs ${}^{i+1}P_{C(i+1)}$, the link masses m_{i+1} and the inertia tensors at the COMs ${}^{C(i+1)}I_{i+1}$. It must be noted however that for design 1, the tensors and COMs were for the initial pose of the robot. In that design the torque transfer links could change position relative to the their supporting links, so the tensors and COMs change and must be recalculated for each pose.

Table 7: Geometric parameters for dynamic models including some masses

Robot model	Geometric parameters and some masses
Common to all models	$L_1=1\text{m};\ L_2=1.066\text{m};\ L_3=0.181\text{m};$ $r_{bg6}=0.15\cdot L_1=0.15\text{m};\ l_4=\frac{r_{bg6}}{5}=0.03\text{m};$ $m_{l1}=16.65\text{kg};\ m_{l2}=54.80\text{kg};\ m_{l3}=42.07\text{kg}$
Design 1	$a_1=0.034\text{m};\ a_2=a_1;\ w_1=2\cdot r_{l1};\ b_1=0.5\cdot w_1;\ b_2=2\cdot a_1;\ c_1=2\cdot a_1;$ $a_3=0.03\text{m};\ a_4=a_3;\ w_2=w_1-a_1-a_3;\ b_3=2\cdot a_3;\ b_4=0.5\cdot w_2;\ c_2=b_3;$ $tmp=w_2-a_3;\ d_1=0.75\cdot tmp;\ d_2=0.25\cdot tmp;\ d_3=-0.25\cdot tmp;$ $d_4=-0.75\cdot tmp;\ d_5=0.25\cdot tmp;\ d_6=-0.25\cdot tmp;\ d_7=-0.75\cdot tmp;$ $r_1=0.25\cdot a_1;\ m_4=1.84\text{kg};\ m_6=1.96\text{kg}$
Design 2	$w_1=2\cdot r_{l1};\ a_1=0.035\text{m};\ b_1=2\cdot a_1;\ c_1=b_1;$ $a_2=0.03\text{m};\ w_2=w_1-a_1-a_2;\ b_2=2\cdot a_2;\ c_2=b_2;$ $d_1=0.75\cdot w_1;\ d_2=0.25\cdot w_1;\ d_3=-0.25\cdot w_1;\ d_4=-0.75\cdot w_1;$ $d_5=0.25\cdot w_2;\ d_6=0;\ d_7=-0.25\cdot w_2;$ $r_1=0.25\cdot a_1;\ m_3=1.84\text{kg};\ m_5=1.96\text{kg}$
Serial robot (light motors)	$r_{l1}=r_{bg6};\ h_{l1}=l_4;\ d_1=r_{bg6};\ d_2=d_1;$ $m_{m2}=3.62\text{kg};\ l_{m2}=0.15\text{m};\ w_{m2}=0.4\cdot l_{m2};$ $m_{m3}=1.86\text{kg};\ l_{m3}=0.12\text{m};\ w_{m3}=0.4\cdot l_{m3};$ $m_{m4}=3.8\text{kg};\ l_{m4}=0.096\text{m};\ w_{m4}=1.6\cdot l_{m4};\ b_{m4}=0.4\cdot l_{m4};$ $d_3=0.5\cdot l_{m4};\ d_4=0.5\cdot l_{m3};\ r_{o2}=0.05\text{m};\ r_{i2}=0.017\text{m};$ $r_{o3}=0.043\text{m};\ r_{i3}=0.017\text{m};$
Serial robot (heavy motors)	$r_{l1}=r_{bg6};\ h_{l1}=l_4;\ d_1=r_{bg6};\ d_2=d_1;$ $m_{m2}=46.02\text{kg};\ l_{m2}=0.35\text{m};\ w_{m2}=0.4\cdot l_{m2};$ $m_{m3}=19.88\text{kg};\ l_{m3}=0.265\text{m};\ w_{m3}=0.4\cdot l_{m3};$ $m_{m4}=14.73\text{kg};\ l_{m4}=0.15\text{m};\ w_{m4}=1.6\cdot l_{m4};\ b_{m4}=0.4\cdot l_{m4};$ $d_3=0.5\cdot l_{m4};\ d_4=0.5\cdot l_{m3};\ r_{o2}=0.05\text{m};\ r_{i2}=0.017\text{m};$ $r_{o3}=0.043\text{m};\ r_{i3}=0.017\text{m};$
Pozzi	$d_1=0.15\cdot L_1=0.15\text{m};\ d_2=\frac{2}{3}d_1=0.1\text{m};$ $d_3=\frac{1}{3}d_1=0.05\text{m};\ m_{eg}=0.42\text{kg}$ $r_p=\begin{bmatrix}0.017 & 0.023 & 0.03 & 0.037 & 0.043 & 0.05\end{bmatrix};$ $r_{eg}=\begin{bmatrix}0 & 0.007 & 0.013 & 0.02 & 0.027 & 0.05\end{bmatrix};$ $r_d=\begin{bmatrix}0.017 & 0.023 & 0.03 & 0.037 & 0.043\end{bmatrix};$

Table 8: Dynamics parameters for Design 1 (initial configuration)

i	${}^{i}\mathbf{P}_{i+1}\,(\mathrm{m})$	${}^{i+1}\mathbf{P}_{C(i+1)}\,(\mathrm{m})$	$\mathbf{m}_{i+1}\,(\mathrm{kg})$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	16.65	$\begin{bmatrix}0.095 & 0 & 0\\0 & 0.095 & 0\\0 & 0 & 0.187\end{bmatrix}$
1	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.508\\-0.0002\\0\end{bmatrix}$	54.80	$\begin{bmatrix}0.926 & -0.01 & 0\\-0.01 & 4.688 & 0\\0 & 0 & 3.802\end{bmatrix}$
2	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.527\\-0.0006\\0\end{bmatrix}$	42.07	$\begin{bmatrix}0.456 & -0.013 & 0\\-0.013 & 3.894 & 0\\0 & 0 & 3.465\end{bmatrix}$
3	$\begin{bmatrix}1.066\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
4	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
5	$\begin{bmatrix}0.181\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\-0.091\end{bmatrix}$	9.18	$\begin{bmatrix}0.030 & 0 & 0\\0 & 0.030 & 0\\0 & 0 & 0.009\end{bmatrix}$

Table 9: Dynamics parameters for Design 2

i	${}^{i}\mathbf{P}_{i+1}\,(\mathrm{m})$	${}^{i+1}\mathbf{P}_{C(i+1)}\,(\mathrm{m})$	$\mathbf{m}_{i+1}\,(\mathrm{kg})$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	16.65	$\begin{bmatrix}0.095 & 0 & 0\\0 & 0.095 & 0\\0 & 0 & 0.187\end{bmatrix}$
1	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.5\\0\\0\end{bmatrix}$	54.80	$\begin{bmatrix}0.958 & 0 & 0\\0 & 4.681 & 0\\0 & 0 & 3.761\end{bmatrix}$
2	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.533\\0\\0\end{bmatrix}$	42.07	$\begin{bmatrix}0.468 & 0 & 0\\0 & 3.892 & 0\\0 & 0 & 3.446\end{bmatrix}$
3	$\begin{bmatrix}1.066\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
4	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
5	$\begin{bmatrix}0.181\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\-0.091\end{bmatrix}$	9.18	$\begin{bmatrix}0.030 & 0 & 0\\0 & 0.030 & 0\\0 & 0 & 0.009\end{bmatrix}$

Table 10: Dynamics parameters for serial robot model (light motors)

i	${}^{i}\mathbf{P}_{i+1}\,(\mathrm{m})$	${}^{i+1}\mathbf{P}_{C(i+1)}\,(\mathrm{m})$	$\mathbf{m}_{i+1}\,(\mathrm{kg})$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\-0.027\\0\end{bmatrix}$	20.27	$\begin{bmatrix}0.17 & 0 & 0\\0 & 0.056 & 0\\0 & 0 & 0.262\end{bmatrix}$
1	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.516\\0\\-0.143\end{bmatrix}$	56.66	$\begin{bmatrix}0.158 & 0 & -0.188\\0 & 5.135 & 0\\-0.188 & 0 & 5.054\end{bmatrix}$
2	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.485\\0\\0\end{bmatrix}$	45.87	$\begin{bmatrix}0.053 & 0 & 0\\0 & 5.196 & 0\\0 & 0 & 5.203\end{bmatrix}$
3	$\begin{bmatrix}1.066\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
4	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
5	$\begin{bmatrix}0.181\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\-0.091\end{bmatrix}$	9.18	$\begin{bmatrix}0.030 & 0 & 0\\0 & 0.030 & 0\\0 & 0 & 0.009\end{bmatrix}$

Table 11: Dynamics parameters for serial robot model (heavy motors)

i	${}^{i}\mathbf{P}_{i+1}\,(\mathrm{m})$	${}^{i+1}\mathbf{P}_{C(i+1)}\,(\mathrm{m})$	$\mathbf{m}_{i+1}\,(\mathrm{kg})$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\-0.110\\0\end{bmatrix}$	62.67	$\begin{bmatrix}0.632 & 0 & 0\\0 & 0.405 & 0\\0 & 0 & 0.724\end{bmatrix}$
1	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.633\\0\\-0.094\end{bmatrix}$	74.68	$\begin{bmatrix}0.784 & 0 & -1.532\\0 & 8.960 & 0\\-1.532 & 0 & 8.334\end{bmatrix}$
2	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.383\\0\\0\end{bmatrix}$	56.80	$\begin{bmatrix}0.165 & 0 & 0\\0 & 7.718 & 0\\0 & 0 & 7.823\end{bmatrix}$
3	$\begin{bmatrix}1.066\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
4	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
5	$\begin{bmatrix}0.181\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\-0.091\end{bmatrix}$	9.18	$\begin{bmatrix}0.030 & 0 & 0\\0 & 0.030 & 0\\0 & 0 & 0.009\end{bmatrix}$

Table 12: Dynamics parameters for Pozzi robot model

i	${}^{i}\mathbf{P}_{i+1}\,(\mathrm{m})$	${}^{i+1}\mathbf{P}_{C(i+1)}\,(\mathrm{m})$	$\mathbf{m}_{i+1}\,(\mathrm{kg})$	${}^{C(i+1)}\mathbf{I}_{i+1}$
0	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	16.65	$\begin{bmatrix}0.095 & 0 & 0\\0 & 0.095 & 0\\0 & 0 & 0.187\end{bmatrix}$
1	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.504\\0\\0.150\end{bmatrix}$	55.22	$\begin{bmatrix}0.077 & 0 & 0.010\\0 & 4.711 & 0\\0.010 & 0 & 4.710\end{bmatrix}$
2	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0.533\\0\\0.05\end{bmatrix}$	42.07	$\begin{bmatrix}0.045 & 0 & 0\\0 & 4.009 & 0\\0 & 0 & 4.009\end{bmatrix}$
3	$\begin{bmatrix}1.066\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
4	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	0	$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$
5	$\begin{bmatrix}0.181\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\-0.091\end{bmatrix}$	9.18	$\begin{bmatrix}0.030 & 0 & 0\\0 & 0.030 & 0\\0 & 0 & 0.009\end{bmatrix}$

The simulations were carried at 2 speeds, the first being very slow so that the dynamic response of machine was close to its static state. This meant that the torques required for motion were slightly larger than that required to maintain a static pose. The second speed was excessive and may not be physically achievable but was necessary to illustrate how the robot models compared with each other when the dynamic response was dominant.

For each path the end-effector was programmed to follow a simple velocity motion profile, along one of the X, Y or Z reference frame axes. It started at the beginning of the path with 0 velocity, and accelerated with constant positive acceleration (so that the velocity curve was a straight line) to a maximum. Thereafter it ramped down to $0\mathrm{m}\cdot\mathrm{s}^{-1}$ with a constant negative acceleration (so that the velocity curve for the second half was a straight line). This is shown in figure 94. In real world applications there must be a finite duration between changes in acceleration, and so the acceleration curve should be an S-curve instead of a straight edged step curve. However the derivative of acceleration was never used and so that bit of real world detail was ignored.

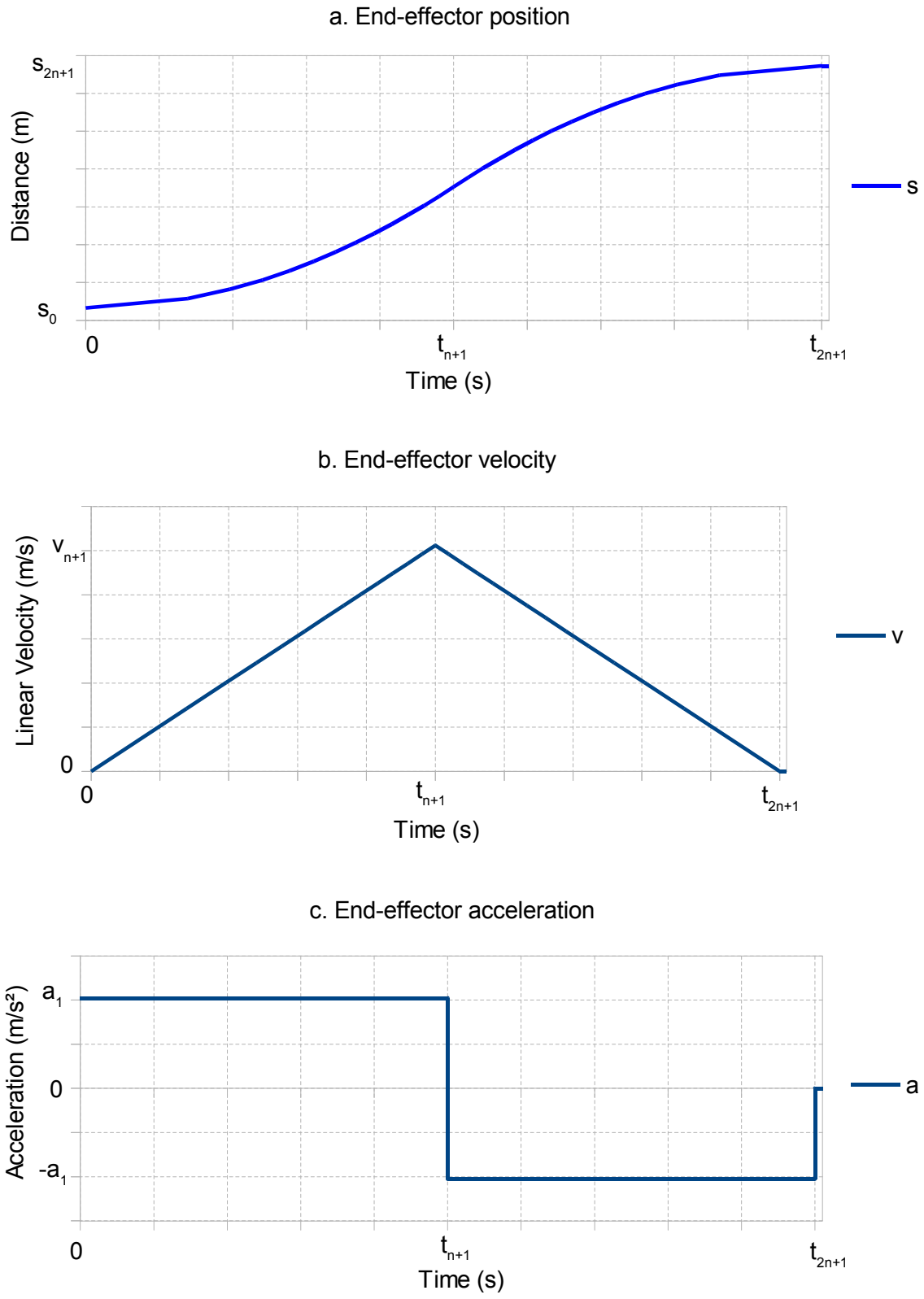


Figure 94: End-effector data for straight line path on selected axis

The time vector was the most important bit of information required for the dynamics and was calculated as follows. The path selected was broken up into a number of smaller straight line paths according to the resolution set, for instance 5mm. The number of steps was calculated by dividing the path length (path length was total distance travelled not the total displacement which could be 0 if a circular path was designed) by the resolution (call this number 2n). There were thus 2n+1 equally spaced points on the path from start to finish. The average velocity (v_{ave}) was found by taking the total distance (S) and dividing it by the total time duration set (T). The mid-path velocity was twice that average value ($2 \cdot v_{ave}$) due to the linear profile being designed. The velocities at the start (v_1) and end (v_{2n+1}) was $0 m \cdot s^{-1}$. The first half acceleration (a_1) was then the difference between the mid-path velocity and the start velocity ($0 m \cdot s^{-1}$) divided by half the total time $\left(\frac{2 \cdot v_{ave}}{0.5 \cdot T} \right)$. The velocity at each point in the first half was proportional to the distance travelled, and was given by $v_i = s_i \frac{2 \cdot v_{ave}}{0.5 \cdot S}$. The time at each path point was then calculated using $t_i = \frac{v_i - v_1}{a_1}$. The time for each point in the second half was calculated in a similar manner, i.e. $t_i = \frac{v_i - v_{ave}}{-a_1} + 0.5 \cdot T$, where $-a_1$ was the acceleration and v_i $\left(v_i = (s_i - S) \frac{2 \cdot v_{ave}}{0.5 \cdot S} \right)$ the velocities in the second half.

The joint angles were calculated for each pose in the selected path, and even though the end-effector's position changed with uniform distance in a predictable fashion the change in joint angles from pose to pose was not straight forward and not uniform. Hence the angular speeds would vary, and so too would the joint angle accelerations. The average angular velocity between 2 consecutive points in the path was given by the change in joint angle divided by the time increment, i.e. $\dot{\theta}_{(i+1)ave} = \frac{\theta_{i+1} - \theta_i}{\Delta t}$. It was also given by the average of the sum of the 2 velocities at those points, i.e. $\dot{\theta}_{(i+1)ave} = \frac{\dot{\theta}_{i+1} + \dot{\theta}_i}{2}$. Using these 2 pieces of information together with the fact that the robot started from rest (0 angular velocity at initial pose, $i = 1$, $\dot{\theta}_1 = 0$), the angular velocity at every other pose was calculated, which started at

$\dot{\theta}_2$ and moved sequentially to the end using the formula $\dot{\theta}_{i+1} = 2\cdot\dot{\theta}_{(i+1)ave} - \dot{\theta}_i = 2\cdot\frac{\theta_{i+1} - \theta_i}{\Delta t} - \dot{\theta}_i$. Each pair of consecutive angular velocities was connected with a straight line, this implied the acceleration between those 2 path points was constant and represented by a flat line with 0 gradient. The graphical visualization of the angular acceleration was therefore a step curve. The acceleration was calculated for values up to $2n$ using the formula $\ddot{\theta}_i = \frac{\dot{\theta}_{i+1} - \dot{\theta}_i}{\Delta t}$. The last value in the acceleration vector was 0 since the robot had to stop once it reached the last point in the path, so $\ddot{\theta}_{2n+1} = 0$. The linear velocities and accelerations of the COMs and origins of the joint coordinate frames were calculated in a similar way.

The equations used in the dynamics algorithm were mentioned in the robot dynamics section and will not be repeated here. The last important calculation was the measure of energy used, and here it was the angular work done. That was calculated by torque multiplied by angular displacement, i.e. $w_i = \tau_i\cdot(\theta_{i+1} - \theta_i)$ for $i=1$ to $2n$. $w_{2n+1} = 0$ Since for $i \geq 2n+1$, $\theta_i = \theta_{2n+1}$ once the last position was reached the joint angles were held. The total energy consumed up to each point in the path was the integral of w_i from the start position to that particular point, provided that the torque and angular data were continuous. Since the data was discrete, an accumulated sum was used instead.

A robot's work path could have a large number of segments (linear or curved) connected in an infinite number of ways. It was decided to compare the dynamics over the simplest comprising elements of a complex path, these were straight line segments and circular curves. The straight line paths were vertical, horizontal and radial (radial lines were horizontal but intersected with the $\hat{z}_1$ axis, extended if necessary). Any other line path could be considered as a linear combination of the above paths, i.e. a diagonal line would be composed of a horizontal line with a vertical or a radial line with a vertical line, etc.

For each straight line path type (radial, horizontal or vertical) the dynamics were tested over a range of lines, by changing the centre position but keeping the length and line type the same. The paths started close to the base origin and moved further away by increasing one

coordinate in that line type's centre location. This is captured concisely in table 13. SCE represents the start, middle and end coordinates of each straight line path, with the x, y, z coordinates listed in column form. Only the total energy consumed at the end of each path was taken, which was plotted against the coordinate that was advanced (the y value for the vertical and horizontal paths; and the z value for the radial paths). Δ Indicates the total change in the variable coordinate for the path centre position (as well as the path variable, which gives path length) and δ indicates the increment from one value to the next. Each model was tested at 2 different speeds (slow and then very fast), indicated by the total time (5s and 0.25s) duration for path motion in table 13. The faster linear speed was set at 20 times the slower one. When the linear path completed in 5s (low speed), the end-effector reached a maximum speed of $0.43\,m\cdot s^{-1}$ at 2.5s, with a gradient (acceleration) of $0.17\,m\cdot s^{-2}$ for the first half and $-0.17\,m\cdot s^{-2}$ for the second half. For high end-effector speed when the path completed in 0.25s, it reached a maximum speed of $8.53\,m\cdot s^{-1}$, with an acceleration of $68.23\,m\cdot s^{-2}$ for the first half and $-68.23\,m\cdot s^{-2}$ for the second half. These values were with respect to the graphs in figure 94 on page 169.

Table 13: Multi-straight line path specifications

Path	SCE (m)	Variable coordinate (m)	Path length (m)	Time (s)
Multi-vertical lines	$\begin{bmatrix} 0 & 0 & 0 \\ y_i & y_i & y_i \\ 0.213 & 0.746 & 1.28 \end{bmatrix}$	$0.32 \le y_i \le 1.706$ $\Delta y = 1.386$ $\delta y = 0.035$	$\Delta z = 1.066$	5 0.25
Multi-horizontal lines	$\begin{bmatrix} -0.533 & 0 & 0.533 \\ y_i & y_i & y_i \\ 0.746 & 0.746 & 0.746 \end{bmatrix}$	$0.32 \le y_i \le 1.706$ $\Delta y = 1.386$ $\delta y = 0.035$	$\Delta x = 1.066$	5 0.25
Multi-radial lines	$\begin{bmatrix} 0 & 0 & 0 \\ 0.213 & 0.746 & 1.28 \\ z_i & z_i & z_i \end{bmatrix}$	$0.533 \le z_i \le 1.599$ $\Delta z = 1.066$ $\delta z = 0.027$	$\Delta y = 1.066$	5 0.25

For the circular paths it was decided to isolate each of the first 3 axes responsible for wrist centre translation, i.e. only 1 of those 3 angles would change while the remaining 5 were kept constant. By doing this the wrist centre moved on a circular arc about the actuated axis. Additionally this allowed a large change (difference between the start and end angular

positions) in joint angle, unlike in the straight line paths. Each circular path, for each axis had a common start angle, which was given relative to that joint's coordinate frame axis. The end angle was varied between selected minimum and maximum values, with the increment given by $\Delta\Theta$. The increment from the start angle to the end angle was given by $\delta\theta$. The circular path specifications are given in table 14. In the table variable m sets the number of paths in the angular range for a joint, and variable n sets the angular resolution for a particular path. Again the dynamics for each model was tested at 2 different speeds (slow then fast), with the faster angular speed set at 20 times the slower one, in keeping with the ratios from the straight line paths.

Table 14: Circular path specifications

Angle varied	**Configuration** $[\theta_1;\theta_2;\theta_3]$; $([\theta_4;\theta_5;\theta_6]=[0°;0°;0°])$	**Range** $2 \le k \le m;\ m \ge 2$ $1 \le i \le n;\ n \ge 2$	**Time (s)**
Angle 1	$[\theta_{1(i)};\ 0°;\ 0°]$	$\Delta\Theta = \frac{180°}{m-1}$ $\Delta\Theta \le \theta_{1(k)} \le 180°$ $\theta_{1(k+1)} = \theta_{1(k)} + \Delta\Theta$ $\theta_{1(n)}=\theta_{1(k)};\ \theta_{1(1)}=0°$ $\delta\theta = \frac{\theta_{1(k)}}{n-1}$ $0° \le \theta_{1(i)} \le \theta_{1(n)}$ $\theta_{1(i+1)} = \theta_{1(i)} + \delta\theta$	40
			2
	$[\theta_{1(i)};\ 90°;\ 0°]$		40
			2
Angle 2	$[0°;\ \theta_{2(i)};\ 0°]$	$\Delta\Theta = \frac{90°}{m-1}$ $\Delta\Theta \le \theta_{2(k)} \le 90°$ $\theta_{2(k+1)} = \theta_{2(k)} + \Delta\Theta$ $\theta_{2(n)}=\theta_{2(k)};\ \theta_{2(1)}=0°$ $\delta\theta = \frac{\theta_{2(k)}}{n-1}$ $0° \le \theta_{2(i)} \le \theta_{2(n)}$ $\theta_{2(i+1)} = \theta_{2(i)} + \delta\theta$	40
			2
	$[0°;\ \theta_{2(i)};\ 170°]$		40
			2
Angle 3	$[0°;\ 0°;\ \theta_{3(i)}]$	$\Delta\Theta = \frac{180°}{m-1}$ $(-90°+\Delta\Theta) \le \theta_{3(k)} \le 90°$ $\theta_{3(k+1)} = \theta_{3(k)} + \Delta\Theta$ $\theta_{3(n)}=\theta_{3(k)};\ \theta_{3(1)}=-90°$ $\delta\theta = \frac{\theta_{3(k)}}{n-1}$ $-90° \le \theta_{3(i)} \le \theta_{3(n)}$ $\theta_{3(i+1)} = \theta_{3(i)} + \delta\theta$	40
			2

There were 2 limits for arm mass distribution (or arm configuration) which gave the best and worst case joint torque (and subsequently joint work) for axes 1 and 2. For axis 1 the worst case situation was when each COM was at the furthest location it could occupy (with respect to that axis), which forced the largest mass moment about axis 1. This occurred when the arm pointed radially out, with links 1 and 2 parallel and perfectly horizontal, and the wrist origin was on the workspace boundary. That was a singularity position but it didn't matter as only angle 1 would change (similarly for the next configuration). The best case mass distribution for axis 1 was when each COM was at the closest location it could occupy, and subsequently the mass moment about axis 1 was the lowest. Here the arm pointed vertically up, with the wrist origin on the workspace boundary. The best case mass distribution for axis 2 would be when the distal arm folds over the proximal arm (making them parallel but with the wrist origin at the closet distance it could be from the reference frame origin), that pose however would force link collision. Angle 3 was thus set such that the angle between the proximal and distal arm was a minimum without link interference (that angle was 10º, but angle 3 was set to 170º due to the way the kinematic model coordinate frames were defined). The worst case for axis 2 was when the proximal and distal links were parallel and the wrist origin was on the workspace boundary, which forced the largest mass moment about joint 2, for any value of angle 2.

5.5. Multi-straight line path simulation results

These simulation results are illustrated in figures 95 to 97 which depict a comparison of total energy usage only (total energy consumed per model at the end of each path). Complete data tables and additional graphs can be seen in appendix C from page C2 to C37, in the accompanying CD. The graphs for each hybrid design were relative to the serial robot, and were displayed as a percentage thereof for both cases. For each path, the comparison against the light motor (LM) version of the serial robot was shown first, followed by the heavy motor version (HM). The path data was displayed in table 13 on page 172.

5.5.1. Multi-vertical line paths

For the LM, low speed case shown in figure 95a, design 1 had the best energy usage in comparison to all hybrid models. In the region between 775mm and 1200mm all hybrid designs performed worse than the light motor version of the serial robot. More energy was saved when the vertical path was closer to the $\hat{z}_1$ axis, with a minimum of around 96.75%, a maximum saving of 3.25%. When the dynamic response was made dominant, all hybrid designs had better energy usage throughout the y-range, with designs 1 and 2 very close. The further away the vertical line was from the $\hat{z}_1$ axis the closer the energy consumptions got to the serial robot. Also in this case there was a region where the energy usage was most optimal for each hybrid robot, and that was between 600mm and 800mm. The comparison with the heavy motor serial robot at low and high speeds shown in 95c and d, were similar to the comparisons in figures 95a and b respectively, but had more drastic percentages and shifted regions. For the low speed case, the graphs were virtually indistinguishable. The hybrid designs' consumption dropped to a minimum of 82.5% of the serial robot's consumption with small y in 95c. The hybrid designs also reached a maximum of 109%. At high dynamics the hybrid designs performed better throughout the range. The Pozzi model was the worst among the 3, and the best percentage reached by designs 1 and 2 was 75%.

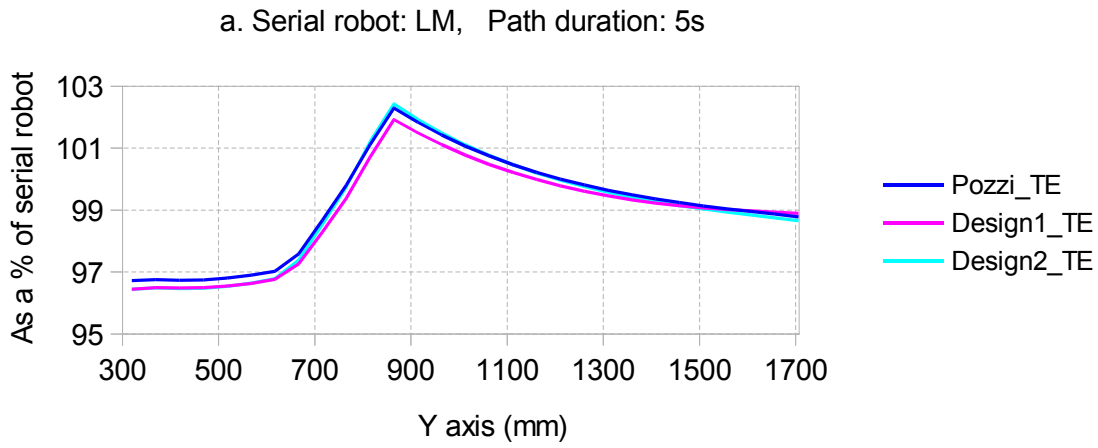
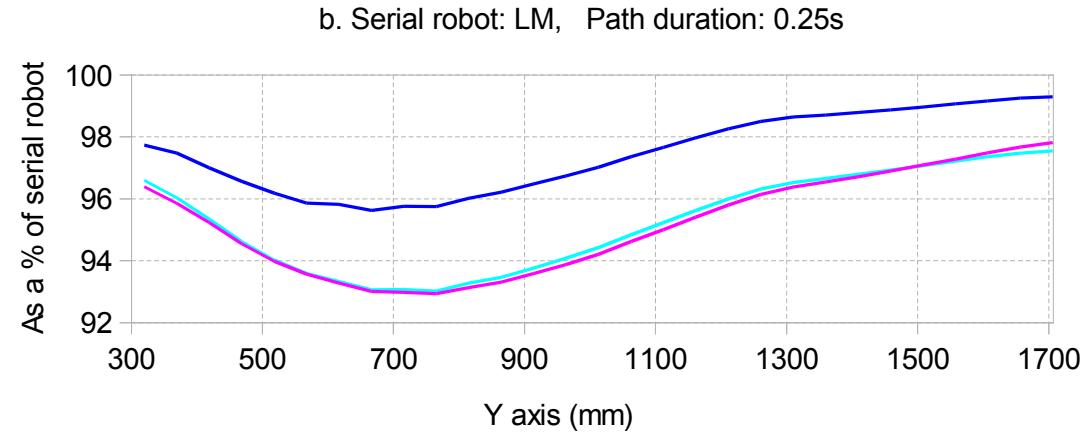
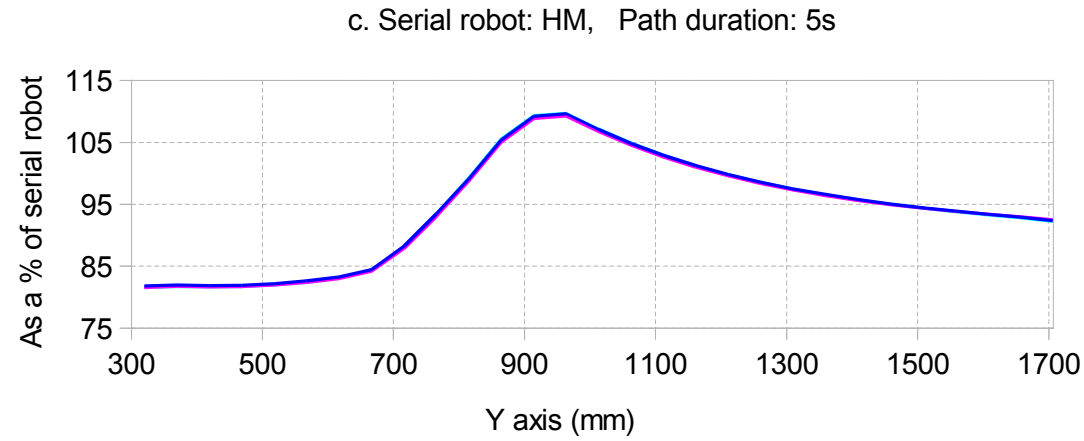


Figure 95 (a): Energy consumption data for multiple vertical paths (YZ plane)





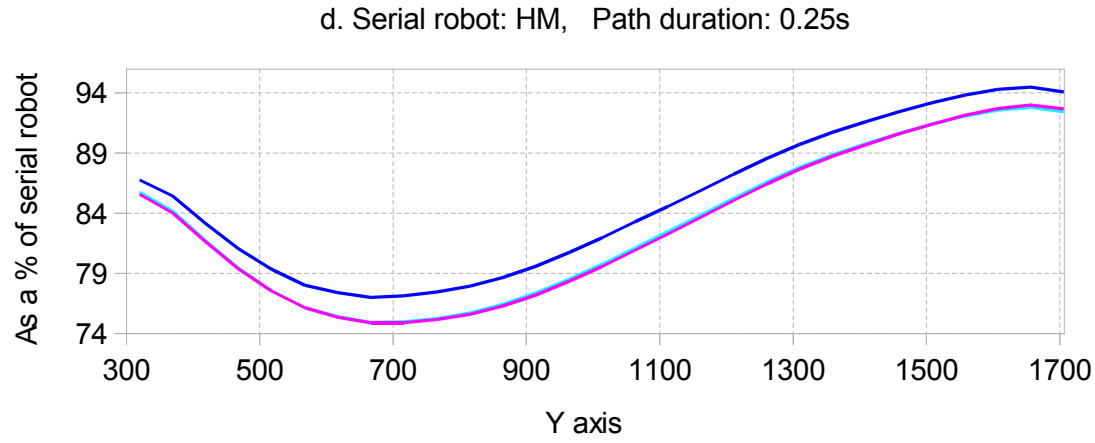


Figure 95(b-d): Energy consumption data for multiple vertical paths (YZ plane)

5.5.2. Multi-horizontal line paths

For the low speed case shown in figure 96a, design 2 had the best energy usage in comparison to the other hybrid models (only marginally better than design 1). Once again there was a region between about 750mm and 1200mm where all hybrid models performed worse than the light motor version of the serial robot. More energy was saved when the horizontal line path was closer to the $\hat{z}_1$ axis, due mostly to the larger displacement in angle 1. It reached a minimum of about 86%, a maximum 14% saving (better than the 3.25% from the previous path's light motor, low speed comparison). When the dynamic response was made dominant, all hybrid designs had better energy usage throughout the y-range for the horizontal paths. The further the horizontal line was from the $\hat{z}_1$ axis the closer the energy consumptions got to the serial robot's. The comparison with the heavy motor serial robot at low and high speeds shown in figures 96c and d, were similar to the comparative graphs in 96a and b respectively, but had more drastic percentages. For the low dynamics case, the hybrid designs dropped to a minimum of 52% of the serial robot's consumption with small y. The hybrid design's consumption jumped to a maximum of 121% in the region where the serial robot was better. As the y value increased from that point the hybrid designs conserved more energy, and it levelled of at about 92%. For the heavy motor high dynamics case the hybrid designs performed better than the serial robot for the entire y range. The best ratio achieved was 45%, by both design 1 and 2 and this was close to the $\hat{z}_1$ axis. That saving was mostly due to the fact that angle 1 had a larger change for horizontal paths (with the same length) that were closer to the $\hat{z}_1$ axis. Its ratios levelled off at about 89% for horizontal lines that were far from $\hat{z}_1$. Design 2 performed the best in all cases but only slightly better than design 1, which could be seen in the tabular data in appendix C, pages C2 to C13 in the accompanying CD.

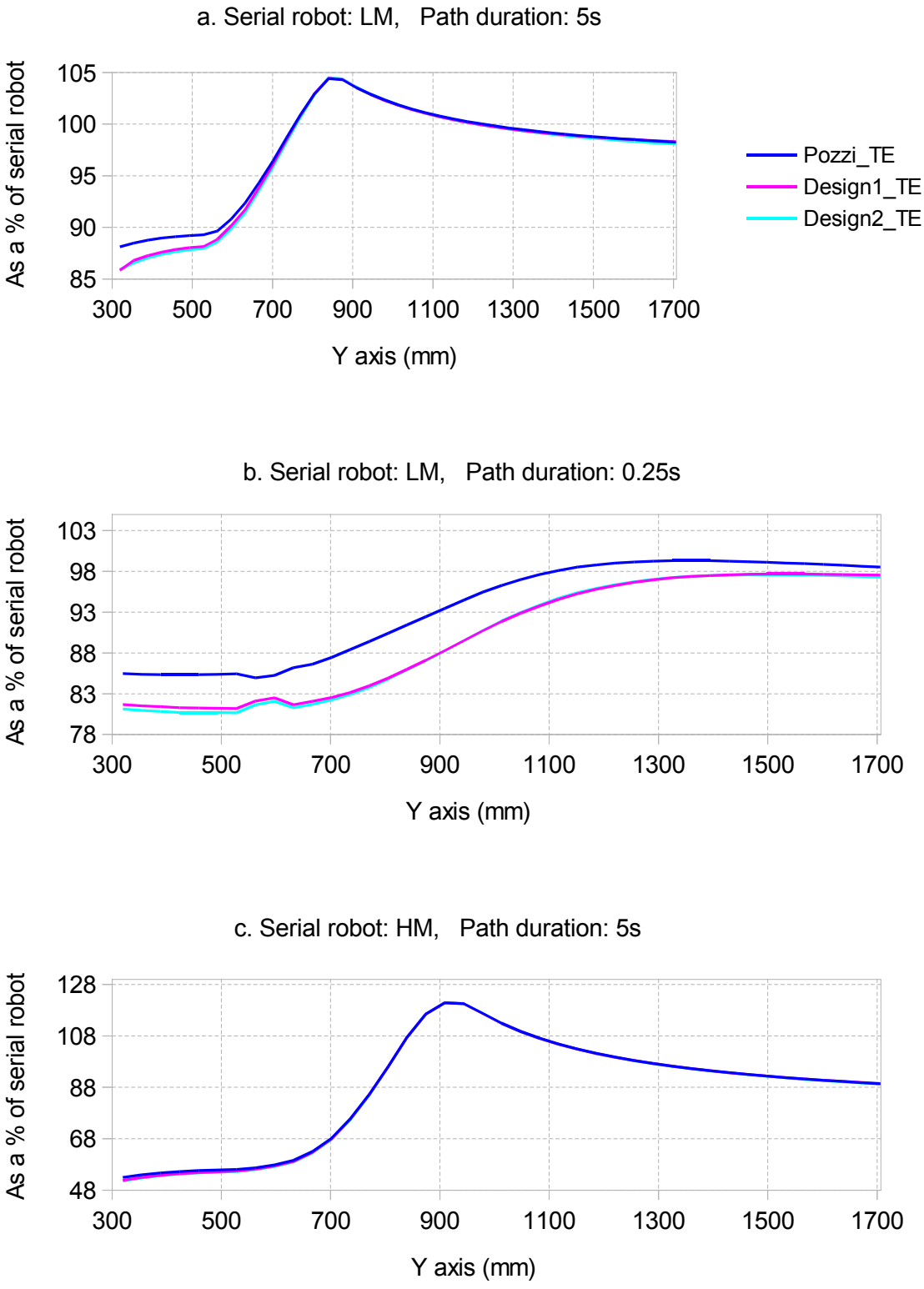


Figure 96 (a-c): Energy consumption data for multiple horizontal paths (XY plane)

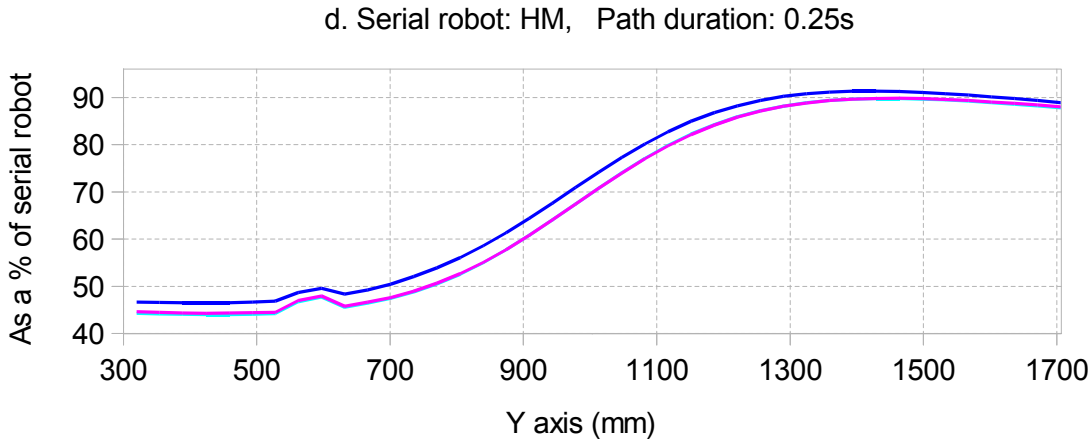
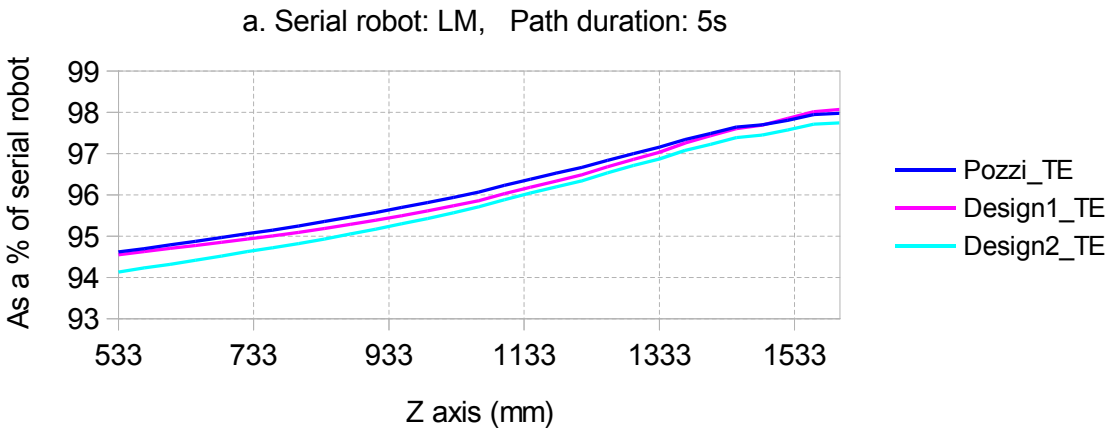
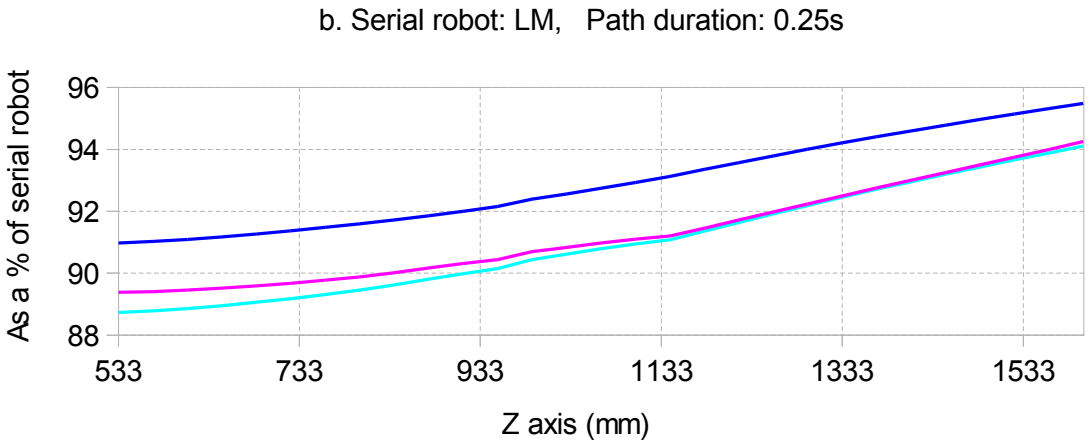


Figure 96(d): Energy consumption data for multiple horizontal paths (XY plane)

5.5.3. Multi-radial line paths

For these paths all hybrid models conserved energy better than the serial robot for all cases as shown in figure 97a-d. The graphs were close to linear and design 2 had the best energy consumption in all cases. Also for each case the hybrid models' joint work done got closer to the serial robot's as the path height increased. Design 2's best percentages over the light motor version of the serial robot, were 94% and 89% for the low and high speed cases respectively. Design 2's best percentages over the heavy version of the serial robot were 71% and 59% respectively.





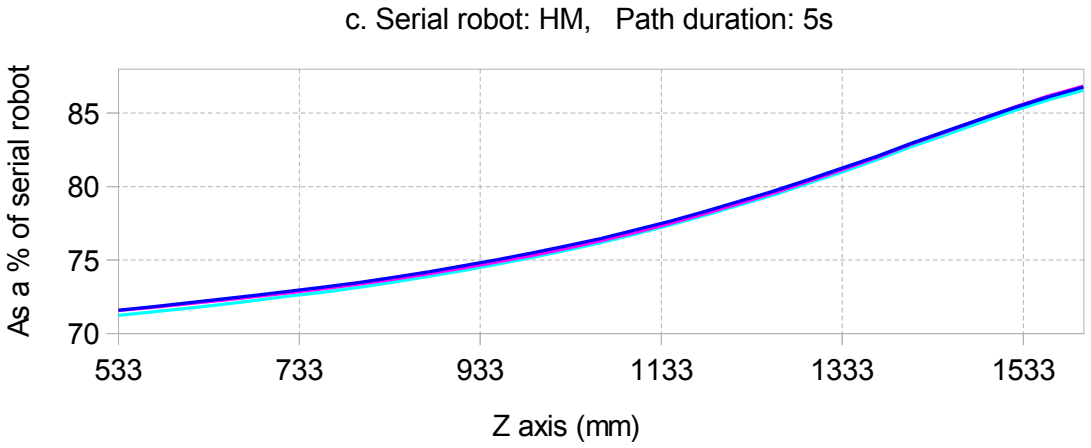


Figure 97(a-c): Energy consumption data for multiple radial paths (YZ plane)

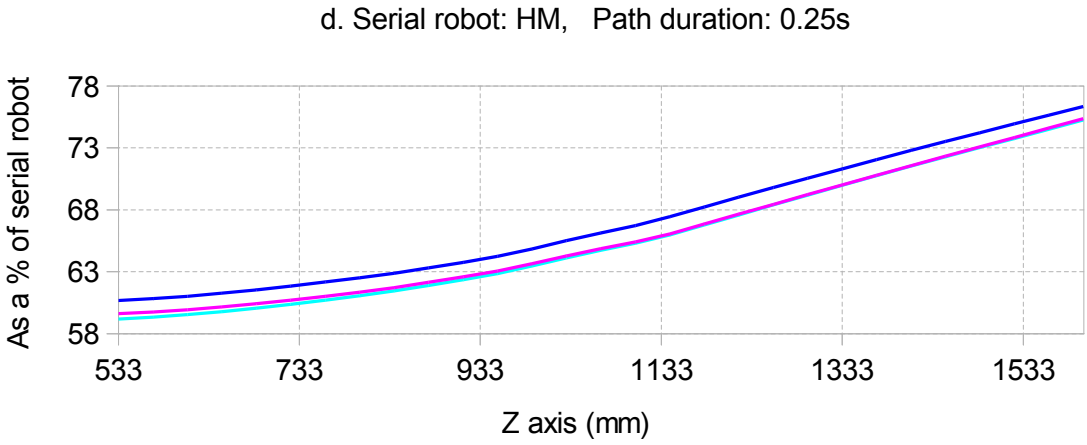


Figure 97(d): Energy consumption data for multiple radial paths (YZ plane)

5.6. Circular path simulation results

The simulation results presented here were a comparison of total energy usage only (total energy consumed per model at the end of each circular path), and is illustrated in tables 15 to 17, and figure Error: Reference source not found. Complete data tables and additional graphs can be viewed in appendix C from page C38 to C97, in the accompanying CD. The data presented for each hybrid design was as a percentage in relation to the serial robot, for both LM and HM cases. The circular path specifications were displayed in table 14 on page 173.

5.6.1. Varying angle 1 only

The percentage energy comparison against the serial robot when varying angle 1 was constant throughout the range for angle 1. This fact allowed presentation in tabular form. Additionally the corresponding percentages for high and low speed were the same for each hybrid design. For all cases design 1 had the best performance. Even though the goal of sub 50% comparative energy usage was reach for the heavy motor case with robot configuration $\theta_2 = 90^\circ$ and $\theta_3 = 0^\circ$, that was actually the best case mass distribution for joint axis 1 (lowest mass moments about that axis), and the total energy used was minimal.

Table 15: Accumulated joint work comparison for circular paths varying angle 1 only

Configuration	Serial robot model	Hybrid designs	Energy usage percentages (%)	
			Low speed (40s)	High speed (2s)
$\theta_2 = 0^\circ$ $\theta_3 = 0^\circ$	LM	Design 1	94.49	94.49
		Design 2	96.09	96.09
		Pozzi	97.14	97.14
	HM	Design 1	80.71	80.71
		Design 2	82.07	82.07
		Pozzi	82.98	82.98
$\theta_2 = 90^\circ$ $\theta_3 = 0^\circ$	LM	Design 1	96.86	96.86
		Design 2	97.86	97.86
		Pozzi	100.18	100.18
	HM	Design 1	43.52	43.52
		Design 2	43.97	43.97
		Pozzi	45.01	45.01

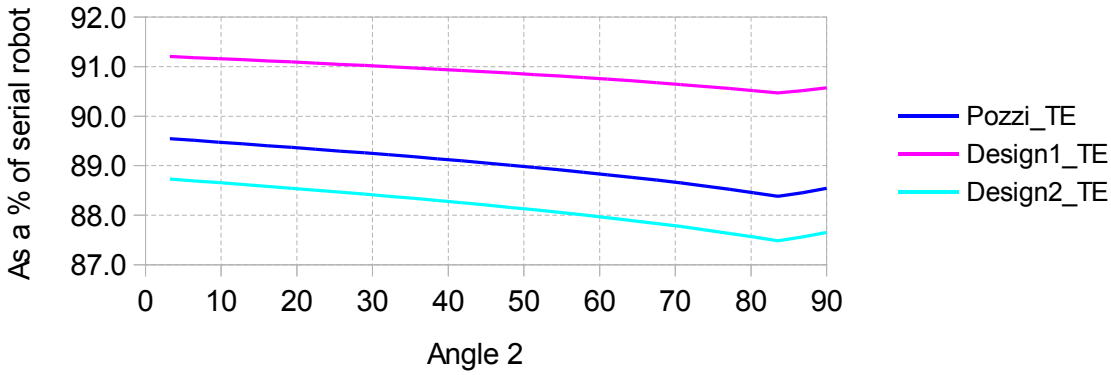
5.6.2. Varying angle 2 only

The percentages for the worst case mass moments about axis 2 ($\theta_3 = 0^\circ$) were very close to constant (varying by only 0.01% in each case) for most of angle 2's range, it was thus indicated as a constant in table 16. Additionally the corresponding percentages for high and low speed were the same for each hybrid design. The complete data sets and graphs can be seen in appendix C, from pages C62 to C73 in the accompanying CD. For the best case mass moments about joint 2, the percentages varied more drastically and the graphs are thus illustrated in figure Error: Reference source not found. Design 1 had the best percentages for the first configuration (worst case mass moments about joint 2), and had the worst percentages for the second configuration.

Table 16: Accumulated joint work comparison for circular paths varying angle 2 only

Configuration	Serial robot model	Hybrid designs	Energy usage percentages (%)	
			Low speed (40s)	High speed (2s)
$\theta_1 = 0^\circ$ $\theta_3 = 0^\circ$	LM	Design 1	94.23	94.23
		Design 2	95.33	95.33
		Pozzi	95.96	95.69
	HM	Design 1	76.04	76.04
		Design 2	76.93	76.93
		Pozzi	77.22	77.22

a. Serial robot: LM, Path duration: 40s



b. Serial robot: LM, Path duration: 2s

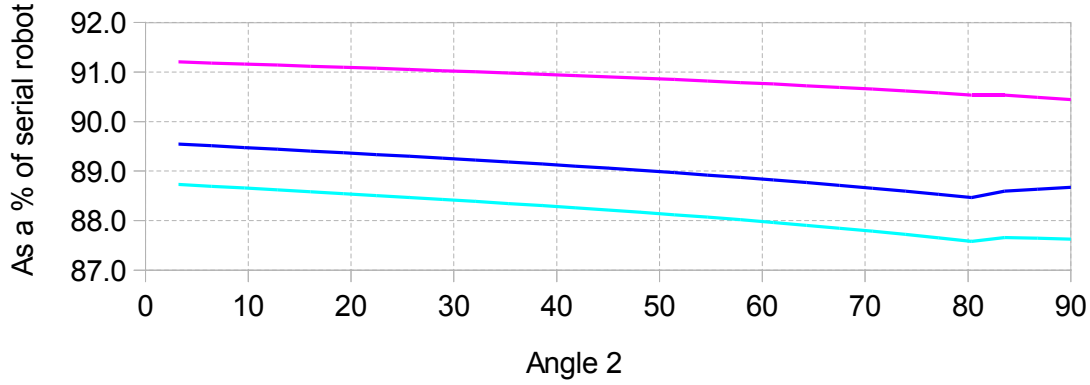
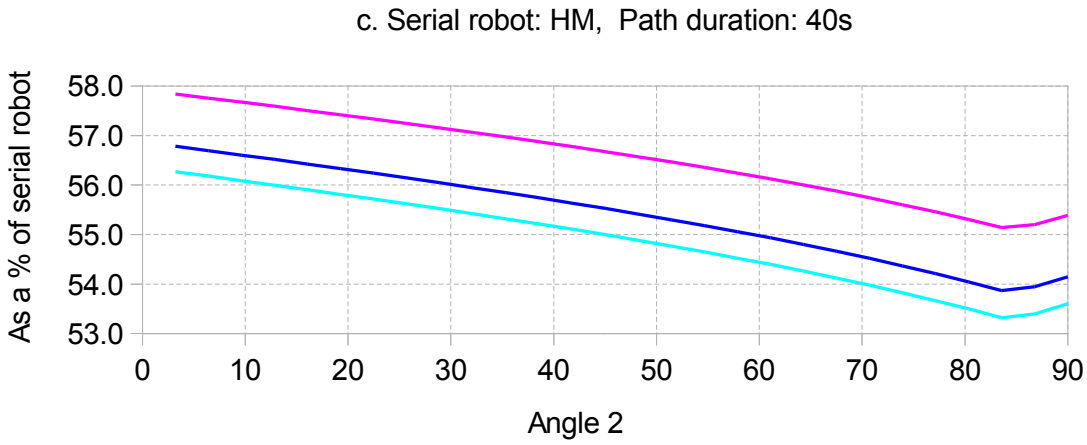


Figure 98(a,b): Energy consumption data for multiple circular paths varying angle 2 only with $\theta_1 = 0^\circ$ and $\theta_3 = 170^\circ$



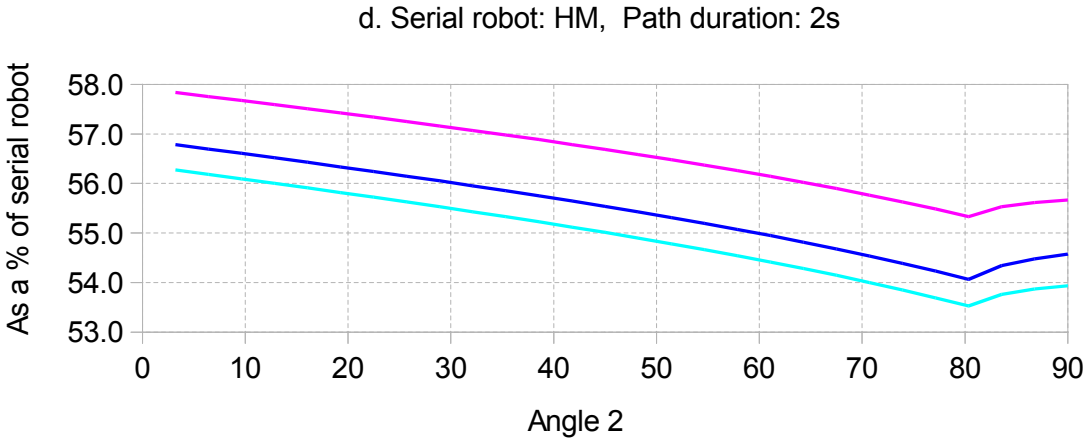


Figure 98(c,d): Energy consumption data for multiple circular paths varying angle 2 only with $\theta_1 = 0^\circ$ and $\theta_3 = 170^\circ$

5.6.3. Varying angle 3 only

The percentage energy comparison against the serial robot when varying angle 3 was constant throughout the range for angle 3. Additionally the corresponding percentages for high and low speed varied by only 0.01% for each hybrid design. For all cases design 1 had the best performance.

Table 17: Accumulated joint work comparison for circular paths varying angle 3 only

Configuration	Serial robot model	Hybrid designs	Energy usage percentages (%)	
			Low speed (40s)	High speed (2s)
$\theta_1 = 0^\circ$ $\theta_2 = 0^\circ$	LM	Design 1	98.41	98.42
		Design 2	100.55	100.56
		Pozzi	100.55	100.56
	HM	Design 1	101.26	101.3
		Design 2	103.48	103.5
		Pozzi	103.48	103.5

5.7. Serial robot's static balancing at joint axis 3

The total mass moment about joint axis 3 for the heavy motor version of the serial robot was the best of all dynamic models, even though the size of mass moved was larger. This resulted in lower angular work done for that joint (indicated in table 17, HM). Since mass was present on both sides of the axis (wrist motors on one side and link mass on the other) there was a mass balancing effect and the complete mass moment about that joint was lower. If the serial robot was to have a greater joint torque at axis 3 than any of the hybrid models then $m_{l3}\cdot\frac{l_2}{2} < (m_{l3}+m_{m4})\left(\frac{l_2}{2} - \frac{m_{m4}\left(\frac{l_2}{2}-d_3\right)}{(m_{m4}+m_{l3})}\right)$ had to be true. This was with reference to figure 82 on page 131. The moment to the left of the < sign was applicable for the hybrid designs, and the portion to the right was for the serial robot. Here m_{l3} represented the mass of the structural portion of the distal link, and all models had this same mass. m_{m4} Was the collective mass of the wrist motors for the serial robot. $\frac{m_{m4}\left(\frac{l_2}{2}-d_3\right)}{(m_{m4}+m_{l3})}$ Was the change in the complete distal link COM location due to the wrist motor mass, whose COM was a distance d_3 from joint axis 3 ($\hat{z}_3$). Multiplying out and simplifying gave $m_{l3}\cdot\frac{l_2}{2} < m_{l3}\cdot\frac{l_2}{2} + m_{m4}\cdot\frac{l_2}{2} - m_{m4}\left(\frac{l_2}{2}-d_3\right) \Rightarrow m_{m4}\left(\frac{l_2}{2}-d_3\right) < m_{m4}\cdot\frac{l_2}{2} \Rightarrow -m_{m4}\cdot d_3 < 0$ or

$d_3 > 0$. This meant that the wrist motor COM would have to be on the same side of axis $\hat{z}_3$ as the COM of the structural part of the distal link. That would defeat the purpose of the balancing effect. Thus the serial robot's joint 3 torques were the best, however the LM version comparison to design 1 (indicated in table 17, LM, first row) indicated otherwise. That result was due to the fact that the distal link COM for design 1 could change position, due to the position of the wrist torque transfer links. Its joint torque could either be better or worse than the LM serial robot and depended on joint angles 2 to 6. For that configuration it gave better mass moments about axis 3 (and hence better energy usage). The inequality above was still valid, it didn't take into account the torque transfer link's mass and position, which was what the tabular result showed.

If the hybrid designs were to have a better torque for joint 3, the mass of the distal link in each model had to be less by a value δm. Then $m_{l3} \cdot \frac{l_2}{2} < (m_{l3} + \delta m + m_{m4}) \left(\frac{l_2}{2} - \frac{m_{m4}\left(\frac{l_2}{2} - d_3\right)}{(m_{m4} + m_{l3})} \right)$ had to be true, which simplified to $\delta m > \frac{2 \cdot m_{m4} \cdot d_3}{l_2}$.

5.8. Chapter summary

The slider-pivot linkage design code ran a number of cases to determine the best position for pivot location, and looked mainly at the orbits generated. This was for design 1's torque transfer links that controlled the elbow and wrist joints. A centre point pivot on the supporting link of the primary 3 bar slider pivot linkage gave the best output trajectory. The variation in output angular speed and acceleration (even for a constant input angular speed) was flattest at that pivot location. The secondary 3 bar link's pivot was also set at the centre of its supporting link, and the output was tested for a range of input angles as well as angle 3 (the angle between the supporting links of the primary and secondary linkages). The output trajectories, as well as variations in angular speed and acceleration for the centre point pivot were the best that could be obtained for any pivot location.

The simulation results for the dynamic models were then presented, to illustrate the most energy conservative design. Joint work was calculated for the first 3 actuated axes of each model, which were responsible for positioning the robot's wrist centre and consumed the most energy. The simulations were run at 2 speeds, one where the dynamic response was minimal and the other where it was dominant. The hybrid designs were compared against 2 serial robot models, one with light motors and the other with heavy motors. The simulations were run over a number of vertical, horizontal and radial straight line paths, as those were the building blocks to obtain any rectilinear path. They were also run for a number of circular paths, where only 1 of those first 3 joint angles were varied. That isolated the major joint's energy consumption.

The results showed that the hybrid models save on energy even when compared to serial robots with low mass motors. For some parts of the vertical and horizontal paths in the low dynamics case, the serial robot was more energy efficient, and that included the heavy motor version. When the dynamic response was made excessive all hybrid designs were better. Overall design's 1 and 2 performed the best, but they only reached the 50% saving mark for a small range in the horizontal path data set, against the heavy motor serial robot. Even when each joint was isolated in the circular paths, the worst case robot configurations which would have a large energy consumption only reached a minimum of 76% (moving joint 2 only). These results were quite extensive and conclusive and it was felt that none of these designs were a suitable replacement for the serial robot.

6. Conclusion

The 2 aims and scientific contribution of this research PhD project were realized:

- Two novel 6 DOF hybrid machines were designed (one more than what was initially intended). They both had a lower machine moving mass in comparison to a serial robot, with an equally large workspace and dexterity. In both designs all actuators were fixed to the stationary base.
- Simulated tests were conducted over a number of straight line and circular paths, and the hybrid designs were compared to each other and the serial robot. The results indicated that the hybrid designs of this thesis were better than the existing patents. They could save energy but those percentages were not significant enough.

The main objectives of this thesis were accomplished:

- An extensive research study was undertaken across a number of robotic manipulators with varying architectures i.e., parallel, serial and hybrid. A noticeable gap was the lack of low inertia robots similar to PKMs, having a large workspace to footprint ratio similar to serial robots.
- Mechanisms were researched and some were designed that could be used in the hybrid manipulator, these included various linkages and geared mechanisms.
- The designs were simplified and modelled mathematically for both the kinematics and dynamics.
- Motion simulations were completed for simple and complex paths in both Scilab and SolidWorks, and videos were created which appear on the accompanying CD.
- A scaled model was built using perspex and servo motors for design 1 which had 4 working DOFs. That was sufficient to prove that the geared mechanisms and links for that design, when put together, were functional.

- Tests were created to perform an energy analysis. This last objective became the focus towards the end of this thesis and was accomplished quite rigorously for the designs presented. The designs were also compared against an existing patent and the serial robot.

There are 2 broad classifications of industrial robots i.e., serial kinematic machines and parallel kinematics machines. Serial kinematic chains were open ended paths, whereas parallel kinematic chains had closed paths. Each architecture had its own advantages and disadvantages and generally they were complementary. The serial robot was dominant in industry and had a multitude of applications, whereas parallel robots were highly specialized but for their specific applications they outperformed serial machines. Hybrid machines could bridge the gap between serial and parallel robots, and were usually identified as having distinct serial and parallel parts that were connected together serially.

Two low inertia concept designs were discussed in this thesis, and in both designs the motors used to create joint motion were placed at the stationary base. The main aims of these designs were to maintain the large workspace of the serial robot and have a large end-effector dexterity with minimal inertia. It was hoped that the reduced inertia would improve the energy usage profile when compared to a serial robot, if they were used under the same operating conditions. A reduced inertia could mean that for the same energy input the robot arm could move faster, or carry a heavier payload or reach further, possibly a moderate combination of all 3. The main robot joints were actuated through a number of linkages. In the first design 3 bar slider-pivot linkages were used, in the second extended torque transfer rods were used with gear heads at both ends. In the prior art search there were 2 patented designs that attempted to accomplish the same goal, these were a European patent, from *Bisiach (1989)* [86] and a German patent from *Pozzi (1989)* [87]. Both patents described a robot with actuators at the base, which used coaxial shafts and concentric geared mechanisms to transfer torque to the correct intended axis. When compared to the subject designs of this thesis those designs had a few significant disadvantages i.e. they would be

more expensive to manufacture and maintain, they used more material, had an extra DOF (this was the most significant as those were 7 DOF designs), were more energy inefficient and less accurate. Since there were a number of gears used in these designs, the problem of gear backlash was an issue. However for designs 1 and 2, high accuracy gearboxes (such as harmonic and cycloidal drives), could be positioned at the main joints, to control the angle between links i and i+1 (self locking gearboxes could also be used). These types of gearboxes are not back drivable and hence the inaccuracy caused by a large number of gears would not affect the controlled joint. The accumulated backlash however would appear as a time delay between the desired time of an angle change and the actual motion at the joint. Designs 1 and 2 were illustrated from the base up, and described how all parts fitted together and how everything worked. There was some doubt as to the classification of these designs but ultimately it was classified as hybrid, since the designs exhibited characteristics of both a serial and parallel robot. The mechanical designs were then modelled mathematically so that simulations could be carried out. The first part that was modelled was the 3 bar slider-pivot linkage used in design 1, and from that the forward kinematics was solved easily. The inverse kinematics for that linkage however did not have a closed form solution and an iterative algorithm had to be used to obtain the input joint angle when given an output angle. Some force modelling was illustrated to show how the linkage transferred force from the input to the output location via the slider-pivot link. The machine level kinematics and dynamics represented the bulk of the modelling section. A skeleton line model was used to represent the kinematics of both designs, and this model would be used later for the serial and Pozzi models in their dynamic simulations as well. This line model kept the essential aspects of the designs that were needed to solve for the end-effector's position and orientation and all joint angles. Once the Denavit-Hartenberg parameters were established the forward kinematics could be solved through established joint frame matrix multiplication. The inverse kinematics on the other hand was solved using a mixed geometric-algebraic approach, and the problem was split into 2 parts. This could be done because the wrist was designed to be spherical, i.e. all 3 wrist joint axes were concurrent, and the orientation of the wrist could be decoupled from its spatial location. First a coordinate geometry type approach was used to solve for the first 3 joint angles that would position the

wrist origin in the workspace. Matrix multiplication was then used to separate the unknown wrist joint angles used to orient the end-effector from the angles used for spatial positioning just calculated, and the known wrist orientation matrix. Basic trigonometry was then used to solve for the wrist angles. In general there were a total of 8 robot configurations for any end-effector pose. The dynamics algorithm from *Craig (2005)* [89] was then discussed, which used the iterative Newton-Euler approach. There were 2 phases of this algorithm, the first of which moved in an outward direction from the robot base to the end-effector and was used to calculate the velocities and accelerations of coordinate frame origins and COMs, for each pose in a time stamped robot path. The inward iteration then moved in the opposite direction from the end-effector to the robot base and used those velocities and accelerations to calculate forces and torques. This algorithm required knowledge of inertia tensors and locations of COMs for each robot configuration along a path, and for that reason simplified versions of designs 1 and 2 were created. The simplifications maintained their essential characteristics, and facilitated calculation of those parameters. Simplified dynamic models were also created for a typical serial robot and the Pozzi patent (which referred to both the Pozzi and Bisiach designs). The Pozzi model however was not exactly equivalent as it had 1 less DOF than the actual design and had less moving parts. Hence this simplified model was more energy efficient than the real design could ever hope to be. The reasons for creating these models were to compare the energy usage characteristics of the hybrid designs against the serial robot, and to determine which design was most energy efficient. The analysis did not consider the strength of specific shapes, and was limited only to an energy comparison which ignored friction.

A short chapter which highlighted the most important aspects in the construction of a scaled functional model for design 1 followed. The model was made using cheap perspex and model helicopter servos, so that multiple configurations/changes/fixes could be implemented at low cost. The model highlighted some structural problems, and also magnified backlash issues. Changes were made that got the model to work, and 4 DOFs were completed. A video of the forward kinematic control is attached in the accompanying CD, which shows construction detail as well.

There were a number of Scilab scripts written to generate the results displayed in the simulations chapter which were described. Some instructions were given on how to run the code which appears on the attached CD. The scripts were separated into 3 categories (2 of which were inter-linked), i.e. 3bar slider pivot linkage design, machine level kinematics (forward and inverse) and machine level dynamics (which was entirely dependent on the kinematics). The code can be seen in appendix A.

The slider-pivot linkage design code ran a number of cases to determine the best position for pivot location, and looked mainly at the orbits generated. This was for design 1's torque transfer links that controlled the elbow and wrist joints. A uniform circular output was desired but the closest that the linkage could provide was used, and the choice of pivot location that did that best was at the centre point of the supporting link. Furthermore the variation in output angular speed and acceleration (even for a constant input angular speed) was flattest at that pivot location. The pivot for the secondary linkage was also set at the mid-point of its supporting link, and the output was tested for a range of input angles to the primary linkage as well as angle 3, which configured the angle between the supporting links of the primary and secondary linkage sets. The output trajectories were compared as well as the variations in angular speed and acceleration for constant input angular speed. Those curves were the best that could be obtained for any pivot location.

The results from simulations of the dynamic models for the hybrid and serial robot designs were then presented. The reason for these simulations was to determine the most energy efficient design in terms of usage (other criteria such as structural integrity etc., were not considered). Due to financial costs related to the construction and commissioning of the new hybrid robots, as well as energy consumed for additional moving parts in their designs, it was stated that the hybrid designs would have to be at least 50% more efficient to even be considered a worthy replacement for the serial robot. The simulations were run at 2 speeds, one where the dynamic response was minimal and the other where it was dominant, even if

the actual speeds set were excessive. The reason for doing that was to present a stark contrast in operating behaviour and to determine where exactly the hybrid robots were better (if indeed they were). Additionally the hybrid designs would be compared against 2 serial robot models, one with ultra light motors and the other with heavy motors. Again this was to figure out if moving the motor mass to the robot base in the hybrid designs would improve energy usage, and at what point in terms of serial robot motor mass. The simulations were also carried out for a number of straight line and circular paths. Robot paths consisted of a multitude of straight line and curved segments, and there were 3 basic straight edge components that were the building blocks to obtain any rectilinear path. For that reason all 4 robot dynamic models were tested on multiple vertical, horizontal and radial line paths. The models were also tested on circular paths, where only 1 angle was varied, for both the best and worst case robot configurations (in relation to the joint being controlled). The simulation results displayed the energy used at the end of each path, as a percentage of the serial robot's work done (both versions) at both low and high speed.

From the results it could be seen that the hybrid models do save on energy even for low mass motors. There were some parts of the vertical and horizontal paths in the low dynamics case, where the serial robot was more energy efficient including the heavy motor version. However in all cases when the dynamic response was made excessive all hybrid designs performed better. Overall designs 1 and 2 performed best. The Pozzi model was the worst and it must be remembered that the model was simplified (missing a DOF and some links, the real Pozzi and Bisiach designs were 7 DOF), and hence used less energy than the actual design would. None of models reached the 50% saving mark for a significant range in the workspace. The conclusion was that none of these designs were a suitable replacement of the serial robot.

The serial robot's mass balancing effect at joint 3 was quite significant and helpful. Improvements to the hybrid designs could be to leave the wrist motors on the opposite side

of axis 3 in relation to the distal link, and only place motors 2 and 3 at the stationary base (using the concentric geared mechanism). The serial robot could also be improved by placing the motor for axis 3 (which was attached to the proximal arm), with the wrist motors on the distal arm. That would bring the COM of the distal arm closer to axis 3, and might reduce the mass moment on axis 3. Also the same balancing trick could be attempted at the robot base for motor 2 i.e. placing that motor on the opposite side of axis 2 in relation to the proximal arm. The base would then have to be properly designed to allow a large range of motion about axis 1. These are design options that could be investigated in another study.

To conclude the aims set forth in this thesis were realized, and all the objectives listed have been met.

References

[1]: No Author Info., "Robot Sales in 2011 Exceeded All Expectations", IFR (International Federation of Robotics), Publication type: Article, 2012, Pages: NA, *http://www.ifr.org/news/ifr-press-release/robot-sales-in-2011-exceeded-all-expectations-361/*.

[2]: No author info, "Executive Summary of World Robotics 2011 – Industrial Robots", World Robotics, Publication type: Article, 2011, Pages: NA, *http://www.worldrobotics.org/uploads/media/2011_Executive_Summary.pdf*.

[3]: ABB Robotics, "10 Good Reasons to invest in Robotics", Publication type: Brochure, 2007, *www.abb.com/robotics*.

[4]: No author info., "Executive Summary of World Robotics 2009 – Industrial Robots", World Robotics, Publication type: Article, 2009, Pages: NA, *http://www.dis.uniroma1.it/~deluca/rob1/2009_WorldRobotics_ExecSummary.pdf*.

[5]: M. Zoppi, "High Dynamics Parallel Mechanisms Contributions to Force Transmission and Singularity Analysis", Publication type: Thesis, Dept. of Mechanics and Machine Design, The University of Genoa, March 2004, *http://www.dimec.unige.it/pmar/pages/download/theses/Zoppi4_1_28.pdf*.

[6]: D. Meike and L. Ribickis, "Analysis of the Energy Efficient Usage Methods of Medium and High Payload Industrial Robots in the Automobile Industry", Publication type: Conference, 10th International Symposium - Topical Problems in the Field of Electrical and Power Engineering, Pärnu-Estonia, January 2011, Pages: 62-66, ISBN/ISSN: NA, *http://egdk.ttu.ee/files/parnu2011winter/Parnu2011_winter_062-066.pdf*.

[7]: No author info., "Industrial Robots Used to Conserve Energy", Publication type: Internet article, RobotWorx, September 2007, *http://www.robots.com/blog/viewing/industrial-robots-used-to-conserve-energy/28*.

[8]: No author info., "Case Studies of Industrial Robots - Whittman Austria", Publication type: Internet article, International Federation of Robotics, 2011, *http://www.ifr.org/industrial-robots/eco-tech-robotics/wittmann-austria-ifr-partner-144/*.

[9]: No author info., "Case Studies of Industrial Robots - Staubli France - Natural Solutions to Green Automation", Publication type: Internet article, International Federation of Robotics, 2011, *http://www.ifr.org/industrial-robots/eco-tech-robotics/staubli-france-ifr-partner-182/*.

[10]: P. S. Donelan, "Singularities of Robot Manipulators", Publication type: Conference, 2005 Marseille Singularity School and Conference, Marseille-France, 2007, Pages: 189-217, ISBN/ISSN: 9789812704108, *http://www.mendeley.com/research/singularities-of-robot-manipulators/*.

[11]: M. Carricato and V. Parenti-Castelli, "A Novel Fully Decoupled Two- Degrees-of-Freedom Parallel Wrist", Publication type: Journal, The International Journal of Robotics Research, June 2004, Vol. 23, No. , Pages: 661-667, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/23/6/661.abstract*.

[12]: M. Zoppi, L. Bruzzone and R.M. Molfino, "A Novel 5 DOF Interconnected-Chains PKM for Manufacturing of Revolute Surfaces", Publication type: Conference, Proceedings of the 4th Parallel Kinematics Seminar (PKS2004), Chemnitz-Germany, April 2004, Pages: 437-448, ISBN/ISSN: 3-937524-05-3, *http://www.dimec.unige.it/pmar/pages/the%20lab/publications.htm*.

[13]: I. A. Bonev, "Geometric Analysis of Parallel Mechanisms", Publication type: Thesis,

facult´e des sciences et de g´enie, universit´e laval, November 2002, *http://robot.gmc.ulaval.ca/docs/theses/ilian_bonev.pdf*.

[14]: L. E. Bruzzone, R. M. Molfino and M. Zoppi, "Mechatronic design of a parallel robot for high-speed, impedance-controlled manipulation", Publication type: Conference, The 11th Mediterranean Conference on Control and Automation, Rhodes-Greece, June 2003, Pages: NA, ISBN/ISSN: 960-87706-1-0, *http://med.ee.nd.edu/MED11-2003/begin.pdf*.

[15]: A. Tasora, P. Righettini and S. Chatterton, "Design of the 'Granit' Parallel Kinematic Manipulator", Publication type: Conference, 14th International Workshop on Robotics in Alpe-Adria-Danube Region, Bucharest-Romania, May 2005, Pages: 1-6, ISBN/ISSN: NA, *http://ied.unipr.it/tasora/pubblicazioni/iadat2005_tasora.pdf*.

[16]: M. G. Her, C. Y. Chen, M. Karkoub and Y. C. Hung, "A Generalised Approach to Point-to-Point Motion of Multi-DOF Parallel Manipulators Illustrated with TMPM", Publication type: Journal, International Journal of Advanced Manufacturing Technology, July 2003, Vol. 22, No. , Pages: 216-223, ISBN/ISSN: ISBN/ISSN: 0268-3768, *http://www.springerlink.com/content/nqd7j7g0pfv9t8j8/*.

[17]: X. Tang, X. J. Liu and J. Wang, "A Novel 2-DOF Parallel Mechanism Based Design of a New 5-Axis Hybrid Machine Tool", Publication type: Conference, Proceedings of the 2003 IEEE International Conference on Robotics & Automation, Taipei-Taiwan, September 2003, Pages: 3990-3995, ISBN/ISSN: 0-7803-7736-2, *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1242210*.

[18]: S. Rowe, "A Parallel Robot for the Strain Imager (SALSA)", Institut Laue-Langevin (ILL), Publication type: Article, 2002, Pages: NA, *http://www.ill.eu/fileadmin/users_files/Annual_Report/AR-02/site/areport/fset_96.htm*.

[19]: C. Kossowski and L. Notash, "A Novel Wire Actuated Parallel Robot with Space Applications", Publication type: Conference, Proceedings of the 2001 CCToMM Symposium on Mechanisms, Machines, and Mechatronics, Quebec-Canada, June 2001, Pages: NA, ISBN/ISSN: NA, *http://www.engr.uvic.ca/~ram/2001_CCToMM_Symposium/call_eng.html*.

[20]: Y. Y. Kuen, "Geometry, Dynamics and Control of Parallel Manipulators", Publication type: Thesis, , Hong Kong University of Science and Technology, August 2002, *www.ee.ust.hk/~eeyoyo/thesis.pdf*.

[21]: J. G. Persson and K. Anderson, "Modelling and Model Based Performance Prediction for Parallel Kinematic Manipulators", Publication type: Conference, Mekatronikmötet 2003, Gothenborg-Sweden, August 2003, Pages: NA, ISBN/ISSN: NA, *http://apps.md.kth.se/publication_item/web.phtml?ss_brand=MMKResearchPublications&department_id=%27ED%27&PHPSESSID=09b71e43b6e92e39842b9bc1445c76ca*.

[22]: Y. S. Martín, M. Giménez and V. Nabat, "Don't Ignore PKM Machines", Manufacturing Engineering, Publication type: Article, 2008, Pages: NA, *http://www.blnz.com/news/2008/05/14/DONT_IGNORE_MACHINES_4647.html*.

[23]: Y. M. Moon and S. Kota, "Design of Compliant Parallel Kinematics Machines", Publication type: Conference, 2002 ASME Design Engineering Technical Conference (DETC 2002) - 27th Biennial Mechanisms and Robotics Conference, Montreal-Canada, October 2002, Pages: NA, ISBN/ISSN: NA, *http://www.engin.umich.edu/labs/csdl/papers/CPKM_DETC_2002.pdf*.

[24]: B. Dasgupta, "Recent Trends in Robotics", , Publication type: Article, 2008, Pages: NA, *http://home.iitk.ac.in/~dasgupta/teaching/robot1/recent.pdf*.

[25]: S. Briot and I. A. Bonev, "Are Parallel Robots more Accurate than Serial Robots?", Publication type: Journal, Transactions of the Canadian Society for Mechanical

Engineering, 2007, Vol. 31, No. 4, Pages: 445-455, ISBN/ISSN: 0315-8977, *http://www.tcsme.org/Vol31-No4.html*.

[26]: H. Bruyninckx, "Parallel Robots", The Robotics Webbook, Publication type: Book, 2005, ISBN/ISSN: NA.

[27]: A. Burisch, S. Soetebier, J. Wrege and Rolf Slatter, "http://mikroharmonicdrive.de/pdf/talks/2005_3-mechrob-1808.pdf", , Publication type: Article, 2005, Pages: NA, *http://mikroharmonicdrive.de/pdf/talks/2005_3-mechrob-1808.pdf*.

[28]: F. Hao and J.P. Merlet, "Multi-Criteria Optimal Design of Parallel Manipulators Based on Interval Analysis", Publication type: Journal, Journal of Mechanism and Machine Theory, February 2005, Vol. 40, No. 2, Pages: 157-171, ISBN/ISSN: NA, *http://www.sciencedirect.com/science/article/pii/S0094114X04001211*.

[29]: Y.J. Lou, G.F. Liu, and Z.X. Li, "A General Approach for Optimal Kinematic Design of Parallel Manipulators", Publication type: Conference, IEEE International Conference on Robotics and Automation (ICRA'04), Louisiana-USA, April-May 2004, Pages: 1050-4729, ISBN/ISSN: 0-7803-8232-3, *http://ai.stanford.edu/~liugf/TRO-LLL.pdf*.

[30]: I. A. Bonev and C. M. Gosselin, "A Geometric Algorithm for the Computation of the Constant-Orientation Workspace of 6-RUS Parallel Manipulators", Publication type: Conference, ASME 2000 Design Engineering Technical Conferencesand Computers and Information in Engineering Conference (DETC'00), Maryland-USA, September 2000, Pages: NA, ISBN/ISSN: NA, *http://etsmtl.ca/professeurs/ibonev/documents/pdf/MECH_14106.pdf*.

[31]: J. P. Merlet, "Determination of the Orientation Workspace of Parallel Manipulators", Publication type: Journal, Journal of Intelligent and Robotic Systems, 1995, Vol. 13, No. 2, Pages: 143-160, ISBN/ISSN: 0921-0296, *http://www.springerlink.com/content/r121258320420337/*.

[32]: U. A. Tol, J. P. Clerc and G. J. Wiens, "Micro/Macro Approach for Dexterity Enhancement of PKMs", Publication type: Conference, Proceedings of the WORKSHOP on Fundamental Issues and Future Research Directions for Parallel Mechanisms and Manipulators, Quebec-Canada, October 2002, Pages: NA, ISBN/ISSN: NA, *URL: NA*.

[33]: D. Chablat, P. Wenger, F. Majou and J. P. Merlet, "An Interval Analysis Based Study for the Design and the Comparison of Three-Degrees-of-Freedom Parallel Kinematic Machines", Publication type: Journal, The International Journal of Robotics Research, June 2004, Vol. 23, No. 6, Pages: 615-624, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/23/6.toc*.

[34]: G. Liu, Y. Lou and Z. Li (2003), "Singularities of Parallel Manipulators: A Geometric Treatment", Publication type: Journal, IEEE Transactions on Robotics and Automation, August 2003, Vol. 19, No. 4, Pages: 579 - 594, ISBN/ISSN: 1042-296X, *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1220710*.

[35]: M. Honegger, "Non-linear Adaptive Control of a 6 DOF Parallel Manipulator", Publication type: Conference, 4th International Conference on Motion and Vibration Control (MOVIC'98), Zurich-Switzerland, August 1998, Pages: 961-966, ISBN/ISSN: NA, *http://www.ifr.mavt.ethz.ch/publications/honegger98b.pdf*.

[36]: R. Clavel, M. Thurneysen, J. Giovanola, M. Schnyder and D. Jeannerat, "HITA-STT - A New 5 DOF Parallel Kinematics for Production Applications", Publication type: Conference, Proceedings of the 33rd International Symposium on Robotics, Stockholm-Sweden, October 2002, Pages: NA, ISBN/ISSN: NA, *http://www.mendeley.com/research/new-5-dof-parallel-kinematics-production-applications/#page-1*.

[37]: X. J. Liu, J. Wang, K. K. Oh and J. Kim, "A New Approach to the Design of a DELTA Robot with a Desired Workspace", Publication type: Journal, Journal of Intelligent & Robotic Systems, February 2004, Vol. 39, No. , Pages: 209-225, ISBN/ISSN: 0921-0296, *http://www.springerlink.com/content/q85u123j348n7475/*.

[38]: C.C. Ng, S.K. Ong and A.Y.C. Nee, "Design and Development of 3-DOF Modular Micro Parallel Kinematic Manipulator", Publication type: Journal, The International Journal of Advanced Manufacturing Technology, 2006, Vol. 31, No. 1, Pages: 188-200, ISBN/ISSN: 0268-3768, *http://www.springerlink.com/content/j31667317t8v86w8/*.

[39]: M. A. Laribi, L. Romdhane and S. Zeghloul, "Analysis and Dimensional Synthesis of the Delta for a prescribed workspace", Publication type: Journal, Mechanism and Machine Theory, 2007, Vol. 42, No. 7, Pages: 859-870, ISBN/ISSN: 0094-114X, *http://www.mendeley.com/research/analysis-and-dimensional-synthesis-of-the-delta-robot-for-a-prescribed-workspace/*.

[40]: X. J. Liu, J. Kim and J. Wang, "Two Novel Parallel Mechanisms with Less than Six DOFs and the Applications", Publication type: Conference, Workshop on Fundamental Issues and Future Research Directions for Parallel Mechanisms and Manipulators, Québec-Canada, October 2002, Pages: 172-177, ISBN/ISSN: NA, *URL: NA*.

[41]: H. Giberti, P. Righettini and A. Tasora, "Design and Experimental Test of a Pneumatic Translational 3DOF Parallel Manipulator", Publication type: Conference, RAAD 2001 - 10th International WorkShop on Robotics, Alpe-Adria Danube, May 2001, Pages: NA, ISBN/ISSN: NA, *http://ied.unipr.it/tasora/publications.html*.

[42]: S. Briot and I. A. Bonev, "Accuracy Analysis of 3 DOF Planar Parallel Robots", Publication type: Journal, Mechanism and Machine Theory, 2008, Vol. 43, No. 4, Pages: 445-458, ISBN/ISSN: 0094-114X, *http://www.mendeley.com/research/accuracy-analysis-3dof-planar-parallel-robots-7/*.

[43]: M. Carricato, "Fully Isotropic Four-Degrees-of-Freedom Parallel Mechanisms for Schoenflies Motion", Publication type: Journal, The International Journal of Robotics Research, May 2005, Vol. 24, No. 5, Pages: 397-414, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/24/5/397.abstract*.

[44]: A. Fattah and A. M. H. Ghasemi, "Isotropic Design of Spatial Parallel Manipulators", Publication type: Journal, The International Journal of Robotics Research, September 2002, Vol. 21, No. 9, Pages: 811-824, ISBN/ISSN: 0278-3649, *http://www.ijrr.org/historic/contents/21_09/a031842.pdf*.

[45]: X. Z. Zheng, H. Z. Bin and Y. G. Luo, "Kinematic Analysis of a Hybrid Serial-Parallel Manipulator", Publication type: Journal, The International Journal of Advanced Manufacturing Technology, June 2004, Vol. 23, No. 11, Pages: 925–930, ISBN/ISSN: 0268-3768, *http://www.springerlink.com/content/lbvduy5dgagh7fdx/*.

[46]: H. Cui and Z. Zhu, "Industrial Robotics: Theory, Modelling and Control - Error Modeling and Accuracy of Parallel Industrial Robots", , Publication type: Book, 2006, ISBN/ISSN: 3-86611-285-8.

[47]: M. J. D. Hayes, M. L. Husty And P. J. Zsombor-Murray, "Singular Configurations of Wrist-Partitioned 6R Serial Robots: A Geometric Perspective for Users", Publication type: Journal, Transactions of the Canadian Society for Mechanical Engineering (CSME), 2002, Vol. 26, No. 1, Pages: 41-45, ISBN/ISSN: 0315-8977, *http://www.mendeley.com/research/singular-configurations-wristpartitioned-6r-serial-robots-geometric-perspective-users-configurations-singuli-eres-des-robots-s-poignet-sph-erique-et-six-couples-rot-oides-une-perspective/*.

[48]: ABB Robotics, "IRB 760 Industrial Robot", Publication type: Brochure, 2011, *www.abb.com/robotics*.

[49]: ABB Robotics, "IRB 6640/ IRB 6640ID Industrial Robot", Publication type: Brochure,

2010, *www.abb.com/robotics*.

[50]: J. P. Merlet, "Parallel Robots", Kluwer Academic Publisher, Publication type: Book, 2000, ISBN/ISSN: 0-7923-6308-6.

[51]: M. Carricato and V. Parenti-Castelli, "Singularity-Free Fully-Isotropic Translational Parallel Mechanisms", Publication type: Journal, The International Journal of Robotics Research, February 2002, Vol. 21, No. 2, Pages: 161-174, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/21/2/161.abstract*.

[52]: Z. Zhu, J. Li, Z. Gan, H. Zhang, "Kinematic and Dynamic Modelling for Real-Time Control of Tau Parallel Robot", Publication type: Journal, Mechanism and Machine Theory, 2005, Vol. 40, No. 9, Pages: 1051-1067, ISBN/ISSN: 0094-114X, *http://www.mendeley.com/research/kinematic-dynamic-modelling-realtime-control-tau-parallel-robot-11/*.

[53]: J. Tlusty, J. Ziegert and S. Ridgeway, "Fundamental Comparison of the Use of Serial and Parallel Kinematics for Machines Tools", Publication type: Journal, CIRP Annals - Manufacturing Technology, 1999, Vol. 48, No. 1, Pages: 351-356, ISBN/ISSN: 0007-8506, *http://www.sciencedirect.com/science/article/pii/S0007850607632004*.

[54]: B. R. Hopkins and R. L. Williams II, "Kinematics, Design and Control of the 6-PSU Platform", Publication type: Journal, Industrial Robot, An International Journal, 2002, Vol. 29, No. 5, Pages: 443-451, ISBN/ISSN: 0143-991X, *http://www.mendeley.com/research/kinematics-design-control-6psu-platform/*.

[55]: G. Yang, I. M.Chen, W. Chen and W. Lin, "Kinematic Design of a Six-DOF Parallel-Kinematics Machine with Decoupled-Motion Architecture", Publication type: Journal, IEEE Transactions on Robotics, October 2004, Vol. , No. , Pages: 876-887, ISBN/ISSN: 1552-3098, *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1339387*.

[56]: R. L. Williams II and D. B. Poling, "Spherically-Actuated Platform Manipulator", Publication type: Journal, Journal of Robotic Systems, February 2001, Vol. 18, No. 3, Pages: 147-157, ISBN/ISSN: 1556-4967, *http://onlinelibrary.wiley.com/doi/10.1002/rob.1012/pdf*.

[57]: Y. Jin, I. M. Chen and G. Yang, "Structure Synthesis and Singularity Analysis of a Parallel Manipulator Based on Selective Actuation", Publication type: Conference, IEEE International Conference on Robotics and Automation (ICRA '04), Louisiana-USA, April 2004, Pages: 4533-4538, ISBN/ISSN: 0-7803-8232-3, *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1302432*.

[58]: D. Deblaise, C. Baradat, P. Maurine and V. Arakelian, "Improvement of Positioning Accuracy of Delta Parallel Robot", Publication type: Conference, 21st International Congress of Theoretical and Applied Mechanics (ICTAM 2004), , August 2004, Pages: NA, ISBN/ISSN: NA, *http://fluid.ippt.gov.pl/ictam04/CD_ICTAM04/SM16/10553/SM16_10553.pdf*.

[59]: H. S. Kim, "Kinematic Calibration of a Cartesian Parallel Manipulator", Publication type: Journal, International Journal of Control, Automation, and Systems, September 2005, Vol. 3, No. 3, Pages: 453-460, ISBN/ISSN: 1598-6446, *www.ijcas.org/admin/paper/files/IJCAS_v3_n3_pp.453-460.pdf*.

[60]: X. Kong and C. M. Gosselin, "Kinematics and Singularity Analysis of a Novel Type of 3-CRR 3-DOF Translational Parallel Manipulator;", Publication type: Journal, The International Journal of Robotics Research, September 2002, Vol. 21, No. 9, Pages: 791-798, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/21/9/791.abstract*.

[61]: P. Righettini, A. Tasora and H. Giberti, "Mechatronic Design of a 3-DOF Parallel Translational Manipulator", Publication type: Conference, 11th Workshop on Robotics (RAAD 2002), Alpe-Adria-Danube, 2002, Pages: NA, ISBN/ISSN: NA,

ied.unipr.it/tasora/pubblicazioni/raad2002_robot.pdf.

[62]: Q. Xu, Y. Li and S.J. Pereira, "Kinematic Analysis and Optimization of a New Compliant Parallel Micromanipulator", Publication type: Journal, International Journal of Advanced Robotic Systems, 2006, Vol. 3, No. 4, Pages: 351-358, ISBN/ISSN: 1729-8806, *http://www.doaj.org/doaj?func=abstract&id=737573*.

[63]: Y. Li and Q. Xu, "Kinematic Analysis and Design of a New 3-DOF Translational Parallel Manipulator", Publication type: Journal, Journal of Mechanical Design, July 2006, Vol. 128, No. , Pages: 729-737, ISBN/ISSN: 1050-0472, *www.sftw.umac.mo/~yangmin/journalpdf/J06-1-Li-Xu-JMD.pdf*.

[64]: No author info., "Case Studies of Industrial Robots - Adept USA", Publication type: Internet article, International Federation of Robotics, 2011, *http://www.ifr.org/industrial-robots/eco-tech-robotics/adept-usa-ifr-partner-176/*.

[65]: G. Yang, I. M. Chen, W. K. Lim and S. H. Yeo, "Kinematic Design of Modular Reconfigurable In-Parallel Robots", Publication type: Journal, Autonomous Robots, January 2001, Vol. 10, No. 1, Pages: 83-89, ISBN/ISSN: 0929-5593, *http://dl.acm.org/citation.cfm?id=591845*.

[66]: P. Vischer and R. Clavel, "Argos, a Novel 3 DOF Parallel Wrist Mechanism", Publication type: Journal, The International Journal of Robotics Research, January 2000, Vol. 19, No. 1, Pages: 5-11, ISBN/ISSN: 0278-3649, *http://ijr.sagepub.com/content/19/1/5.abstract*.

[67]: H. Mohammadipanah and H. Zohoor, "Design and Analysis of a Novel 8-DOF Hybrid Manipulator", Publication type: Journal, World Academy of Science, Engineering and Technology, 2009, Vol. 34, No. , Pages: 237-244, ISBN/ISSN: 2010-3778, *http://www.waset.org/journals/waset/v34.php*.

[68]: H. S. Choi, C. S. Han, K. Y. Lee and S. H. Lee, "Development of Hybrid Robot for Construction Works with Pneumatic Actuator", Publication type: Conference, The 20th International Symposium on Automation and Robotics in Construction (ISARC), Eindhoven-Holland, 2003, Pages: 191-196, ISBN/ISSN: NA, *http://www.iaarc.org/publications/fulltext/isarc2003-43.pdf*.

[69]: A. S. Sørensen, O. G. Jakobsen, P. Favrholdt, H. G. Petersen, N. J. Jacobsen and J. Steinicke, "Implementation of a Practical Reconfigurable Manipulator System Based on Hybrid Parallel and Sequential Elements", Publication type: Conference, IEEE International Conference on Intelligent Manipulation and Grasping International Conference (IMG04), Genova-Italy, July 2004, Pages: 404-409, ISBN/ISSN: 88-900426-1-3, *http://www.forskningsdatabasen.dk/Search.external?operation=search&search-query=au:%22Jakobsen+Ole%20Graa%22*.

[70]: L. J. Stocco and S. E. Salcudean, "Hybrid Serial/Parallel Manipulator", Publication type: Patent, Patent number: 6047610, April 2000, *http://www.freepatentsonline.com/6047610.html*.

[71]: S. A. Joshi and L. W. Tsai, "The Kinematics of a Class of 3-DOF, 4-Legged Parallel Manipulators", Publication type: Journal, Journal of Mechanical Design, March 2003, Vol. 125, No. 1, Pages: 52-61, ISBN/ISSN: 1050-0472, *http://dx.doi.org/10.1115/1.1540992*.

[72]: C. Yan, F. Gao and Y. Zhang, "Kinematic Modelling of a Serial–Parallel Forging Manipulator with Application to Heavy-Duty Manipulations", Publication type: Journal, Mechanics Based Design of Structures and Machines, An International Journal, February 2010, Vol. 38, No. , Pages: 105-129, ISBN/ISSN: 1539-7742, *http://www.tandfonline.com/loi/lmbd20*.

[73]: M. Badescu and C. Mavroidis, "New Performance Indices and Workspace Analysis of Reconfigurable Hyper-Redundant Robotic Arms", Publication type: Journal, The

International Journal of Robotics Research, June 2004, Vol. 23, No. 6, Pages: 643-659, ISBN/ISSN: 0278-3649, *ijr.sagepub.com/content/23/6/643.abstract*.

[74]: V. Poppeova, J. Uricek and V. Bulej, "The Development of Mechanism Based on Hybrid Kinematic Structure", Publication type: Journal, Modern Machinery Science Journal, March 2011, Vol. , No. , Pages: 228-231, ISBN/ISSN: NA, *URL: NA*.

[75]: M. Ceccarelli, G. Carbone and E. Ottaviano, "An Optimum Design Procedure for both Serial and Parallel Manipulators", Publication type: Journal, Journal of Mechanical Engineering Science, July 2007, Vol. 221, No. 7, Pages: 829-843, ISBN/ISSN: 0954-4062, *http://pic.sagepub.com/content/221/7/829.abstract*.

[76]: M. H. Perng and L. Hsiao, "Inverse Kinematic Solutions for a Fully Parallel Robot with Singularity Robustness", Publication type: Journal, The International Journal of Robotics Research, June 1999, Vol. 18, No. 6, Pages: 575-583, ISBN/ISSN: 0278-3649, *ijr.sagepub.com/content/18/6/575.full.pdf*.

[77]: J. F. O'Brien and J. T. Wen, "Kinematic Control of Parallel Robots in the Presence of Unstable Singularities", Publication type: Conference, IEEE International Conference on Robotics and Automation (ICRA 2001), Seoul-Korea, May 2001, Pages: 354-359, ISBN/ISSN: 0-7803-6576-3, *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=932577*.

[78]: I. A. Bonev, D. Zlatanov and C. M. Gosselin, "Singularity Analysis of 3-DOF Planar Parallel Mechanisms via Screw Theory", Publication type: Journal, Journal of Mechanical Design, September 2003, Vol. 125, No. 3, Pages: 573-581, ISBN/ISSN: 1050-0472, *http://dx.doi.org/10.1115/1.1582878*.

[79]: J. E. Lloyd, "Removing the Singularities of Serial Manipulators by Transforming the Workspace", Publication type: Conference, IEEE International Conference on Robotics and Automation (ICRA 98), Leuven-Belgium, May 1998, Pages: 2935-2940, ISBN/ISSN: 0-7803-4301-8, *http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=680733&contentType=Conference+Publications*.

[80]: A. Lauletta, "The Basics of Harmonic Drive Gearing", Gear Product News, Publication type: Article, 2006, Pages: NA, *http://www.powertransmission.com/issues/0706/harmonic.htm*.

[81]: Bayside Precision Gearheads, "Get into Gear, Using Gearheads in Motion Control, Precision Planetary vs. Harmonic/Cycloidal drives", Bayside Precision Gearheads, Publication type: Article, 1997, Pages: NA, *www.electromate.com/db_support/attachments/MarkR.pdf.pdf*.

[82]: No author info., "Gears and Gear Drives", Motion System Design (MSD), Publication type: Article, 2001, Pages: 145-160, *www.emerson-ept.com/eptroot/public/schools/gears2.pdf*.

[83]: A. L. Kapelevich and E. Taye, "Self-Locking Gears: Design and Potential Applications", Publication type: Technical document, October 2010, ISBN/ISSN: 978-1-55589-992-9, *http://www.akgears.com/pdf/SELF-LOCKING%20GEAR%20-%20DESIGN%20AND%20POTENTIAL%20APPLICATIONS.pdf*.

[84]: Naoki Kanayama, "Concentric Double Axis Mechanism having Concentric Gears", Publication type: Patent, Patent number: 7752938B2, July 2010, *http://www.google.com/patents/US20080236311*.

[85]: K. Yamazoe, "Horizontal Multi-Joint Industrial Robot", Publication type: Patent, Patent number: US6593718B1, July 2003, *http://www.google.com/patents/US6593718*.

[86]: B. Bisiach, "Multi-Articulated Industrial Robot with Several Degrees of Freedom of Movement", Publication type: Patent, Patent number: 0299551A1, January 1989, *http://www.prior-ip.com/application/3234553/multi-articulated-industrial-robot-with-several-degrees-of-freedom-of-movement-*.

[87]: U. Pozzi, "Robot Arm with Multiple Links has Drive Transmitted to Links from Individual Motors through Coaxial Shafts", Publication type: Patent, Patent number: DE3734179A1, April 1989, *http://www.patent-de.com/19890427/DE3734179A1.html*.

[88]: J. L. Meriam and L. G. Kraige, "Engineering Mechanics, Dynamics", Wiley and Sons, Publication type: Book, 2007, ISBN/ISSN: 13 978-0-471-73931-9.

[89]: J. J. Craig, "Introduction to Robotics, Mechanics and Control", Pearson Education, Inc., Publication type: Book, 2005, ISBN/ISSN: 0-13-123629-6.

APPENDIX A: SciLab Code

1 Running Scilab simulation code

To run the Scilab code for the following simulations, Scilab must be installed in the path 'D:\Program Files\scilab-5.3.3'. The folder SCI_Work located in the accompanying CD must be pasted into that path (i.e. 'D:\Program Files\scilab-5.3.3\SCI_Work'). If the D drive was not available, one could install the software and work directory on another drive, but the same path structure must be maintained. When the code is executed from the main script file, one would then have to change the drive letter in all scripts as errors appear. The locations for functions and scripts were hard coded and not defined in a relative sense.

There were 3 main folders within the SCI_Work directory i.e. 'Linkage Designer', 'HKM Kinematics' and 'HKM Dynamics'. Each folder contained a number of scripts (.sce extension) and functions (.sci extension), but there were only a few main scripts that the reader would run to generate the results presented in this thesis, the others were essential but executed in the background.

The 'Linkage Designer' folder contained the code to generate the results for the primary and secondary 3 bar-slider pivot linkages. There were 3 main scripts in that folder i.e. 'LD1_3BarTrajCalLp2_2.sce', 'LD1_3BarTrajCalLp1.sce' and 'LD5_8BarTrajCalLp1.sce'.

'LD1_3BarTrajCalLp2_2.sce' calculated relevant data for the output trajectories of the primary 3 bar slider-pivot linkage, with varying pivot position. It created a 3D stack view and then highlighted 5 trajectories of interest spread evenly about the range for the pivot positions. It then compared it to a perfect circle output, and calculated the average and RMS errors with regard to that perfect circle orbit. It plotted the output angle against the input angle in 3D for all pivots points and then plotted it in 2D for selected pivot points of interest. Lastly it showed the output angle lead-lag with respect to the input angle.

'LD1_3BarTrajCalLp1.sce' calculated all data for the mid-point pivot trajectory of the primary 3 bar slider-pivot linkage. It also compared it to a perfect circle output.

'LD5_8BarTrajCalLp1.sce' generated all data for the output of the secondary 3 bar slider-pivot linkage, for multiple configurations of angle 3, the angle between the proximal and distal links. The orbits were stacked in 3D and 5 selected orbits were shown in 2D. It found the RMS and average errors. It also plotted the output angle against the input angle, and determined output angle lead-lag. These were plotted in both 3D and 2D. Measures of velocity and acceleration were also plotted in 2D and 3D.

The 'HKM Kinematics' and 'HKM Dynamics' folders contained all code for machine level simulation. The 'HKM Kinematics' folder also contained 3 main scripts i.e. 'Forward_Kinematics.sce', 'Inverse_Kinematics.sce' and 'Path_Follow_Select.sce'.

'Forward_Kinematics.sce' implemented the machine level forward kinematics solution. It displayed a skeleton graphic of the machine model and allowed the user to select one of the 6 main angles to control the robot in the forward sense. This was done by pressing a number from 1 to 6 on the keyboard, to select the angle. The up-down and left-right arrow keys were then used to increase or decrease that selected angle. It displayed the angle values and end-effector coordinates, and updated it as the user changed any of the robot angles. The user could also select how the coordinate frames were displayed by pressing the H key.

'Inverse_Kinematics.sce' calculated the joint angles for a given pose. It called the forward kinematics script to generate the graphics if it didn't already exist. A particular desired end-effector pose was then defined through the forward kinematics GUI interface. At this point the robot joint angles were already known as they were selected, this gave the inverse kinematics script a reachable robot pose. The inverse kinematics script was then run again, it took the end-effector pose and calculated all joint angle solution sets that would give the robot that particular pose. The user could view all solution sets in the Scilab terminal by setting the variable 'Display_Check' to 1 close to the end of the script file. The angles displayed in the GUI had to equal one set of the 8 calculated solutions.

'Path_Follow_Select.sce' displayed an animation of path following and allowed the user to select a path (square, circle, circle with changing orientation where the end-effector pointed to a specific constant location, and a text output where the end-effector traced out 'hello world' on the screen). The inverse kinematics solved for the joint angles as the end-effector was moved with the required orientation along the selected path, and the forward kinematics script was called to animate the motion at each point. The path was selected by setting the variable 'This_Path_Select' to a number between 1 and 5 inclusive.

The 'HKM Dynamics' folder contains 3 main scripts i.e. 'Dyn_Path_Select.sce', 'Multi_Path_ECalc.sce' and 'Multi_CirPath_ECalc.sce'.

'Dyn_Path_Select.sce' allowed the user to select a vertical, horizontal or radial path (2 for each, centred on different points). The paths were created and the joint angles were solved and animated for each pose in the path that used a pre-configured orientation. Once the angles were obtained the time vector was created using the duration set through the variable 'Total_Play_Time'. The velocities and accelerations were then calculated for each pose. The inverse dynamics script was then called which created all 4 dynamic models (defined inertia matrices, COMs, etc.) and then solved for the joint torques. Once that was complete a measure of the energy used was determined from the angular work done at the joints.

'Multi_Path_ECalc.sce' calculated the energy used for multiple horizontal, vertical and radial paths, for all models so that they could be compared. The number of paths was set using the 'num_paths' variable. That defined the number of lines between the 2 boundaries, i.e. the 2 vertical, horizontal or radial line paths from the previous script. For a low number of paths the user could comment out the 'global Dont_show;' and 'Dont_show = 1;' lines, to view the animation as the script moved through the number of paths set.

'Multi_CirPath_ECalc.sce' calculated the energy used for each of the 3 main joints responsible for end-effector spatial positioning. It singled out each joint (other joint angles were kept constant) and calculated the joint energy for best and worse case arm mass distribution. For example the worst case mass distribution for joint 1 (control about the vertical axis) would be when the arm was at full stretch and horizontal, the best case would be when the arm pointed vertically up. The joints were also rotated through their full range of motion and experienced large joint angle displacements, unlike in the straight line paths.

2 Linkage Design

2.1 **File name:** LD1_3BarTrajCalLp2_2.sce

Purpose: This routine calculates relevant data for output trajectories of the primary 3 bar slider-pivot linkage, with varying pivot position. It creates a 3D stack view and then highlights 5 trajectories of interest spread evenly about the range for the pivot positions. It then compares it to a perfect circle output, and calculates the average and RMS errors with regard to that perfect circle orbit. It plots the output angle against the input angle in 3D for all pivots points and then plots it in 2D for selected pivot points of interest. Lastly it shows the output angle lead-lag with respect to the input angle.

```
clc;
xdel(winsid()); //Destroy all existing figures
clearglobal();
clear;
lines(0); //disables vertical paging


//Returns screen size in pixels, on this PC it s 1920 by 1200
ScreenSizePX = get(0, "screensize_px");
```

```
//Returns screen size in points, on this PC it s 960 by 600
//ScreenSizePT = get(0, "screensize_pt");

L1_len = 8;
L2_len = 1;

DVindex = 1;
DataView(DVindex) = figure(DVindex); // Create figure having figure_id==1
DataView(DVindex).figure_name = "Linkage Designer"
DataView(DVindex).info_message = ""
Win_Width = ScreenSizePX(3);
Win_Height = ScreenSizePX(4);
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
DataView(DVindex).background = 8; //White

drawlater();
subplot(2,2,1)
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
tmpBnd = (2*L1_len + 0.25*L2_len);
DataView(DVindex).children(1).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
//[xmin ymin; xmax ymax]->must be 2 dimensional

DataView(DVindex).children(1).grid = [0 0]; //Draws it in black
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X Axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y Axis";
toolbar(DataView(DVindex),'off');


AngIndex = 1;
Angles = [0 0 0 0 0 0];



//Link1
Lnks = list(1);
x = [0, L1_len/2, L1_len];
y = [0, 0, 0];
xpoly(x, y);
Lnks($) = get("hdl");
//get handle on current entity (here the polyline entity)

//Link 2 Set
SubLnks2 = list(1);
x = [0, L2_len];
y = [0, 0];
xpoly(x, y);
SubLnks2($) = get("hdl");
//get handle on current entity (here the polyline entity)

x = [L2_len, L2_len+L1_len];
y = [0, 0];
xpoly(x, y);
SubLnks2($+1) = get("hdl");
//get handle on current entity (here the polyline entity)

Lnks($+1) = SubLnks2;

Colours = [1 2 5];//1-> Black; 2-> Blue; 5-> Red

End_i = size(Lnks);
//disp(End_i);
for i = 1:End_i(1)
    End_j = size(Lnks(i));
    //disp(End_j);
    for j = 1:End_j(1)
        Lnks(i)(j).thickness=3;
        Lnks(i)(j).mark_style=9;
        Lnks(i)(j).mark_mode="on";
        Lnks(i)(j).polyline_style=1;
        Lnks(i)(j).foreground = Colours(i);
    end
end
```

```
DrawEntities = %t;

Data3D = list(1);


//Pivot positions
// Use Powers of 2 as we use this for the colourmap as well
Pivot_Positions = 16;
if (Pivot_Positions > 2)
    //Too much data, takes too long to draw and calculate
    DrawEntities = %f;
end

Pivot_Length_Inc = (L1_len - L2_len)/(Pivot_Positions + 1);
Max_Dev = -100;
Min_Dev = 100;
MaxErr = -100;
MinErr = 100;

AngInc = 15; //Sets the angle increment

//Creates a vector with the angular values in it....

for TempAng1 = 0 : AngInc : 360
    Input_Ang($+1) = TempAng1; //This results in a column vector
end
Input_Ang = Input_Ang'; //Make it a row vector


Height = 0;
i = 1;
// We adjust the pivot position here and trace out a contour and then stack them vertically
for Pivot_Point = (L2_len + Pivot_Length_Inc) : Pivot_Length_Inc : (L1_len - Pivot_Length_Inc)
    j = 1;

    Piv_Pos_Vec(i) = Pivot_Point; //Save Pivot Positions

    //Need to reset the Max limits, and Error variables
    Max_X = -100;
    Min_X = 100;
    Max_Y = -100;
    Min_Y = 100;
    SqErrorSum = 0;
    ErrorSum = 0;

    for TempAng1 = 0 : AngInc : 360

        drawlater();
        Angles = [0, TempAng1, 0, 0, 0, 0];
        //LINK 1
        Fraction = Pivot_Point/L1_len;
        x1 = cosd(Angles(1))*[0, Fraction*L1_len, L1_len];
        y1 = sind(Angles(1))*[0, Fraction*L1_len, L1_len];
        z1 = [0 0 0];
        xyz = [x1; y1; z1];
        set(Lnks(1), "data",  [x1' y1']);

        //LINK SET 2
        x21 = cosd(Angles(2))*[0 L2_len];
        y21 = sind(Angles(2))*[0 L2_len];
        z21 = [0 0];
        set(Lnks(2)(1), "data",  [x21' y21']);

        dx21 = x1(2) - x21(2);
        dy21 = y1(2) - y21(2);
        dl21 = (dx21^2 + dy21^2)^0.5;
        dx = dx21/dl21;
        dy = dy21/dl21;

        x22 = L1_len*[0 dx] + [x21(2) x21(2)];
        //translate to the end of link 2,1
        y22 = L1_len*[0 dy] + [y21(2) y21(2)];
        z22 = [0 0];
        set(Lnks(2)(2), "data",  [x22' y22']);

        Orb_Radius = ((x22(2) - x1(3))^2 + (y22(2) - y1(3))^2 )^0.5; //Distance
```

```
        if (Orb_Radius > Max_Dev)
            Max_Dev = Orb_Radius;
        elseif (Orb_Radius < Min_Dev)
            Min_Dev = Orb_Radius;
        end

        // This computes the square and average errors when compared to a
        // perfect circle orbit
        dl = Orb_Radius - L2_len;
        SqErrorSum = SqErrorSum + dl^2;
        ErrorSum = ErrorSum + dl;


        // Finding Max_X and Min_X
        if (x22(2) > Max_X) // Finding Max_X
            Max_X = x22(2);
        elseif (x22(2) < Min_X) // Finding Min_X
            Min_X = x22(2);
        end

        // Finding Max_Y and Min_Y
        if (y22(2) > Max_Y) // Finding Max_Y
            Max_Y = y22(2);
        elseif (y22(2) < Min_Y) // Finding Min_Y
            Min_Y = y22(2);
        end

        if (DrawEntities == %t)
            drawnow();
        end
        ContourLine(:, j) = [x22(2)-x1(3); y22(2)-y1(3); Pivot_Point]
        // Height, Centers the contour line on  (0,0)
        j = j+1;
    end

    Piv_Point_Locs(i) = Pivot_Point;
    Delta_X(i) = Max_X - Min_X;
    Delta_Y(i) = Max_Y - Min_Y;
    RtMeanSqError(i) = (SqErrorSum/j)^0.5;
    MeanError(i) = ErrorSum/j;
    //Need to find Y-limits for plot
    if (RtMeanSqError(i) > MaxErr) then
        MaxErr = RtMeanSqError(i)
    end
    if (MeanError(i) > MaxErr) then
        MaxErr = MeanError(i)
    end
    if (RtMeanSqError(i) < MinErr) then
        MinErr = RtMeanSqError(i)
    end
    if (MeanError(i) < MinErr) then
        MinErr = MeanError(i)
    end

    Data3D($+1) = ContourLine;
    Height = Height + 1
    i=i+1;

end
drawnow();



//*********************************************************************
// PLOTTING DATA
//FIGURE 2: 3D contour plot of trajectories for multiple pivot positions
drawlater();
subplot(2,2,2);

Data3D(1) = null(); //Kill first point as its just the 1 initialised
//earlier and not the list data
End_i = size(Data3D)
for i = 1 : End_i(1);
    //Creating data set for Facet, and closing the contour by adding
    //redundant start point at end
    X(i, :) = Data3D(i)(1,:); //All x data
    Y(i, :) = Data3D(i)(2,:); //All y data
```

```
    Z(i, :) = Data3D(i)(3,:); //All z data
end

// 3D SURFACE PLOT
surf(X, Y, Z);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1; // All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3; // interpolated shading mode. The default is 4 ('flat' mode) for surf


//Set the figure properties after you've drawn the last figure entity

DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "b. 3D Stacking of Trajectory Curves";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y axis";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = "Height (Z axis)";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0];
// Displays grid in black, negative numbers hide it
tmpBnd = 1.05*Max_Dev;
DataView(DVindex).children(1).data_bounds = ...
[-tmpBnd, -tmpBnd, -1; tmpBnd, tmpBnd, (Height + 2)];
//[xmin ymin zmin; xmax ymax zmax]-> for 3D

DataView(DVindex).children(1).isoview = "on"; //"on" Makes square axis look square
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off";
drawnow();



//*********************************************************************
//FIGURE 3: 2D plot of selected contours
//DVindex = 1;
subplot(2,2,3);
drawlater();
//DataView(DVindex) = figure(3);
//DataView(DVindex).figure_name = "Data Graph 2"
//DataView(DVindex).info_message = ""
//DataView(DVindex).figure_position = [-1, -1];
// [0,0] Displays grid in black, negative numbers hide it
//DataView(DVindex).figure_size = [Win_Width, Win_Height];


//Index selection of data in Data3D
Index_2D_Graph = [1, round(0.25*Pivot_Positions), round(0.5*Pivot_Positions), ...
round(0.75*Pivot_Positions), Pivot_Positions];

for i = 1 : 5
    px = Data3D(Index_2D_Graph(i))(1,:); //All x data
    py = Data3D(Index_2D_Graph(i))(2,:); //All y data
    xpoly(px, py);
    Orbit(i) = get("hdl"); //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_mode = "on";
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_Graph(i);
    //maps the colour from the colour map to match with the 3D graph
end

// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
```

```
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"c. Top View Illustrating 5 Trajectories of Interest"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y axis";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [0 0]; //Show grid, neg numbers hide it
tmpBnd = 1.05*Max_Dev;
DataView(DVindex).children(1).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview = "on"; //"on" Makes square axis look square
DataView(DVindex).children(1).hidden_axis_color = 0;

Legd_num = 1;
Legd(Legd_num) = legend(DataView(DVindex).children(1), ['Pivot Position = 1'; ...
'Pivot Position = 2'; 'Pivot Position = 3'; 'Pivot Position = 4'; ...
'Pivot Position = 5']);
Legd(Legd_num).font_size = 5;
Legd(Legd_num).font_style = 5;
Legd(Legd_num).visible = 'on';
Legd(Legd_num).foreground = -1; //Black
Legd(Legd_num).background = -2; //White
Legd(Legd_num).font_style = 6; //Sans Serif close to arial
drawnow();



//*********************************************************************
//FIGURE 4: 2D plot of Delta_x (Max_x-Min_x) and Delta_y (Max_y - Min_y)
subplot(2,2,4);
drawlater();

xpoly(Piv_Point_Locs, Delta_X);
ent_num = 1;
//get handle on current entity (here the polyline entity)
Delta_XY(ent_num) = get("hdl");
Delta_XY(ent_num).thickness = 3;
Delta_XY(ent_num).polyline_style = 1;
Delta_XY(ent_num).line_style = 1; // 1-Solid; 3-Dash dot style
Delta_XY(ent_num).mark_mode = "on";
Delta_XY(ent_num).mark_style = 0;
Delta_XY(ent_num).mark_size_unit = "point";
Delta_XY(ent_num).mark_size = 3;
Delta_XY(ent_num).foreground = 1; //Red

xpoly(Piv_Point_Locs, Delta_Y);
ent_num = 2;
//get handle on current entity (here the polyline entity)
Delta_XY(ent_num) = get("hdl");
Delta_XY(ent_num).thickness = 3;
Delta_XY(ent_num).polyline_style = 1;
Delta_XY(ent_num).line_style = 1; // 1-Solid; 3-Dash dot style
Delta_XY(ent_num).mark_mode = "on";
Delta_XY(ent_num).mark_style = 0;
Delta_XY(ent_num).mark_size_unit = "point";
Delta_XY(ent_num).mark_size = 3;
Delta_XY(ent_num).foreground = Pivot_Positions; //purple or blue

// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"d. Dimensions of Rectangles Containing the Trajectories"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Pivot Position";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "$\Delta $";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
```

```
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [0 0]; //Show grid, neg numbers hide it

DataView(DVindex).children(1).data_bounds = [L2_len, 0; L1_len, 1.95*tmpBnd];
// [xmin ymin; xmax ymax]-> for 2D
// L1_len = 8; L2_len = 1;

DataView(DVindex).children(1).isoview = "off"; //"on" Makes square axis look square
DataView(DVindex).children(1).hidden_axis_color = 0;

Legd_num = 2;
Legd(Legd_num) = legend(DataView(DVindex).children(1), ...
["$\Delta X_{(Max-Min)}$"; "$\Delta Y_{(Max-Min)}$"]);
Legd(Legd_num).font_size = 5;
Legd(Legd_num).font_style = 5;
Legd(Legd_num).visible = 'on';
Legd(Legd_num).foreground = -1; //Black
Legd(Legd_num).background = -2; //White
Legd(Legd_num).font_style = 6; //Sans Serif close to arial
drawnow();



//*********************************************************************
// NEED NEW FIGURE
//FIGURE 5: 2D plot of mean square errors and average errors
DVindex = DVindex + 1;
DataView(DVindex) = figure(DVindex);
DataView(DVindex).figure_name = "Data Graph 2"
DataView(DVindex).info_message = ""
DataView(DVindex).figure_position = [-1, -1];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
subplot(2,2,1);
drawlater();

xpoly(Piv_Point_Locs, RtMeanSqError);
ent_num = 1;
Delta_XY(ent_num) = get("hdl");
//get handle on current entity (here the polyline entity)
Delta_XY(ent_num).thickness = 3;
Delta_XY(ent_num).polyline_style = 1;
Delta_XY(ent_num).line_style = 1; // 1-Solid; 3-Dash dot style
Delta_XY(ent_num).mark_mode = "on";
Delta_XY(ent_num).mark_style = 0;
Delta_XY(ent_num).mark_size_unit = "point";
Delta_XY(ent_num).mark_size = 3;
Delta_XY(ent_num).foreground = 1; //Red

xpoly(Piv_Point_Locs, MeanError);
ent_num = 2;
Delta_XY(ent_num) = get("hdl");
//get handle on current entity (here the polyline entity)
Delta_XY(ent_num).thickness = 3;
Delta_XY(ent_num).polyline_style = 1;
Delta_XY(ent_num).line_style = 1; // 1-Solid; 3-Dash dot style
Delta_XY(ent_num).mark_mode = "on";
Delta_XY(ent_num).mark_style = 0;
Delta_XY(ent_num).mark_size_unit = "point";
Delta_XY(ent_num).mark_size = 3;
Delta_XY(ent_num).foreground = Pivot_Positions; //purple or blue

// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"e. Mean Square and Average Errors for the Trajectories"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Pivot Position";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Error";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [0 0]; //Show grid, neg numbers hide it
```

```
DataView(DVindex).children(1).data_bounds = [L2_len, 0.95*MinErr; L1_len, 1.05*MaxErr];
// [xmin ymin; xmax ymax]-> for 2D
// L1_len = 8; L2_len = 1;

DataView(DVindex).children(1).isoview = "off"; //"on" Makes square axis look square
DataView(DVindex).children(1).hidden_axis_color = 0;

Legd_num = 3;
Legd(Legd_num) = legend(DataView(DVindex).children(1), ...
["Root Mean Square Error"; "Average Error"]);
Legd(Legd_num).font_size = 5;
Legd(Legd_num).font_style = 5;
Legd(Legd_num).visible = 'on';
Legd(Legd_num).foreground = -1; //Black
Legd(Legd_num).background = -2; //White
Legd(Legd_num).font_style = 6; //Sans Serif close to arial
drawnow();


//*********************************************************************
//FIGURE 6: 3D plot of output angle versus input angle
subplot(2,2,2);
drawlater();
DataView(DVindex).figure_name = "Output Angle VS Input Angle"
DataView(DVindex).info_message = ""
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];

//Now we need to generate the data, we use the contour curves in Data3D
//Need to compute the output angle with respect to the input angle
// The first point was killed on line 255
// X,Y and Z matrices were created on line 230

End_indices = size(X);
End_i = End_indices(2); //Columns;
End_j = End_indices(1); //Rows;

Output_Ang_X2 = zeros(End_j, End_i);
Output_Ang_Y2 = zeros(End_j, End_i);
Output_Ang_Z2 = zeros(End_j, End_i);

Angular_Offset_X3 = zeros(End_j, End_i);
Angular_Offset_Y3 = zeros(End_j, End_i);
Angular_Offset_Z3 = zeros(End_j, End_i);

Output_Ang = zeros(1, End_i);
Angular_Offset = zeros(1, End_i);

for j = 1 : End_j
    for i = 1 : End_i
        Len1 = (X(j,i)^2 + Y(j,i)^2)^0.5;
        // These X, Y values are with respect to the orbit centre
        Calc_Ang1 = acosd(X(j,i)/Len1);
        Calc_Ang2 = asind(Y(j,i)/Len1);

        // We need to correct the angle as asind range is [-90 90], and acosd
        // range is [0 180]. In this case acosd yields the correct value but
        // incorrect sign from 0 to 180 so we just need to negate it, in this
        // range the results for asind is negative. For the correct angle in
        // the remaining range, we need to subtract 360 from acosd (in this
        // range asind is positive).

        if (Calc_Ang2 <= 0) then // asind is negative or 0
            Output_Ang(i) = -Calc_Ang1;
        else
            Output_Ang(i) = Calc_Ang1 - 360;
        end

        // Remember the output angle moves in the opposite direction to the input angle
        // If there was a perfect 1 to -1 ratio then the sum of the output angle and the
        // input angle would be 0.
        Angular_Offset(i) = Output_Ang(i) + Input_Ang(i);
        // The difference from a perfect 1 to -1 trajectory.
    end
    Output_Ang($) = -360; // Correction to last angle
    Angular_Offset($) = Output_Ang($) + Input_Ang($); // correction to last angle
```

```
    Output_Ang_X2(j, :) = Output_Ang;
    Output_Ang_Y2(j, :) = Input_Ang;
    Output_Ang_Z2(j, :) = Piv_Pos_Vec(j)*ones(1, End_i);

    Angular_Offset_X3(j, :) = Angular_Offset;
    Angular_Offset_Y3(j, :) = Input_Ang;
    Angular_Offset_Z3(j, :) = Piv_Pos_Vec(j)*ones(1, End_i);
end

// 3D SURFACE PLOT
surf(Output_Ang_X2, Output_Ang_Y2, Output_Ang_Z2);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;
// All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3; // interpolated shading mode. The default is 4 ('flat' mode) for surf

// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"f. Output Angles VS Input Angles for the Trajectories"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Output Angle"; //"Pivot Positions";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Input Angle";
DataView(DVindex).children(1).z_label.font_size = 5;
DataView(DVindex).children(1).z_label.text = "Pivot Positions"; //"Output Angle";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0]; //Show grid, neg numbers hide it
DataView(DVindex).children(1).data_bounds = [-360, 0, 0; 0, 360, Pivot_Positions+1];
//[xmin ymin zmin; xmax ymax zmax]-> for 3D
DataView(DVindex).children(1).isoview = "on"; //"on" Makes square axis look square
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off";
drawnow();



//**************************************************************************
//FIGURE 7: 2D plot of selected output angle vs input angle
subplot(2,2,3);
drawlater();

for i = 1 : 5
    xpoly(Input_Ang, Output_Ang_X2(Index_2D_Graph(i), :));
    Output_Ang_Crv(i) = get("hdl");
    //get handle on current entity (here the polyline entity)
    Output_Ang_Crv(i).thickness = 3;
    Output_Ang_Crv(i).polyline_style = 1;
    Output_Ang_Crv(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Output_Ang_Crv(i).mark_mode = "on";
    Output_Ang_Crv(i).mark_style = 0;
    Output_Ang_Crv(i).mark_size_unit = "point";
    Output_Ang_Crv(i).mark_size = 3;
    Output_Ang_Crv(i).foreground = Index_2D_Graph(i);
    //maps the colour from the colour map to match with the 3D graph
end


// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"g. Top View of Output Angles VS Input Angles for Selected Trajectories"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output Angle";
DataView(DVindex).children(1).z_label.font_angle = 270;
```

```
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [0 0]; //Show grid, neg numbers hide it
DataView(DVindex).children(1).data_bounds = [0, -360; 360, 0];
// [xmin ymin; xmax ymax]-> for 2D
// L1_len = 8; L2_len = 1;
DataView(DVindex).children(1).isoview = "off"; //"on" Makes square axis look square
DataView(DVindex).children(1).hidden_axis_color = 0;
drawnow();



//*********************************************************************
// FIGURE 8: 3D plot of output angle lead-lag
subplot(2,2,4);
drawlater();

// 3D SURFACE PLOT
surf(Angular_Offset_X3, Angular_Offset_Y3, Angular_Offset_Z3);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1; // All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3; // interpolated shading mode. The default is 4 ('flat' mode) for surf

// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"h. Ouput angle lead-lag for all Trajectories"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Ouput angle -Input angle";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Input Angle";
DataView(DVindex).children(1).z_label.font_size = 5;
DataView(DVindex).children(1).z_label.text = "Pivot Positions";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0]; //Show grid, neg numbers hide it
DataView(DVindex).children(1).data_bounds = [-80, 0, 0; 80, 360, Pivot_Positions+1];
//[xmin ymin zmin; xmax ymax zmax]-> for 3D
DataView(DVindex).children(1).isoview = "on"; //"on" Makes square axis look square
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off";
drawnow();


//*********************************************************************
// NEED NEW FIGURE WINDOW
// FIGURE 9:2D plot of output angle vs input angle for selected angle 3
DVindex = DVindex + 1;
DataView(DVindex) = figure(DVindex);
DataView(DVindex).figure_name = "Data Graph 3"
DataView(DVindex).info_message = ""
DataView(DVindex).figure_position = [-1, -1];
DataView(DVindex).figure_size = [Win_Width, Win_Height];

subplot(2,2,1);
drawlater();

for i = 1 : 5
    xpoly(Input_Ang, Angular_Offset_X3(Index_2D_Graph(i), :));
    Angular_Offset_Crv(i) = get("hdl");
    //get handle on current entity (here the polyline entity)
    Angular_Offset_Crv(i).thickness = 3;
    Angular_Offset_Crv(i).polyline_style = 1;
    Angular_Offset_Crv(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Angular_Offset_Crv(i).mark_mode = "on";
    Angular_Offset_Crv(i).mark_style = 0;
    Angular_Offset_Crv(i).mark_size_unit = "point";
    Angular_Offset_Crv(i).mark_size = 3;
    Angular_Offset_Crv(i).foreground = Index_2D_Graph(i);
```

```
    //maps the colour from the colour map to match with the 3D graph
end

// Figure showing 2 lines only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
"Ouput angle lead-lag for Selected Trajectories";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output angle - Input angle";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(Pivot_Positions);
//Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [0 0];
//Show grid, neg numbers hide it
DataView(DVindex).children(1).data_bounds = [0, -85; 360, 85];
// [xmin ymin; xmax ymax]-> for 2D
// L1_len = 8; L2_len = 1;
DataView(DVindex).children(1).isoview = "off";
//"on" Makes square axis look square
DataView(DVindex).children(1).hidden_axis_color = 0;


Legd_num = 5;
Legd(Legd_num) = legend(DataView(DVindex).children(1), ['Pivot Position = 1'; ...
'Pivot Position = 2'; 'Pivot Position = 3'; ...
'Pivot Position = 4'; 'Pivot Position = 5']);
Legd(Legd_num).font_size = 5;
Legd(Legd_num).font_style = 5;
Legd(Legd_num).visible = 'on';
Legd(Legd_num).foreground = -1; //Black
Legd(Legd_num).background = -2; //White
Legd(Legd_num).font_style = 6; //Sans Serif close to arial
drawnow();
```

2.2 **File name:** LD1_3BarTrajCalLp1.sce

Purpose: This routine calculates all relevant data for the mid-point pivot trajectory of the primary 3 bar slider-pivot linkage. It also compares it to a perfect circle output.

```
clc;
clearglobal();
clear;
clearglobal();
clear;
lines(0); //disables vertical paging
xdel(winsid()); //Destroy all existing figures


//Returns screen size in pixels, on this PC it s 1920 by 1200
ScreenSizePX = get(0, "screensize_px");
//Returns screen size in points, on this PC it s 960 by 600
//ScreenSizePT = get(0, "screensize_pt");

L1_len = 8;
L2_len = 1;
L1_len_2 = L1_len;//L1_len + 1.5*L2_len;

AngIndex = 1;
Angles = [0 0 0 0 0 0];
AngInc = 5;

LinkageTester = figure(1); // Create figure having figure_id==1
LinkageTester.figure_name = "Linkage Designer";
```

```
LinkageTester.info_message = "";
Win_Width = ScreenSizePX(3);
Win_Height = ScreenSizePX(4);
LinkageTester.figure_position = [0, 0];
LinkageTester.figure_size = [Win_Width, Win_Height];
LinkageTester.children($).axes_visible = ["on","on","off"];
LinkageTester.background = 8; //White
tmpBnd = (L1_len + L2_len);
//[xmin ymin; xmax ymax]->must be 2 dimensional
LinkageTester.children($).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
LinkageTester.children($).grid = [1 1];
LinkageTester.children($).isoview="on";
LinkageTester.children($).font_size = 5;
LinkageTester.children($).x_label.font_size = 5;
LinkageTester.children($).x_label.text = "X Axis";
LinkageTester.children($).y_label.font_size = 5
LinkageTester.children($).y_label.text = "Y Axis";
//toolbar(LinkageTester,'off');


Lnks = list(1);
x = cosd(Angles(1))*[0, L1_len/2, L1_len];
y = sind(Angles(1))*[0, L1_len/2, L1_len];
xpoly(x, y);
Lnks($) = get("hdl");
//get handle on current entity (here the polyline entity)

SubLnks2 = list(1);
x = cosd(Angles(2))*[0, L2_len];
y = sind(Angles(2))*[0, L2_len];
xpoly(x, y);
SubLnks2($) = get("hdl");
//get handle on current entity (here the polyline entity)

x = cosd(Angles(2))*[L1_len, L2_len+L1_len];
y = sind(Angles(2))*[L1_len, L2_len+L1_len];
xpoly(x, y);
SubLnks2($+1) = get("hdl");
//get handle on current entity (here the polyline entity)

x = cosd(Angles(2))*[L1_len, 2*L1_len];
y = sind(Angles(2))*[L1_len, 2*L1_len];
xpoly(x, y);
SubLnks2($+1) = get("hdl");
//get handle on current entity (here the polyline entity)

Lnks($+1) = SubLnks2;



Colours = [1 2 5];//1-> Black; 2-> Blue; 5-> Red

End_i = size(Lnks);
//disp(End_i);
for i = 1:End_i(1)
  End_j = size(Lnks(i));
  //disp(End_j);
  for j = 1:End_j(1)
    Lnks(i)(j).thickness=3;
    Lnks(i)(j).mark_style=9;
    Lnks(i)(j).mark_mode="on";
    Lnks(i)(j).polyline_style=1;
    Lnks(i)(j).foreground = Colours(i);
  end
end

Output_Point_Traj = [];
MinTrajDev = 100;//Deliberately set the first value high.
MaxTrajDev = -100;//Deliberately set the first value low.
SqError1 = 0;//going to find the mean square error
AveError1 = 0;//going to find the mean error
NumElem1 = 0;


for TempAng = 0 : AngInc : 360
  Input_Ang($+1) = TempAng;
```

```
  Angles(2)= TempAng;
  drawlater();
  //LINK 1
  x1 = cosd(Angles(1))*[0 L1_len/2 L1_len];
  y1 = sind(Angles(1))*[0 L1_len/2 L1_len];
  z1 = [0 0 0];
  xyz = [x1; y1; z1];
  xtemp = xyz(1,:);
  ytemp = xyz(2,:);
  set(Lnks(1), "data",  [xtemp' ytemp']);

  //LINK 2 set
  x21 = cosd(Angles(2))*[0 L2_len];
  y21 = sind(Angles(2))*[0 L2_len];
  z21 = [0 0];
  set(Lnks(2)(1), "data",  [x21' y21']);

  // This part finds the unit vector from the end of the driving link to the
  // midpoint of the stationary link (just direction).
  dx = x1(2) - x21(2);
  dy = y1(2) - y21(2);
  dl = (dx^2 + dy^2)^0.5;
  dx = dx/dl;
  dy = dy/dl;

  // This code increases the magnitude of that unit vector to be the length
  // of the stationary link, and translates it to the end of the driving link
  x22 = L1_len_2*[0 dx] + [x21(2) x21(2)];
  y22 = L1_len_2*[0 dy] + [y21(2) y21(2)];
  z22 = [0 0];
  xyz = [x22; y22; z22];
  xtemp = xyz(1,:);
  ytemp = xyz(2,:);
  set(Lnks(2)(2), "data",  [xtemp' ytemp']);

  // This code adds the upper arm link thats being driven
  dx = x22(2) - x1(3);
  dy = y22(2) - y1(3);
  dl = (dx^2 + dy^2)^0.5;
  dx = dx/dl;
  dy = dy/dl;
  x23 = L1_len*[0 dx] + [x1(3) x1(3)];
  y23 = L1_len*[0 dy] + [y1(3) y1(3)];
  z23 = [0 0];
  xyz = [x23; y23; z23];
  xtemp = xyz(1,:);
  ytemp = xyz(2,:);
  set(Lnks(2)(3), "data",  [xtemp' ytemp']);


  // This code determines the length deviation from the perfect circle
  // orbit.
  //Tracking mid-point of line
  COM(:, j) = [0.5*(x22(1)+x22(2)); 0.5*(y22(1)+y22(2))];
  Output_Point_Traj(:,$+1) = [x22(2); y22(2)];
  //Used for angular Velocity
  Output_Pnt_Transl(:, $+1) = [x22(2); y22(2)] - [L1_len; 0];
  dx = x22(2) - L1_len;
  dy = y22(2);
  dl = (dx^2 + dy^2)^0.5 - L2_len;
  dl_vec($+1) = dl; //deviation from length L2_len
  dl_norm_vec($+1) = dl/L2_len; //Normalised on L2_len

   //Finding the minimum and maximum deviation from the orbit
  if (dl > MaxTrajDev) then
    MaxTrajDev = dl;
  elseif (dl < MinTrajDev)
    MinTrajDev = dl;
  end

  SqError1 = SqError1 + dl^2;
  AveError1 = AveError1 + dl;
  NumElem1 = NumElem1 + 1;

  //drawnow();
end
```

```
mprintf('\nMax Inner Deviation: %f', MinTrajDev);
mprintf('\nMax Inner Deviation Normalized: %f', MinTrajDev/L2_len);
mprintf('\nMax Outer Deviation: %f', MaxTrajDev);
mprintf('\nMax Outer Deviation Normalized: %f', MaxTrajDev/L2_len);
mprintf('\nAverage Error: %f', AveError1/NumElem1);
mprintf('\nAverage Error Normalised: %f', AveError1/(NumElem1*L2_len));
mprintf('\nSquare Mean Error: %f', SqError1/(NumElem1));
mprintf('\nSquare Mean Error Normalised: %f\n', SqError1/(NumElem1*L2_len*L2_len));


Output_Point_Traj(:,1)=[];//kill the first point because its [0;0]
x = Output_Point_Traj(1,:);
y = Output_Point_Traj(2,:);
xpoly(x, y);
Orbit1 = get("hdl"); //get handle on current entity (here the polyline entity)
Orbit1.thickness = 2;
Orbit1.mark_style = 9;
Orbit1.mark_mode = "off";
Orbit1.polyline_style = 1;
Orbit1.line_style = 3; //Dash dot style
Orbit1.foreground = Colours(3); //red


//Perfect_Circ(:,1)=[];//kill the first point because its [0;0]
sz_OP_Traj = size(Output_Point_Traj);
d_pc_ang = 360/(sz_OP_Traj(2) - 1);
Perfect_Circ = [];
for pc_ang = 0 : d_pc_ang : 360
    Perfect_Circ = [Perfect_Circ, [cosd(pc_ang); sind(pc_ang)]];
end
x = Perfect_Circ(1,:)+ L1_len;
y = Perfect_Circ(2,:);


xpoly(x, y)
Circ = get("hdl"); //get handle on current entity (here the polyline entity)
Circ.thickness = 2;
Circ.mark_style = 9;
Circ.mark_mode = "off";
Circ.polyline_style = 1;
Circ.line_style = 3; //Dash dot style
Circ.foreground = Colours(2); //blue

DVindex = 1;
//FIGURE 2 **********************************
DataView(DVindex) = figure(DVindex+1);
DataView(DVindex).figure_name = "Data Graphs";
DataView(DVindex).info_message = "Need to Print a message here";
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
DataView(DVindex).background = 8; //White

subplot(2,2,1);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).auto_scale="on";
tmpBnd = 1.25*L2_len;
//[xmin ymin; xmax ymax]->must be 2 dimensional
DataView(DVindex).children(1).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "X Axis";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Y Axis";

x = Output_Point_Traj(1,:) - L1_len;
y = Output_Point_Traj(2,:);
xpoly(x, y);
Orbit1 = get("hdl"); //get handle on current entity (here the polyline entity)
Orbit1.thickness = 3;
Orbit1.mark_style = 9;
Orbit1.mark_mode = "off";
Orbit1.polyline_style = 1;
Orbit1.line_style = 3; //Dash dot style
Orbit1.foreground = Colours(3); //red
```

```
LO_OutTraj_x = x';
LO_OutTraj_y = y';

x = Perfect_Circ(1,:);
y = Perfect_Circ(2,:);
xpoly(x, y);
Circ = get("hdl"); //get handle on current entity (here the polyline entity)
Circ.thickness = 3;
Circ.mark_style = 9;
Circ.mark_mode = "off";
Circ.polyline_style = 1;
Circ.line_style = 1; //Solid style
Circ.foreground = Colours(2); //blue

LO_OutTraj_x = [LO_OutTraj_x, x'];
LO_OutTraj_y = [LO_OutTraj_y, y'];
drawnow();


//FIGURE 3 **********************************
subplot(2,2,2);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Offset Length";

plot(Input_Ang, dl_vec);
//get handle on line drawn
DataLine2 = DataView(DVindex).children(1).children(1).children(1);
DataLine2.thickness = 3;
DataLine2.mark_style = 9;
DataLine2.mark_mode = "off";
DataLine2.polyline_style = 1;
DataLine2.line_style = 1; //solid
DataLine2.foreground = Colours(2); //blue
drawnow();


//FIGURE 4 **********************************
subplot(2,2,3);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Normalised Offset Length";

plot(Input_Ang, dl_norm_vec);
//get handle on line drawn
DataLine3 = DataView(DVindex).children(1).children(1).children(1);
DataLine3.thickness = 3;
DataLine3.mark_style = 9;
DataLine3.mark_mode = "off";
DataLine3.polyline_style = 1;
DataLine3.line_style = 1; //solid
DataLine3.foreground = Colours(2); //blue

LO_Norm_Offsets = [Input_Ang, dl_norm_vec];

drawnow();

//FIGURE 5 **********************************
subplot(2,2,4);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
```

```
DataView(DVindex).children(1).y_label.text = "Ouput Angle";


//Need to compute the output angle with respect to the input angle
Output_Pnt_Transl(:,1)=[];//kill the first point because its [0;0]
End_i = size(Output_Pnt_Transl);
for i = 1 : End_i(2)
  Len1 = (Output_Pnt_Transl(1,i)^2 + Output_Pnt_Transl(2,i)^2)^0.5;
  // Will be taking the dot product with the vetor [1,0] to get the cosine
  Ang_frm_Cos($+1) = Output_Pnt_Transl(1,i)/Len1;
  Ang_frm_Cos($) = acosd(Ang_frm_Cos($));

  Ang_frm_Sin($+1) = Output_Pnt_Transl(2,i)/Len1;
  Ang_frm_Sin($) = asind(Ang_frm_Sin($));

  // We need to correct the angle as asind range is [-90 90], and acosd
  // range is [0 180]. In this case acosd yields the correct value but
  // incorrect sign from 0 to 180 so we just need to negate it, in this
  // range the results for asind is negative. For the correct angle in
  // the remaining range, we need to subtract 360 from acosd (in this
  // range asind is positive).

  if (Ang_frm_Sin($) <= 0) then // asind is negative or 0
      Output_Ang($+1) = -Ang_frm_Cos($);
  else
      Output_Ang($+1) = Ang_frm_Cos($) - 360;
  end

  IOAng_LeadLag($+1) = Output_Ang($) + Input_Ang(i);
  // The difference from a perfect 1 to -1 trajectory.
end

Output_Ang($) = -360; // Correction to last angle
IOAng_LeadLag($) = Output_Ang($) + Input_Ang($); // correction to last angle


plot(Input_Ang, Output_Ang);
//get handle on line drawn
DataLine4 = DataView(DVindex).children(1).children(1).children(1);
DataLine4.thickness = 3;
DataLine4.mark_style = 9;
DataLine4.mark_mode = "off";
DataLine4.polyline_style = 1;
DataLine4.line_style = 3; //1 ->solid; 3 ->dash-dot
DataLine4.foreground = 5; //1-> Black; 2-> Blue; 5-> Red

LO_IO_Angs = [Input_Ang, Output_Ang];

A1 = [0, 360];
A2 = [0, -360];
plot(A1, A2);
//get handle on line drawn
DataLine4 = DataView(DVindex).children(1).children(1).children(1);
DataLine4.thickness = 2;
DataLine4.mark_style = 9;
DataLine4.mark_mode = "off";
DataLine4.polyline_style = 1;
DataLine4.line_style = 1; //1 ->solid; 3 ->dash-dot
DataLine4.foreground = 2; //1-> Black; 2-> Blue; 5-> Red
drawnow();


//FIGURE 6 **********************************
DVindex = 2;
DataView(DVindex) = figure(DVindex+1);
DataView(DVindex).figure_name = "Data Graphs 2";
DataView(DVindex).info_message = "";
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
DataView(DVindex).background = 8; //White

subplot(2,2,1);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
```

```
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Output Angle Lead/Lag";

plot(Input_Ang, IOAng_LeadLag);
//get handle on line drawn
DataLine5 = DataView(DVindex).children(1).children(1).children(1);
DataLine5.thickness = 3;
DataLine5.mark_style = 9;
DataLine5.mark_mode = "off";
DataLine5.polyline_style = 1;
DataLine5.line_style = 1; //solid
DataLine5.foreground = 2; //1-> Black; 2-> Blue; 5-> Red

LO_LeadLag = [Input_Ang, IOAng_LeadLag];
drawnow();


//FIGURE 7 *********************************
subplot(2,2,2);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Time/Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Angular Speed";


//Need to compute the angular velocity
// Already killed the first point in Output_Pnt_Transl which is [0,0] above!
End_i = size(Output_Pnt_Transl);
for i = 2 : End_i(2)
  //Len1 = (Output_Pnt_Transl(1,i-1)^2 + Output_Pnt_Transl(2,i-1)^2)^0.5;
  //Len2 = (Output_Pnt_Transl(1,i)^2 + Output_Pnt_Transl(2,i)^2)^0.5;
  //Output_Ang_Vel($+1) = (Output_Pnt_Transl(1,i-1)*Output_Pnt_Transl(1,i) + ...
  //Output_Pnt_Transl(2,i-1)*Output_Pnt_Transl(2,i))/(Len1*Len2);
  //Dot product of unit vectors gives cosine of angle

  Output_Ang_Vel($+1) = (Output_Ang(i) - Output_Ang(i-1))/AngInc;
  Input_Ang_Vel ($+1) = (Input_Ang(i-1) + Input_Ang(i))/2;
end


plot(Input_Ang_Vel, Output_Ang_Vel);
//get handle on line drawn
DataLine5 = DataView(DVindex).children(1).children(1).children(1);
DataLine5.thickness = 3;
DataLine5.mark_style = 9;
DataLine5.mark_mode = "off";
DataLine5.polyline_style = 1;
DataLine5.line_style = 1; //solid
DataLine5.foreground = 2; //1-> Black; 2-> Blue; 5-> Red
drawnow();

LO_Ang_Vel = [Input_Ang_Vel, Output_Ang_Vel];

//FIGURE 8 *********************************
subplot(2,2,3);
drawlater();
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Time/Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Angular Acceleration";


//Need to compute the angular acceleration
End_i = size(Output_Ang_Vel);
for i = 2 : End_i(1)
  //Output_Ang_Acc($+1) = Output_Ang_Vel(i) - Output_Ang_Vel(i-1);
  //a delta_t of 1 second!

  Output_Ang_Acc($+1) = (Output_Ang_Vel(i) - Output_Ang_Vel(i-1))/AngInc;
```

```
  Input_Ang_Acc ($+1) = (Input_Ang_Vel(i-1) + Input_Ang_Vel(i))/2;
end

plot(Input_Ang_Acc, Output_Ang_Acc);
//get handle on line drawn
DataLine6 = DataView(DVindex).children(1).children(1).children(1);
DataLine6.thickness = 3;
DataLine6.mark_style = 9;
DataLine6.mark_mode = "off";
DataLine6.polyline_style = 1;
DataLine6.line_style = 1; //1-> solid,  3-> Dash dot style
DataLine6.foreground = 2; //1-> Black; 2-> Blue; 5-> Red
drawnow();

LO_Ang_Acc = [Input_Ang_Acc, Output_Ang_Acc];
```

2.3 **File name:** LD5_8BarTrajCalLp1.sce

Purpose: This routine calculates all relevant data for the output of the secondary 3 bar slider-pivot linkage, for multiple configurations of angle 3, the angle between the proximal and distal links. The orbits are stacked in 3D and 5 selected orbits are shown in 2D. It finds the RMS and average errors. It also plots the output angle against the input angle, and determines output angle lead-lag. These are plotted in both 3D and 2D. Measures of velocity and acceleration are also plotted in 2D and 3D.

```
clc;
xdel(winsid()); //Destroy all existing figures
clearglobal();
clear;
clear;
lines(0); //disables vertical paging
Save_Data = 0;


Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\Linkage Designer";
getd(Funtion_Path);//Loads all functions from this directory into SCILAB
Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\Test Scripts";
getd(Funtion_Path);//Loads all functions from this directory into SCILAB
Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics";
getd(Funtion_Path);//Loads all functions from this directory into SCILAB


//Returns screen size in pixels, on this PC it s 1920 by 1200
ScreenSizePX = get(0, "screensize_px")
//Returns screen size in points, on this PC it s 960 by 600
//ScreenSizePT = get(0, "screensize_pt")

L1_len = 8;
L2_len = 1;

DVindex = 1;
DataView = list(0);
DataView(DVindex) = figure(DVindex);
// Create figure having figure_id==1
DataView(DVindex).figure_name = "Linkage Designer"
DataView(DVindex).info_message = ""
Win_Width = ScreenSizePX(3);
Win_Height = ScreenSizePX(4);
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
DataView(DVindex).background = 8; //White

// ***************************************************
// FIGURE 1: Draws the linkage
subplot (2,2,1);
drawlater;
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
tmpBnd = (2*L1_len + 0.25*L2_len);
//[xmin ymin; xmax ymax]->must be 2 dimensional
DataView(DVindex).children(1).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
```

```
DataView(DVindex).children(1).grid = [0 0]; // displays it in black
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X Axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y Axis";
toolbar(DataView(DVindex),'off');


AngIndex = 1;
Angles = [0 0 0 0 0 0];
//AngInc = 30;


//Link1
Lnks = list(1);
x = [0, L1_len/2, L1_len];
y = [0, 0, 0];
xpoly(x, y);
Lnks($) = get("hdl"); //get handle on current entity (here the polyline entity)

//Link 2 Set
SubLnks2 = list(1);
x = [0, L2_len];
y = [0, 0];
xpoly(x, y);
SubLnks2($) = get("hdl"); //get handle on current entity (here the polyline entity)

x = [L2_len, L2_len+L1_len];
y = [0, 0];
xpoly(x, y);
SubLnks2($+1) = get("hdl"); //get handle on current entity (here the polyline entity)

x = [L1_len+L2_len, 2*L1_len+L2_len];
y = [0, 0];
xpoly(x, y);
SubLnks2($+1) = get("hdl"); //get handle on current entity (here the polyline entity)
Lnks($+1) = SubLnks2;

//Link 3 Set
SubLnks3 = list(1);
x = [0, L2_len];
y = [0, 0];
xpoly(x, y);
SubLnks3($) = get("hdl"); //get handle on current entity (here the polyline entity)

x = [L2_len, L2_len+L1_len];
y = [0, 0];
xpoly(x, y);
SubLnks3($+1) = get("hdl"); //get handle on current entity (here the polyline entity)

x = [L1_len+L2_len, 2*L1_len+L2_len];
y = [0, 0];
xpoly(x, y);
SubLnks3($+1) = get("hdl"); //get handle on current entity (here the polyline entity)


Lnks($+1) = SubLnks3;

Colours = [1 2 5];//1-> Black; 2-> Blue; 5-> Red

End_i = size(Lnks);
//disp(End_i);
for i = 1:End_i(1)
  End_j = size(Lnks(i));
  //disp(End_j);
  for j = 1:End_j(1)
    Lnks(i)(j).thickness=3;
    Lnks(i)(j).mark_style=9;
    Lnks(i)(j).mark_mode="on";
    Lnks(i)(j).polyline_style=1;
    Lnks(i)(j).foreground = Colours(i);
  end
end

DrawEntities = %f;
```

```
Max_Dev = -100;
Min_Dev = 100;
MaxErr = -100;
MinErr = 100;

Ang2Vec = [];
Data3D = list(1);
Height = 0;
i = 1;
end_Ang2 = 360; //360;
end_Ang3 = 360;
AngInc2 = 10
AngInc3 = 5;

for TempAng2 = 0: AngInc2: end_Ang2

  //Need to reset the Max limits, and Error variables
  Max_X = -100;
  Min_X = 100;
  Max_Y = -100;
  Min_Y = 100;
  SqErrorSum = 0;
  ErrorSum = 0;

  j = 1;
  Input_Ang = [];
  for TempAng3 = 0: AngInc3: end_Ang3

    drawlater();
    Angles = [0, TempAng2, TempAng3, 0, 0, 0];
    //LINK 1
    x1 = cosd(Angles(1))*[0 L1_len/2 L1_len];
    y1 = sind(Angles(1))*[0 L1_len/2 L1_len];
    z1 = [0 0 0];
    xyz = [x1; y1; z1];
    set(Lnks(1), "data",  [x1' y1']);

    //LINK SET 2
    x21 = cosd(Angles(2))*[0 L2_len];
    y21 = sind(Angles(2))*[0 L2_len];
    z21 = [0 0];
    set(Lnks(2)(1), "data",  [x21' y21']);

    dx21 = x1(2) - x21(2);
    dy21 = y1(2) - y21(2);
    dl21 = (dx21^2 + dy21^2)^0.5;
    dx = dx21/dl21;
    dy = dy21/dl21;

    //translate to the end of link 2,1
    x22 = L1_len*[0 dx] + [x21(2) x21(2)];
    y22 = L1_len*[0 dy] + [y21(2) y21(2)];
    z22 = [0 0];
    set(Lnks(2)(2), "data",  [x22' y22']);

    dx22 = x22(2) - x1(3);
    dy22 = y22(2) - y1(3);
    dl22 = (dx22^2 + dy22^2)^0.5;
    dx = dx22/dl22;
    dy = dy22/dl22;

    //translate to the end of link 1
    x23 = L1_len*[0 dx/2 dx] + [x1(3) x1(3) x1(3)];
    y23 = L1_len*[0 dy/2 dy] + [y1(3) y1(3) y1(3)];
    z23 = [0 0 0];
    set(Lnks(2)(3), "data",  [x23' y23']);

    //LINK SET 3
    x31 = cosd(Angles(3))*[0 L2_len];
    y31 = sind(Angles(3))*[0 L2_len];
    z31 = [0 0];
    set(Lnks(3)(1), "data",  [x31' y31']);

    dx31 = x1(2) - x31(2);
    dy31 = y1(2) - y31(2);
    dl31 = (dx31^2 + dy31^2)^0.5;
    dx = dx31/dl31;
```

```
  dy = dy31/dl31;

  //translate to the end of link 3,1
  x32 = L1_len*[0 dx] + [x31(2) x31(2)];
  y32 = L1_len*[0 dy] + [y31(2) y31(2)];
  z32 = [0 0];
  set(Lnks(3)(2), "data",  [x32' y32']);

  dx32 = x23(2) - x32(2);
  dy32 = y23(2) - y32(2);
  dl32 = (dx32^2 + dy32^2)^0.5;
  dx = dx32/dl32;
  dy = dy32/dl32;

  //translate to the end of link 3,2
  x33 = L1_len*[0 dx] + [x32(2) x32(2)];
  y33 = L1_len*[0 dy] + [y32(2) y32(2)];
  z33 = [0 0];
  set(Lnks(3)(3), "data",  [x33' y33']);


  Orb_Radius = ((x33(2)-x23(3))^2 + (y33(2)-y23(3))^2)^0.5;
  if (Orb_Radius > Max_Dev)
    Max_Dev = Orb_Radius;
  elseif (Orb_Radius < Min_Dev)
    Min_Dev = Orb_Radius;
  end

  // This computes the square and average errors when compared to a
  // perfect circle orbit
  dl = Orb_Radius - L2_len;
  SqErrorSum = SqErrorSum + dl^2;
  ErrorSum = ErrorSum + dl;

  // Finding Max_X and Min_X
  if (x33(2) > Max_X) // Finding Max_X
    Max_X = x33(2);
  elseif (x33(2) < Min_X) // Finding Min_X
    Min_X = x33(2);
  end

  // Finding Max_Y and Min_Y
  if (y33(2) > Max_Y) // Finding Max_Y
    Max_Y = y33(2);
  elseif (y33(2) < Min_Y) // Finding Min_Y
    Min_Y = y33(2);
  end

  if (DrawEntities == %t)
    drawnow();
  end
  ContourLine(:, j) = [x33(2)-x23(3); y33(2)-y23(3); TempAng2]; //Height]
  j = j+1;
end


Delta_X(i) = Max_X - Min_X;
Delta_Y(i) = Max_Y - Min_Y;
RtMeanSqError(i) = (SqErrorSum/j)^0.5;
MeanError(i) = ErrorSum/j;
//Need to find Y-limits for plot
if (RtMeanSqError(i) > MaxErr) then
    MaxErr = RtMeanSqError(i)
end
if (MeanError(i) > MaxErr) then
    MaxErr = MeanError(i)
end
if (RtMeanSqError(i) < MinErr) then
    MinErr = RtMeanSqError(i)
end
if (MeanError(i) < MinErr) then
    MinErr = MeanError(i)
end


Data3D($+1) = ContourLine;
Height = i;
```

```
  i=i+1;
  Ang2Vec = [Ang2Vec; TempAng2];
end

LO_ContRec = [Ang2Vec, Delta_X, Delta_Y];
LO_Errors = [Ang2Vec, MeanError, RtMeanSqError];



//*******************************************
//FIGURE 2: Stacks the trajectories in 3D
drawlater();
subplot (2,2,2);

Data3D(1) = null(); //Kill first point as its just the 1 initialised
//earlier adn not the list data

End_i = size(Data3D)
for i = 1 : End_i(1);
    // Creating data set for Facet, and closing the contour by adding
    // redundant start point at end
    X(i, :) = [Data3D(i)(1,:), Data3D(i)(1,1)]; //All x data
    Y(i, :) = [Data3D(i)(2,:), Data3D(i)(2,1)]; //All y data
    Z(i, :) = [Data3D(i)(3,:), Data3D(i)(3,1)]; //All z data
end

// 3D SURFACE PLOT
drawlater();
surf(X, Y, Z);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;
// All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3;
// interpolated shading mode. The default is 4 ('flat' mode) for surf


DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "b. 3D Stacking of Trajectory Curves";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = "Angle 3";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
// Displays grid in black, negative numbers hide it
DataView(DVindex).children(1).grid = [0 0 0];
tmpBnd = (1.25*L2_len);
//[xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).data_bounds = ...
            [-tmpBnd, -tmpBnd, -1; tmpBnd, tmpBnd, TempAng2+10];
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off"
drawnow();




//*********************************************************************
//FIGURE 3: Plots selected contours from the 3D stack
drawlater();
subplot (2,2,3);


LO_Data_Angs = [];
LO_Data_x = [];
```

```
LO_Data_y = [];

//Index selection of data in Data3D
sz_Data3D = size(Data3D);
Index_2D_Graph = [2, round(0.25*sz_Data3D), round(0.5*sz_Data3D),...
round(0.75*sz_Data3D), sz_Data3D-1];

LO_Data_Pivs = [];
LO_Data_x = [];
LO_Data_y = [];
drawlater();
for i = 1 : 5
    px = Data3D(Index_2D_Graph(i))(1,:); //All x data
    py = Data3D(Index_2D_Graph(i))(2,:); //All y data
    xpoly(px, py);
    Orbit(i) = get("hdl");
    //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).mark_style = 9;
    Orbit(i).mark_mode = "off";
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_mode = "on";
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_Graph(i);
    //maps the colour from the colour map to match with the 3D graph

    LO_Data_Angs = [LO_Data_Angs, Ang2Vec(Index_2D_Graph(i))];
    LO_Data_x = [LO_Data_x, px'];
    LO_Data_y = [LO_Data_y, py'];
end

// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "c. Top View Illustrating 5 Trajectories of Interest"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y axis";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid
tmpBnd = 1.05*Max_Dev;
DataView(DVindex).children(1).data_bounds = [-tmpBnd, -tmpBnd; tmpBnd, tmpBnd];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;

Legd = legend(DataView(DVindex).children(1), ['1'; '2'; '3'; '4'; '5']);
Legd.font_size = 5;
Legd.font_style = 5;
Legd.visible = 'on';
Legd.foreground = -1; //Black
Legd.background = -2; //White
Legd.font_style = 6; //Sans Serif close to arial
drawnow();




// ********************************************************************
// FIGURE 4: Plots the output angle versus the input angle for multiple
// values of angle 3


drawlater();
subplot(2,2,4);
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
```

```
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Ouput Angle";


Ang2Vec_mat = [];
Input_Ang_vec = [];
for TempAng3 = 0: AngInc3: end_Ang3
    Input_Ang_vec = [Input_Ang_vec, TempAng3];
    Ang2Vec_mat = [Ang2Vec_mat, Ang2Vec];
end


sz_X = size(X);
End_k = sz_X(2)-1; //Last column is repeated
End_p = sz_X(1);

Input_Ang = [];
Output_Ang = zeros(End_p, End_k);
IOAng_LeadLag = zeros(End_p, End_k);

for p = 1 : End_p
    for k = 1 : End_k
        Len1 = (X(p,k)^2 + Y(p,k)^2)^0.5;
        // Will be taking the dot product with the vetor [1,0] to get the cosine
        Cosd_ang = X(p,k)/Len1;
        Sind_ang = Y(p,k)/Len1;
        Output_Ang(p, k) = A_sin_cos_d(Sind_ang, Cosd_ang);
        IOAng_LeadLag(p, k) = Output_Ang(p, k) - Input_Ang_vec(k);
        // The difference from a perfect 1 to -1 trajectory.

    end //...for k = 1 : End_k
    Input_Ang = [Input_Ang; Input_Ang_vec];
end //...for p = 1 : End_p


//Make continuous about 0,-360 boundary, first along rows...
Output_Ang = Output_Ang';
Output_Ang = Ang_Continuity(Output_Ang);
Output_Ang = Output_Ang';


//then along columns ... (have to do this 3 times)
Output_Ang = Ang_Continuity(Output_Ang);
Output_Ang = Ang_Continuity(Output_Ang);
Output_Ang = Ang_Continuity(Output_Ang);


Start_Ang = Output_Ang(:, 1); //first column



//HAVE TO USE A SURF COMMAND HERE, FOR 3D SURFACE PLOT
surf(Input_Ang, Output_Ang, Ang2Vec_mat);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;
// All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3;
// interpolated shading mode. The default is 4 ('flat' mode) for surf


DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "d. 3D Stacking of Output Angle";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "x"; //"X axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = ""; //"Y axis";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = ""; //"Height (Z axis)";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
```

```
// Displays grid in black, negative numbers hide it
DataView(DVindex).children(1).grid = [0 0 0];
tmpBnd = (1.25*L2_len);
//[xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).data_bounds = [0, -720, 0; TempAng3, 360, TempAng2+10];
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off"
drawnow()



// NEXT FIGURE WINDOW
// FIGURE 5: Plots the output angle versus the input angle for selected values
// of angle 3.
DVindex = DVindex+1;
DataView(DVindex) = figure(DVindex);
DataView(DVindex).figure_name = "Data Graph 2"
DataView(DVindex).info_message = ""
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];
drawlater();
subplot(2,2,1);

//Index selection of data in Data3D
sz_Output_Ang= size(Output_Ang);
num_rows = sz_Output_Ang(1);
Index_2D_Graph = [2, round(0.25*num_rows), round(0.5*num_rows),...
round(0.75*num_rows), num_rows-1];


LO_Sel_Angs = [];
LO_Data_IA = [];
LO_Data_OA = [];
for i = 1 : 5
    px = Input_Ang(Index_2D_Graph(i), :); //All x data
    py = Output_Ang(Index_2D_Graph(i), :) - Output_Ang(Index_2D_Graph(i), 1);
    //All y data
    xpoly(px, py);
    Orbit(i) = get("hdl"); //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).mark_style = 9;
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_Graph(i);
    Orbit(i).mark_mode = "off";
    //maps the colour from the colour map to match with the 3D graph

    LO_Sel_Angs = [LO_Sel_Angs, Ang2Vec(Index_2D_Graph(i))];
    LO_Data_IA = [LO_Data_IA, px'];
    LO_Data_OA = [LO_Data_OA, py'];
end


// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "e. Top View Illustrating 5 Trajectories of Interest"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "X axis";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Y axis";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid
tmpBnd = 1.05*Max_Dev;
DataView(DVindex).children(1).data_bounds = [0, 0; 360, 360];
// [xmin ymin; xmax ymax]-> for 2D
```

```
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;

Legd = legend(DataView(DVindex).children(1), ['1'; '2'; '3'; '4'; '5']);
Legd.font_size = 5;
Legd.font_style = 5;
Legd.visible = 'on';
Legd.foreground = -1; //Black
Legd.background = -2; //White
Legd.font_style = 6; //Sans Serif close to arial



// ********************************************************************
// FIGURE 6: Plots output start angle versus angle 3 for all contours in the
// stack.
subplot(2,2,2);

tmp_mat = [Ang2Vec, Start_Ang];

plot(Ang2Vec, Start_Ang);
//get handle on line drawn
Line_Hdl = DataView(DVindex).children(1).children(1).children(1);
Line_Hdl.thickness = 2;
Line_Hdl.mark_style = 9;
Line_Hdl.mark_mode = "off";
Line_Hdl.polyline_style = 1;
Line_Hdl.line_style = 1; //1 ->solid; 3 ->dash-dot
Line_Hdl.foreground = 2; //1-> Black; 2-> Blue; 5-> Red



// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "f. Start angle versus angle 3"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Angle 3";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Start angle";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid
tmpBnd = 1.05*Max_Dev;
DataView(DVindex).children(1).data_bounds = [0, -720; 360, 0];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
drawnow();




//Make continuous about 0,-360 boundary, first along rows...
IOAng_LeadLag = IOAng_LeadLag';
IOAng_LeadLag = Ang_Continuity(IOAng_LeadLag);
IOAng_LeadLag = IOAng_LeadLag';

//then along columns ... (have to do this 3 times)
IOAng_LeadLag = Ang_Continuity(IOAng_LeadLag);
IOAng_LeadLag = Ang_Continuity(IOAng_LeadLag);
IOAng_LeadLag = Ang_Continuity(IOAng_LeadLag);


// ********************************************************************
// FIGURE 7: Output angle lead-lag versus input angle for multiple values
// of angle 3


drawlater();
subplot(2,2,3);
//HAVE TO USE A SURF COMMAND HERE, 3D SURFACE PLOT
```

```
surf(Input_Ang, IOAng_LeadLag, Ang2Vec_mat);
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;
// All facets are painted using one color index per facet proportional to z.
//e.color_flag = 3;
// interpolated shading mode. The default is 4 ('flat' mode) for surf


DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "g. 3D Stacking of output angle lead-lag";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output ang";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = "Angle 3";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0];
// Displays grid in black, negative numbers hide it

//[Result, Msg] = Find_Min_Max(x)
x_bounds = Find_Min_Max(Input_Ang);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
y_bounds = Find_Min_Max(IOAng_LeadLag);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
z_bounds = Find_Min_Max(Ang2Vec);
minz = z_bounds(1) - 0.1*abs(z_bounds(1));
maxz = z_bounds(2) + 0.1*abs(z_bounds(2));

DataView(DVindex).children(1).data_bounds = [minx, miny, minz; maxx, maxy, maxz];
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off"
drawnow()




// ********************************************************************
//FIGURE 8: Output angle lead-lag versus input angle for selected values
// of angle 3
drawlater();
subplot(2,2,4);
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Ouput Angle";



//Index selection of data in Data3D
sz_IOAng_LL= size(IOAng_LeadLag);
num_rows = sz_IOAng_LL(1);
Index_2D_Graph = [2, round(0.25*num_rows), round(0.5*num_rows),...
                     round(0.75*num_rows), num_rows-1];

LOSel_Angs = [];
LODat_LLIA = [];
LODat_LLOA = [];
for i = 1 : 5
    px = Input_Ang(Index_2D_Graph(i), :); //All x data
    py = IOAng_LeadLag(Index_2D_Graph(i), :) - IOAng_LeadLag(Index_2D_Graph(i), 1);
    //All y data
```

```
    xpoly(px, py);
    Orbit(i) = get("hdl"); //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).mark_style = 9;
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_Graph(i);
    Orbit(i).mark_mode = "off";
    //maps the colour from the colour map to match with the 3D graph

    LOSel_Angs = [LO_Sel_Angs, Ang2Vec(Index_2D_Graph(i))];
    LODat_LLIA = [LODat_LLIA, px'];
    LODat_LLOA = [LODat_LLOA, py'];
end


// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "h. Output angle lead-lag, for selected values of angle 3"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output ang lead-lag";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid


//[Result, Msg] = Find_Min_Max(x)
y_bounds = Find_Min_Max(LODat_LLOA);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
x_bounds = Find_Min_Max(Input_Ang);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
DataView(DVindex).children(1).data_bounds = [minx, miny; maxx, maxy];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;


Legd = legend(DataView(DVindex).children(1), ['1'; '2'; '3'; '4'; '5']);
Legd.font_size = 5;
Legd.font_style = 5;
Legd.visible = 'on';
Legd.foreground = -1; //Black
Legd.background = -2; //White
Legd.font_style = 6; //Sans Serif close to arial
drawnow();



// NEW FIGURE WINDOW *************************************************
////FIGURE 9: Plots the angular velocity for the output of the secondary
// 3 bar linkage, for multiple values of angle 3
DVindex = DVindex+1;
DataView(DVindex) = figure(DVindex);
DataView(DVindex).figure_name = "Data Graph 3"
DataView(DVindex).info_message = ""
DataView(DVindex).figure_position = [0, 0];
DataView(DVindex).figure_size = [Win_Width, Win_Height];

drawlater();
subplot(2,2,1);
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Input Angle";
```

```
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Ouput Angle";


//Need to compute the angular velocity
sz_Output_Ang = size(Output_Ang);
End_p = sz_Output_Ang(1);
End_k = sz_Output_Ang(2);
OAng_Vel = zeros(End_p, End_k-1);
IAng_Vel = zeros(End_p, End_k-1);
for p = 1 : End_p   // Velocity
    for k = 2 : End_k  // Velocity

                OAng_Vel(p,k-1)  =  (Output_Ang(p,k)  -  Output_Ang(p,k-1))/(Input_Ang(p,k)  -
Input_Ang(p,k-1));
        IAng_Vel(p,k-1) = 0.5*(Input_Ang(p,k) + Input_Ang(p,k-1));

    end //...for k = 1 : End_k   // Velocity
end //...for p = 1 : End_p   // Velocity


//HAVE TO USE A SURF COMMAND HERE!!!!
// 3D SURFACE PLOT
surf(IAng_Vel, OAng_Vel, Ang2Vec_mat(:, 1:End_k-1));
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;


DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "i. 3D Stacking of anglular velocity";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output ang";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = "Angle 3";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(sz_Output_Ang(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0];
// Displays grid in black, negative numbers hide it

//[Result, Msg] = Find_Min_Max(x)
x_bounds = Find_Min_Max(IAng_Vel);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
y_bounds = Find_Min_Max(OAng_Vel);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
z_bounds = Find_Min_Max(Ang2Vec);
minz = z_bounds(1) - 0.1*abs(z_bounds(1));
maxz = z_bounds(2) + 0.1*abs(z_bounds(2));

DataView(DVindex).children(1).data_bounds = [minx, miny, minz; maxx, maxy, maxz];
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off"
drawnow()



// ***************************************************************************
////FIGURE 10: Plots the angular velocity for the output of the secondary
// 3 bar linkage, for selected values of angle 3
drawlater();
subplot(2,2,2);
//Index selection of data in Data3D
sz_OAng_Vel= size(OAng_Vel);
num_rows = sz_OAng_Vel(1);
Index_2D_VelG = [2, round(0.25*num_rows), round(0.5*num_rows),...
                    round(0.75*num_rows), num_rows-1];
```

```
LOSel_Angs = [];
LO_OAng_Vel = [];
LO_IAng_Vel = [];
for i = 1 : 5
    px = IAng_Vel(Index_2D_VelG(i), :); //All x data
    py = OAng_Vel(Index_2D_VelG(i), :);
    //All y data
    xpoly(px, py);
    Orbit(i) = get("hdl"); //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).mark_style = 9;
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_VelG(i);
    Orbit(i).mark_mode = "off";
    //maps the colour from the colour map to match with the 3D graph

    LOSel_Angs = [LO_Sel_Angs, Ang2Vec(Index_2D_VelG(i))];
    LO_IAng_Vel = [LO_IAng_Vel, px'];
    LO_OAng_Vel = [LO_OAng_Vel, py'];
end



// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "j. Output anglular velocity, for selected values of angle 3"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output angular velocity";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(End_i(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid


//[Result, Msg] = Find_Min_Max(x)
y_bounds = Find_Min_Max(LO_OAng_Vel);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
x_bounds = Find_Min_Max(LO_IAng_Vel);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
DataView(DVindex).children(1).data_bounds = [minx, miny; maxx, maxy];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;


Legd = legend(DataView(DVindex).children(1), ['1'; '2'; '3'; '4'; '5']);
Legd.font_size = 5;
Legd.font_style = 5;
Legd.visible = 'on';
Legd.foreground = -1; //Black
Legd.background = -2; //White
Legd.font_style = 6; //Sans Serif close to arial
drawnow();







// ************************************************************************
////FIGURE 11: Plots the angular acceleration for the output of the secondary
// 3 bar linkage, for selected values of angle 3
```

```
drawlater();
subplot(2,2,3);
DataView(DVindex).children(1).axes_visible = ["on","on","off"];
DataView(DVindex).children(1).grid = [1 1];
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).x_label.font_size = 5;
DataView(DVindex).children(1).x_label.text = "Input Angle";
DataView(DVindex).children(1).y_label.font_size = 5;
DataView(DVindex).children(1).y_label.text = "Ouput Angle";



//Need to compute the angular velocity
// Already killed the first point in Output_Pnt_Transl which is [0,0] above!
sz_OAng_Vel = size(OAng_Vel);
End_p = sz_OAng_Vel(1);
End_k = sz_OAng_Vel(2);
OAng_Acc = zeros(End_p, End_k-1);
IAng_Acc = zeros(End_p, End_k-1);
for p = 1 : End_p  // Accleration
    for k = 2 : End_k // Accleration

        OAng_Acc(p,k-1) = (OAng_Vel(p,k) - OAng_Vel(p,k-1))/(IAng_Vel(p,k) - IAng_Vel(p,k-1));
        IAng_Acc(p,k-1) = 0.5*(IAng_Vel(p,k) + IAng_Vel(p,k-1));

    end //...for k = 1 : End_k   // Accleration
end //...for p = 1 : End_p   // Accleration



//HAVE TO USE A SURF COMMAND HERE, 3D SURFACE PLOT
surf(IAng_Acc, OAng_Acc, Ang2Vec_mat(:, 1:End_k-1));
e = gce();
e.cdata_mapping = 'direct' // default is 'scaled' relative to the colormap
e.color_mode = 2;
e.color_flag = 1;



DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = "k. 3D Stacking of anglular Acceleration";
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output ang";
DataView(DVindex).children(1).z_label.font_size = 5
DataView(DVindex).children(1).z_label.text = "Angle 3";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(sz_OAng_Vel(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on","on"];
DataView(DVindex).children(1).grid = [0 0 0];
// Displays grid in black, negative numbers hide it

//[Result, Msg] = Find_Min_Max(x)
x_bounds = Find_Min_Max(IAng_Acc);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
y_bounds = Find_Min_Max(OAng_Acc);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
z_bounds = Find_Min_Max(Ang2Vec);
minz = z_bounds(1) - 0.1*abs(z_bounds(1));
maxz = z_bounds(2) + 0.1*abs(z_bounds(2));

DataView(DVindex).children(1).data_bounds = [minx, miny, minz; maxx, maxy, maxz];
DataView(DVindex).children(1).view = "3d";
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;
DataView(DVindex).children(1).box = "off"
drawnow()
```

```
// ************************************************************************
////FIGURE 12: Plots the angular acceleration for the output of the secondary
// 3 bar linkage, for selected values of angle 3
drawlater();
subplot(2,2,4);
//Index selection of data in Data3D
sz_OAng_Acc = size(OAng_Acc);
num_rows = sz_OAng_Acc(1);
Index_2D_AccG = [2, round(0.25*num_rows), round(0.5*num_rows),...
                    round(0.75*num_rows), num_rows-1];

LOSel_Angs = [];
LO_OAng_Acc = [];
LO_IAng_Acc = [];
for i = 1 : 5
    px = IAng_Acc(Index_2D_AccG(i), :); //All x data
    py = OAng_Acc(Index_2D_AccG(i), :);
    //All y data
    xpoly(px, py);
    Orbit(i) = get("hdl"); //get handle on current entity (here the polyline entity)
    Orbit(i).thickness = 3;
    Orbit(i).mark_style = 9;
    Orbit(i).polyline_style = 1;
    Orbit(i).line_style = 1; // 1-Solid; 3-Dash dot style
    Orbit(i).mark_style = 0;
    Orbit(i).mark_size_unit = "point";
    Orbit(i).mark_size = 3;
    Orbit(i).foreground = Index_2D_AccG(i);
    Orbit(i).mark_mode = "off";
    //maps the colour from the colour map to match with the 3D graph

    LOSel_Angs = [LO_Sel_Angs, Ang2Vec(Index_2D_AccG(i))];
    LO_IAng_Acc = [LO_IAng_Acc, px'];
    LO_OAng_Acc = [LO_OAng_Acc, py'];
end



// Figure showing 2D contours only
//Set the figure properties after you've drawn the last figure entity
DataView(DVindex).children(1).font_size = 5;
DataView(DVindex).children(1).title.text = ...
    "1. Output anglular accerlation, for selected values of angle 3"
DataView(DVindex).children(1).title.font_size = 6;
DataView(DVindex).children(1).x_label.font_size = 5
DataView(DVindex).children(1).x_label.text = "Input ang";
DataView(DVindex).children(1).y_label.font_size = 5
DataView(DVindex).children(1).y_label.text = "Output angular acceleration";
DataView(DVindex).children(1).z_label.font_angle = 270;
DataView(DVindex).color_map = rainbowcolormap(sz_OAng_Acc(1)); //Set RGB Colour Map
DataView(DVindex).background = -2; //White
DataView(DVindex).children(1).axes_visible = ["on","on"];
DataView(DVindex).children(1).grid = [-1 -1]; //Hide grid


//[Result, Msg] = Find_Min_Max(x)
y_bounds = Find_Min_Max(LO_OAng_Acc);
miny = y_bounds(1) - 0.1*abs(y_bounds(1));
maxy = y_bounds(2) + 0.1*abs(y_bounds(2));
x_bounds = Find_Min_Max(LO_IAng_Acc);
minx = x_bounds(1) - 0.1*abs(x_bounds(1));
maxx = x_bounds(2) + 0.1*abs(x_bounds(2));
DataView(DVindex).children(1).data_bounds = [minx, miny; maxx, maxy];
// [xmin ymin; xmax ymax]-> for 2D
DataView(DVindex).children(1).isoview="on";
DataView(DVindex).children(1).hidden_axis_color = 0;


Legd = legend(DataView(DVindex).children(1), ['1'; '2'; '3'; '4'; '5']);
Legd.font_size = 5;
Legd.font_style = 5;
Legd.visible = 'on';
Legd.foreground = -1; //Black
```

```
Legd.background = -2; //White
Legd.font_style = 6; //Sans Serif close to arial
drawnow();
```

2.4 **File name:** A_sin_cos_d.sci

Purpose: Calculates the angle in degrees from the given sine and cosine values.

```
function [Result, Msg]=A_sin_cos_d(SindA, CosdA)

    This_Fun_Name = "A_sin_cos_d:";
    Err_Msg = "";

    tolerance = 10^-4;

    if (SindA > 1+tolerance) | (SindA < -1-tolerance) then
        ang = [];
        Err_Msg = "Abs value of sin > 1: SindA = " + string(SindA);
        disp(This_Fun_Name + " " + Err_Msg);

    elseif (CosdA > 1+tolerance) | (CosdA < -1-tolerance) then
        ang = [];
        Err_Msg = "Abs value of cos > 1: CosdA = " + string(CosdA);
        disp(This_Fun_Name + " " + Err_Msg);

    elseif ( abs(1 - (SindA^2 + CosdA^2)) > tolerance) then
        ang = [];
        Err_Msg = "Sum of squares ~= 1!";
        disp(This_Fun_Name + " " + Err_Msg);

    elseif ( SindA == 0 ) then
        // ang = 0 or ang = 180
        ang = 0;
        if (abs(cosd(ang) - CosdA) < tolerance) then
            // ang is zero
        elseif (abs(cosd(ang) + CosdA) < tolerance) then
            // ang = 180
            ang = 180;
        else
            ang = [];
            Err_Msg = "Sin is zero but cos is neither 1 nor -1!";
            disp(This_Fun_Name + " " + Err_Msg);
            disp("Sind = " + string(SindA));
            disp("Cosd = " + string(CosdA));
        end

    elseif (SindA > 0) then
        //asind range is -90 to 90, sind > 0 the other possibility is 180 - ang
        // Already checked if SindA is withing tolerance range so next bit
        // of code is fine
        if (SindA > 1) then
            SindA = 1;
            ang = 90;
            Err_Msg = "sin > 1 but within tolerance: (SindA - 1) = ";
            disp(This_Fun_Name + " " + Err_Msg);
            disp(SindA-1);

        else
            ang = asind(SindA);
        end


        if (abs(cosd(ang) - CosdA) < tolerance) then
            // ang assumtion was correct
        elseif (abs(cosd(180 - ang) - CosdA) < tolerance) then
            // ang = 180 - ang
            ang = 180 - ang;
        else
            Err_Msg = 'SindA is +ve but, cosd(asind(SindA)) ~=' + ...
                    ' CosdA and cosd(180 - asind(SindA)) ~= CosdA!';
```

```
            disp(This_Fun_Name + " " + Err_Msg);
            disp("Input Sind = " + string(SindA));
            disp("Input Cosd = " + string(CosdA));
            disp("Angle = " + string(ang));
            disp("Cosd(Angle) = " + string(cosd(ang)));
            disp("Cosd(Angle) - Input Cosd = " + string(cosd(ang) - CosdA));
            ang = [];
        end

    elseif (SindA < 0) then
        //asind range is -90 to 90, sind < 0 the other possibility is
        //180 + abs(ang), and ang is negative

        //Already checked if SindA is withing tolerance range so next
        //bit of code is fine
        if (SindA < -1) then
            SindA = -1;
            ang = 270;
            Err_Msg = "sin < -1 but within tolerance: (SindA + 1) = ";
            disp(This_Fun_Name + " " + Err_Msg);
            disp(SindA+1);
        else
            ang = asind(SindA);
        end


        if (abs(cosd(ang) - CosdA) < tolerance) then
            // ang assumtion was correct
            //All angles are positive
            ang = pmodulo(ang, 360);
        elseif (abs(cosd(180 + abs(ang)) - CosdA) < tolerance) then
            // ang = 180 + abs(ang)
            ang = 180 + abs(ang);
        else
            Err_Msg = 'SindA is -ve but, cosd(asind(SindA)) ~=' + ...
                    ' CosdA and cosd(180 + abs(asind(SindA))) ~= CosdA!';
            disp(This_Fun_Name + " " + Err_Msg);
            disp("Input Sind = " + string(SindA));
            disp("Input Cosd = " + string(CosdA));
            disp("Angle = " + string(ang));
            disp("Cosd(Angle) = " + string(cosd(ang)));
            disp("Cosd(Angle) - Input Cosd = " + string(cosd(ang) - CosdA));
            ang = [];
        end
    end

    Result = ang;
    Msg = This_Fun_Name + Err_Msg;
    if (ang == []) then
        disp(Msg);
    end

endfunction
```

2.5 **File name:** Ang_Continuity.sci

Purpose: This function wraps the angles around the zero-360 and -360-zero boundaries for curve continuity.

```
function [Result, Msg]=Ang_Continuity(Angs)

    This_Fun_Name = "Ang_Continuity:";
    Err_Msg = "";

    //This algorithm traverses about Angs along the columns.
    SZ_Angs = size(Angs); //Animation_Angs has 6 columns
```

```
    for i = 1 : SZ_Angs(2)
        for j = 2 : SZ_Angs(1)

            if (180 < Angs(j,i)) & ((Angs(j,i)-Angs(j-1,i)) >= 180) then
                //0 360 boundary, zero to negative transition!
                // 27, 10, 3, 355, 346 ---- for continuity instead of going
                // from 3 to 355, it must go from 3 to -5....
                Angs(j,i) = Angs(j,i)-360;

            elseif (Angs(j,i) < 180) & ((Angs(j,i)-Angs(j-1,i)) <= -180) then
                //360 0 boundary, 360+ degrees!
                // 340, 355, 3, 10, 27 ---- for continuity instead of going
                // from 355 to 3, it must go from 355 to 363....
                Angs(j,i) = Angs(j,i)+360;

            elseif (-180 < Angs(j,i)) & ((Angs(j,i)-Angs(j-1,i)) >= 180) then
                //-360 0 boundary, -360 degrees!
                // -340, -355, -3, -10, -27 ---- for continuity instead of
                // going from -355 to -3, it must go from -355 to -363....
                Angs(j,i) = Angs(j,i)-360;

            elseif (Angs(j,i)< -180) & ((Angs(j,i)-Angs(j-1,i)) <= -180) then
                //0 -360 boundary, 360+ degrees!
                // -27 -10 -3 -355 -340 ---- for continuity instead of going
                // from -3 to -355, it must go from -3 to 5 ....
                Angs(j,i) = Angs(j,i)+360;
            else

                if (180 <= Angs(j,i)) & ((Angs(j,i)-Angs(j-1,i)) <= -180) then
                    // 568, 600, 186, 199 ---- for continuity instead of going
                    // this adjustment is necessary as we have already changed
                    // Angs(j-1,i) !!!
                    Angs(j,i) = Angs(j,i)+360;
                end //... if (180 < Angs(j,i)) & ((Angs(j,i)-Angs(j-1,i)) <= -180) then

                if (Angs(j,i) <= -180) & ((Angs(j,i)-Angs(j-1,i)) >= 180) then
                    // -568, -600, -186, -199 ---- for continuity instead of going
                    // this adjustment is necessary as we have already changed
                    // Angs(j-1,i) !!!
                    Angs(j,i) = Angs(j,i)-360;
                end //... if (Angs(j,i) < -180) & ((Angs(j,i)-Angs(j-1,i)) >= 180) then

            end //...  if (180 < Angs(j,i) & ((Angs(j,i)-Angs(j-1,i)) > 180) then

        end //j = 2 : SZ_Animation_Angs(1)
    end //i = 1 : SZ_Animation_Angs(2)

    Result = Angs;
    Msg = This_Fun_Name + Err_Msg;

endfunction
```

2.6 **File name:** Find_Min_Max.sci

Purpose: Finds the minimum and maximum values of an input matrix.

```
function [Result, Msg] = Find_Min_Max(x)
    //Used when errors are generated, indicating the name of the function
    //that generated the error
    This_Fun_Name = 'Find_Min_Max:';
    Err_Msg = '';

    //set initial values
    min_x = x(1);
    max_x = x(1);

    Mat_Indices = size(x);
    for i = 1 : Mat_Indices(1)     //rows of matrices x
        for j = 1 : Mat_Indices(2) //columns of matrices x
            //Find min and max x
            if (x(i,j) < min_x) then
```

```
                min_x = x(i,j);
            elseif (x(i,j) > max_x) then
                max_x = x(i,j);
            end
        end
    end

    Result = [min_x, max_x];
    Msg = This_Fun_Name + Err_Msg;

endfunction
```

3 HKM Kinematics

3.1 **File name:** Forward_Kinematics.sce

Purpose: Forward kinematics implementation. Displays a skeleton graphic of the machine and allows the user to select one of the 6 angles to control the robot in the forward sense. This is done by pressing a number from 1 to 6 on the keyboard, to select the angle. The up-down and left-right arrow keys are then used to increase or decrease that selected angle. It displays the angle values and end effector coordinates, and updates it as the user changes any of the robot angles. The user can also select how the coordinate frames are displayed by pressing the H key.

```
if ~exists('delete_and_clear') then
    global delete_and_clear;
    delete_and_clear = 1;
end //...if ~exists('delete_and_clear') then


if delete_and_clear then
    clc;
    xdel(winsid()); //Destroy all existing figures
    clearglobal();
    clear;
    lines(0); //disables vertical paging
    disp('Deleting all variables and clearing workspace... (at Forward_Kinematics)');

    global delete_and_clear;
    delete_and_clear = 0;
end




if (~exists ('Loaded_Functions')) then
    lines(0); //disables vertical paging
    // LOAD FUNCTIONS FROM DIRECTORY
    global Loaded_Functions;
    Loaded_Functions = 1;
    Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\Linkage Designer";
    getd(Funtion_Path);//Loads all functions from this directory into SCILAB
    Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics";
    getd(Funtion_Path);//Loads all functions from this directory into SCILAB
    Funtion_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics";
    getd(Funtion_Path);//Loads all functions from this directory into SCILAB
end //if (~exists ('Loaded_Functions')) then




global ik;
ik = 0; //In forward kinematics mode


if ~exists('LinkageTester') then
```

```
    disp("Creating Figure!");
    //Returns screen size in pixels, on this PC it s 1920 by 1200
    ScreenSizePX = get(0, "screensize_px");
    //Returns screen size in points, on this PC it s 960 by 600
    //ScreenSizePT = get(0, "screensize_pt");
    global L1a_len;
    global L1b_len;
    global L3_len;

    L1a_len = 1000;
    L1b_len = 1066.2938; //1066.2937611; //1000;
    L3_len =  181.2938; //181.2937611;//  101.2937611; //100;


    //global LinkageTester;
    drawlater();
    LinkageTester = figure(1); // Create figure having figure_id==1
    LinkageTester.figure_name = "Linkage Designer"
    LinkageTester.info_message = ""
    Win_Width = ScreenSizePX(3);
    Win_Height = ScreenSizePX(4);
    LinkageTester.figure_position = [0, 0];
    LinkageTester.figure_size = [Win_Width, Win_Height];

    close(figure(0));
else
    // Do nothing
end

drawlater();


global Light_Colours;
Light_Colours = [color("indian red"), color("pale green"), color("light blue")];
global Dark_Colours;
Dark_Colours = [color("red"), color("springgreen3"), color("blue")];
global Between_Colours;
Between_Colours = [color("violetred"), color("darkseagreen"), color("navyblue")];


// DisplayModes
if ~exists('Display_Modes') then
    global Display_Modes;
    Display_Modes = 1;
end //if ~exists(Display_Modes)


global Display_Switches;
Display_Switches = tlist(["Display_Switches", "Majority", ...
                    "Current_Triad", "Seventh_Triad"],  [], [], []);


// JOINT CREATION
global Initial_Joint_Angles;
Initial_Joint_Angles = [0  0   0;...
                        90 0   0;...
                        0  0   0;...
                        0  90  0;...
                        0  -90 0;...
                        0  90  0;...
                        0  -90 0]; // [90 0 0]


global Joint_Index;
Joint_Index = tlist(["Joint_Index", "Current", "Previous"], 1, 0);
// Joint has position and Axes to indicate orientation
Temp_J = tlist(["Joint", "Joint_Origin", "Joint_Angle", "Link_Length", "Triad_Data"], ...
                    tlist(["Joint_Origin", "X", "Y", "Z"], 0, 0, 0), ...
                    tlist(["Joint_Angle", "angXd", "angYd", "angZd"], 0, 0, 0), ...
                    0, ...
                    tlist(["Triad_Data", "Vec_Len", "Origin_Handles", "Link_Handle"], ...
                           4.5,...
                           tlist(["Origin_Handles", "X_dir", "Y_dir", "Z_dir"]), ...
                                  [], [], [] ), ...
                           []...
             );
```

```
global Kin_Model;
Kin_Model = tlist(["Kinematic_Model", "Joints"], list(0));

Link_Lengths = [0, 0, L1a_len,  L1b_len,   0, L3_len, 0];
// [F0-F1, F1-F2, F2-F3, F3-F4, F4-F5, F5-F6, F6-F7]


Size_Joint_Angles = size(Initial_Joint_Angles);
End_i = Size_Joint_Angles(1);
for i = 1 : End_i     // Initializing all joint angles data
    Temp_J.Joint_Angle.angXd = Initial_Joint_Angles(i, 1);
    Temp_J.Joint_Angle.angYd = Initial_Joint_Angles(i, 2);
    Temp_J.Joint_Angle.angZd = Initial_Joint_Angles(i, 3);;
    Temp_J.Link_Length = Link_Lengths(i);
    Temp_J.Triad_Data.Vec_Len = 0.3*L1a_len;
    Kin_Model.Joints(i) =  Temp_J;
end


    global RotMatAll;
    RotMatAll = list(0);
    global TransVecAll;
    TransVecAll = list(0);
    Create_Triad (End_i); //Creates triads



    // START UI CONTROL INITIALIZATION ************************************
    // Draw Text UI Controls
    ui_Xpos = 0.04;
    ui_Ypos = 0.88;
    ui_W = 0.1;
    ui_H = 0.05;
    ui_Col_Scale = 1/255;
    global ui_controls;
    for i = 1: 11
        ui(i) = uicontrol(LinkageTester, "style", "text");
    end

    for i = 1 : 11
        ui_Ypos = ui_Ypos - ui_H;
        ui(i).Units = "normalized";
        //ui(i).String = "Angle " + string(i) + ": " + "100.45";
        ui(i).Position = [ui_Xpos, ui_Ypos, ui_W, ui_H];
        ui(i).BackgroundColor = ui_Col_Scale*name2rgb("white");
        ui(i).ForegroundColor = ui_Col_Scale*name2rgb("black");
        ui(i).FontUnits = "normalized";
        ui(i).FontSize = 0.038;
        ui(i).FontWeight = "normal";
    end

    ui(1).String = "ANGLES-";
    ui(1).Position = ui(1).Position - [0.02, 0, 0, 0];

    for i = 1 : 6
        ui(i+1).String = string(i) + ":  " + ...
                    string(Kin_Model.Joints(i).Joint_Angle.angZd + ...
                    Initial_Joint_Angles(i, 3));
        ui(i+1).FontWeight = "normal";
    end

    ui(8).String = "COORDINATES-";
    ui(8).Position = ui(8).Position - [0.02, 0.05, 0, 0];
    ui(9).String = "X:  " + string(TransVecAll(6)(1));
    ui(9).Position = ui(9).Position - [0, 0.05, 0, 0];;
    ui(10).String = "Y:  " + string(TransVecAll(6)(2));
    ui(10).Position = ui(10).Position - [0, 0.05, 0, 0];;
    ui(11).String = "Z:  " + string(TransVecAll(6)(3));
    ui(11).Position = ui(11).Position - [0, 0.05, 0, 0];;

    ui_controls = ui;
    // END UI CONTROL INITIALIZATION ************************************



    LinkageTester.background = -2; //White
```

```
    LinkageTester.children($).axes_visible = ["on","on","on"];
    tmpBnd = (L1a_len + L1b_len + 1.5*L3_len);
    LinkageTester.children($).isoview = "off";
    LinkageTester.children($).view = "3d";
    LinkageTester.children($).data_bounds = ...
                [-tmpBnd, -tmpBnd, -4; tmpBnd, tmpBnd, tmpBnd+5];
    //[x_left,y_up,width,height], specifies upper left corner
    LinkageTester.children($).axes_bounds = [0.2, 0.1, 0.70, 1-0.2];
    LinkageTester.children($).margins = [0, 0, 0, 0]; //[0.125,0.125,0.125,0.125]
    LinkageTester.children($).grid = [0, 0, 0];
    LinkageTester.children($).font_color = -1;
    LinkageTester.children($).foreground = -1;
    LinkageTester.children($).hidden_axis_color = -1;
    LinkageTester.children($).font_size = 5;
    LinkageTester.children($).x_label.font_size = 5
    LinkageTester.children($).x_label.font_foreground = -1;
    LinkageTester.children($).x_label.text = "X Axis";
    LinkageTester.children($).y_label.font_size = 5;
    LinkageTester.children($).y_label.font_foreground = -1;
    LinkageTester.children($).y_label.text = "Y Axis";
    //LinkageTester.children($).y_label.position = [3200,-290];
    LinkageTester.children($).z_label.font_size = 5;
    LinkageTester.children($).z_label.font_foreground = -1;
    LinkageTester.children($).z_label.text = "Z Axis";
    LinkageTester.children($).z_label.font_angle = 270;
    LinkageTester.children($).box = "back_half";

    global My_View_Point;
    My_View_Point = 1; //Default is 3D
    global Current_View_Point;
    Current_View_Point = [71, 39.75];
    LinkageTester.children($).rotation_angles = Current_View_Point;

    LinkageTester.event_handler = 'ContEvntHdlr';
    LinkageTester.event_handler_enable = "on";
    toolbar(LinkageTester,'off');

    drawnow();
    scf(LinkageTester) // Raises a figure handle


Save_Fig_Kin_Model = 0;
if ~exists('Save_Fig_Kin_Model') then
    Save_Fig_Kin_Model = 1;
    // Save Figure and Kinematic model
    if (Save_Fig_Kin_Model == 1) then
        Location = 'D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\';
        File_Name = Location + 'FK.dat';
        save(File_Name, Kin_Model)
    end
end
```

3.2 **File name:** ContEvntHdlr.sce

Purpose: Runs in the background and updates the values of the control angles as the changes the selected angle. It also calls the routines to hide or display coordinate frames and update graphics and labels.

```
// disp("-Debug line 92: ContEvntHdlr")
function [Result, Msg] = ContEvntHdlr(win, x, y, ibut)

  // ALL GLOBAL VARIABLES
  global Joint_Index;
  global Kin_Model;
  global Initial_Joint_Angles;
  global TransVecAll;
  global Display_Switches;
  global My_View_Point;
  global ik;
```

```
  global L1a_len;
  global L1b_len;
  global L3_len;
  global Current_View_Point;
  global AngInc;  //Defined first here...
  AngInc = 5;
  // ***************************************************************
  // SOME LOCAL VARIABLES

  kin_scr_path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
  Script_Path = kin_scr_path + "Angle_Reference_Display.sce";

  This_Fun_Name = 'ContEvntHdlr:';
  Err_Msg = '';
  //disp('Inside Figure event handler!');
  RotLnk = %f;
  Num_of_Joints = size(Kin_Model.Joints);
  drawlater();

  if (ibut == 38)|(ibut == 39) then       //Right or Up keys pressed
      Kin_Model.Joints(Joint_Index.Current).Joint_Angle.angZd = ...
        pmodulo(Kin_Model.Joints(Joint_Index.Current).Joint_Angle.angZd + AngInc, 360);
      RotLnk = %t;


  elseif (ibut == 37)|(ibut == 40) then   //left or down keys pressed
      Kin_Model.Joints(Joint_Index.Current).Joint_Angle.angZd = ...
        pmodulo(Kin_Model.Joints(Joint_Index.Current).Joint_Angle.angZd - AngInc, 360);
      RotLnk = %t;


  elseif (ibut >= 49)&(ibut <= 54) then   //Numbers 1 to 6

      Temp_Index = ibut - 48;
      if (Temp_Index > Num_of_Joints) then
          disp('Invalid index selection, index greater number of joints!');
      else

          if(Joint_Index.Current ~= Temp_Index) then
              //disp('ContEvntHdlr - Line 67')
              Joint_Index.Previous = Joint_Index.Current;
              Joint_Index.Current = Temp_Index;
              Kin_Model.Joints(Joint_Index.Previous).Triad_Data.Vec_Len = 0.25*L1a_len;
              Kin_Model.Joints(Joint_Index.Current).Triad_Data.Vec_Len = 0.3*L1a_len;
              Create_Triad(Num_of_Joints); //Change triad focus; temporarily sets
              //the current angle to 1 and updates everything!
              //disp('Controlling Angle: ' + string(Temp_Index));
          end
      end
      // Call function to switch the visibility of current triad on and previous off
      Triad_Visibility();
      //UPDATES THE REFERENCE ANGLE GRAPHIC
      exec(Script_Path, -1);


  elseif (ibut == 104) then  //h - display selection of triads
      RotLnk = %t; // to update the triads
      //Increment Display modes immediately as it starts of at 1
      global Display_Modes;
      Display_Modes = Display_Modes + 1;
      disp('Display Mode: ' + string(Display_Modes));
      // Call function to switch the visibility of the triads on and off
      Triad_Visibility();


elseif (ibut == 105) then  //i - display different solution for
      // inverse kinematics
      RotLnk = %t; // to update the triads
      global Sol_Nr;
      Sol_Nr = Sol_Nr + 1;
      if (Sol_Nr > 8) then
          Sol_Nr = 1;
      end
      Tmp_Txt = "Displaying Inverse Kinematics Solution: " + string(Sol_Nr);
      disp(Tmp_Txt);
      End_J_Indx = 6;
      if (ik == 1) then  // in inverse kinematics!
```

```
        global All_Angles;
        for i = 1 : End_J_Indx
            //Setting the angles in the kinematic model to the ones we've
            //just calculated
            Kin_Model.Joints(i).Joint_Angle.angZd = All_Angles(Sol_Nr, i);
        end
    end



elseif (ibut == 114) then // "r" has been pressed, reset angles
    for i = 1 : Num_of_Joints
        Kin_Model.Joints(i).Joint_Angle.angZd = Initial_Joint_Angles(i, 3);
    end
    Joint_Index.Previous = Joint_Index.Current;
    Joint_Index.Current = 1; // This will redraw all graphics
    RotLnk = %t;
    disp("All angles have been reset!");
    disp('Controlling Angle: ' + string(1));


elseif (ibut == 118) then //v - change views, 2D-xy; 2D-xz; 2D-yz; 3D

    My_View_Point = My_View_Point + 1;

    if (1 == My_View_Point) then
        //Set to 3D view
        Current_View_Point = [71, 39.75];

    elseif (2 == My_View_Point) then
         //Set to 2D-xy view
         Current_View_Point = [0, 270];

    elseif (3 == My_View_Point) then
         //Set to 2D-xz view
         Current_View_Point = [90, 0];

    elseif (4 == My_View_Point) then
         //Set to 2D-yz view
         Current_View_Point = [90, 90];

    elseif (5 == My_View_Point) then
         My_View_Point = 1; // Set to 3D
         Current_View_Point = [71, 39.75];
    end  // checking if (1 == My_View_Point)

    Current_Fig = gcf();
    // Assign Current view point
    Current_Fig.children($).rotation_angles = Current_View_Point;
    Current_Fig.children($).box = "back_half";


elseif (ibut > 0)&(ibut < 255) then
     disp(ibut)
end  // Checking value of ibut


if (RotLnk == %t) then//To make sure that this routine only enters when the
  //arrow keys are pressed!!!
  RotLnk = %f;//To make sure that this routine only enters when the
  // arrow keys are pressed!!!
  if (Joint_Index.Current > Num_of_Joints) then
      Err_Msg = 'Invalid joint index, index greater number of joints!';
      disp(Err_Msg);

  else // Can proceed with changing the graphic configuration
      Update_Graphics_Model();
      //UPDATES THE REFERENCE ANGLE GRAPHIC
      exec(Script_Path, -1);

  end

end

drawnow();
drawlater();
```

```
  Result = 1;
  Msg = Err_Msg;
endfunction
```

3.3 **File name:** Update_Graphics_Model.sce

Purpose: Updates the graphics based on the current robot configuration.

```
// disp("Debug line 92: - Update Graphics_Model!")
function [Result, Msg]=Update_Graphics_Model()
    This_Fun_Name = 'Update_Graphics_Model:';
    Err_Msg = '';

    // ALL GLOBAL VARIABLES ****************************
    global Joint_Index;
    global Kin_Model;
    global Initial_Joint_Angles;
    global RotMatAll;
    global TransVecAll;
    global ui_controls

    // SPECIFICALLY FOR INVERSE KINEMATICS
    global ik_Circle;
    global ik_Square;
    global ik;


    drawlater();
    if (ik == 1) then // in inverse kinematics mode
        J_Index = 1; // Means redraw all graphics
    else
        // draw from current joint to last joint
        J_Index = Joint_Index.Current;
    end
    Num_of_Joints = size(Kin_Model.Joints);

    for i = J_Index : Num_of_Joints
        angX = Kin_Model.Joints(i).Joint_Angle.angXd;
        angY = Kin_Model.Joints(i).Joint_Angle.angYd;
        angZ = Kin_Model.Joints(i).Joint_Angle.angZd;

        gtext(i) = "Angle " + string(i) + ": " + string(angZ);


        Link_Length = Kin_Model.Joints(i).Link_Length;
        X_vec = Link_Length*[1; 0; 0];


        if (i == 1)
            RotMatAll(i) = Rotate_XYZ(angX, angY, angZ);
            TransVecAll(i) = RotMatAll(i)*X_vec;
            Trans_Vec = [0 0; 0 0; 0 0]';
            // Can Update the link vector here
            Kin_Model.Joints(i).Triad_Data.Link_Handle.data = Trans_Vec;

        else // from joints 2 to 6
            //Rx*Ry*Rz;
            RotMatAll(i) = RotMatAll(i-1)*Rotate_XYZ(angX, angY, angZ);
            //disp(RotMatAll(i))
            TransVecAll(i) = TransVecAll(i-1) + RotMatAll(i-1)*X_vec;
            //TransVecAll(i-1) + RotMatAll(i)*X_vec;
            //disp(TransVecAll(i))
            // Can Update the link vector here
            if (i == 7) then
                Kin_Model.Joints(i).Triad_Data.Link_Handle.data = ...
                            [[0; 0; 0], TransVecAll(i)]';
            else
                Kin_Model.Joints(i).Triad_Data.Link_Handle.data = ...
                            [TransVecAll(i-1), TransVecAll(i)]';
            end
            Trans_Vec = [TransVecAll(i), TransVecAll(i)]';
```

```
            // [TransVecAll(i-1), TransVecAll(i-1)]';
        end

        Origin = [0; 0; 0];
        Vec_Length = Kin_Model.Joints(i).Triad_Data.Vec_Len;
        X_vec = Vec_Length*[1; 0; 0];
        Y_vec = Vec_Length*[0; 1; 0];
        Z_vec = Vec_Length*[0; 0; 1];

        Unit_Vecs = RotMatAll(i)*[X_vec, Y_vec, Z_vec];
        X_vec = Unit_Vecs(:,1);
        Y_vec = Unit_Vecs(:,2);
        Z_vec = Unit_Vecs(:,3);


        // Updates the triad graphic handles -
        // translates the triad to correct position
        Kin_Model.Joints(i).Triad_Data.Origin_Handles.X_dir.data = ...
                                        [Origin X_vec]' + Trans_Vec;
        Kin_Model.Joints(i).Triad_Data.Origin_Handles.Y_dir.data = ...
                                        [Origin Y_vec]' + Trans_Vec;
        Kin_Model.Joints(i).Triad_Data.Origin_Handles.Z_dir.data = ...
                                        [Origin Z_vec]' + Trans_Vec;
        //Save the origin coordinates
        Kin_Model.Joints(i).Joint_Origin.X = Trans_Vec(1, 1);
        Kin_Model.Joints(i).Joint_Origin.Y = Trans_Vec(1, 2);
        Kin_Model.Joints(i).Joint_Origin.Z = Trans_Vec(1, 3);

    end

    //*****************************************************************
    //UPDATE ANGLE TEXT
    for i = 1 : 6

        ui_controls(i+1).String = string(i) + ": " +  sprintf('%5.2f', ...
                       pmodulo(Kin_Model.Joints(i).Joint_Angle.angZd + ...
                       Initial_Joint_Angles(i, 3), 360));
        // string(pmodulo (Kin_Model.Joints(i).Joint_Angle.angZd + ...
        // Initial_Joint_Angles(i, 3), 360));
        ui_controls(i+1).FontWeight = "normal";
        ui_controls(i+1).FontSize = 0.038;

    end

    // UPDATE POSITION TEXT
    ui_controls(9).String = "X:  " + sprintf('%5.2f', TransVecAll(6)(1));
    ui_controls(10).String = "Y:  " + sprintf('%5.2f', TransVecAll(6)(2));
    ui_controls(11).String = "Z:  " + sprintf('%5.2f', TransVecAll(6)(3));
    ui_controls(9).FontWeight = "normal";
    ui_controls(10).FontWeight = "normal";
    ui_controls(11).FontWeight = "normal";



    drawnow();
    drawlater();
    drawlater();

    Result = 1;
    Msg = This_Fun_Name + Err_Msg;

endfunction
```

3.4 **File name:** Create_Triad.sce

Purpose: Creates the coordinate frame axes for each joint and the end effector.

```
//CREATE GRAPHIC TRIAD ****************************
// disp("-Debug line 92: Create_Triad")
```

```
function [Result, Msg] = Create_Triad (Num_Triads)
    This_Fun_Name = 'Create_Triad:';
    Err_Msg = '';

    //GLOBAL VARIABLES *********************************
    global Joint_Index;
    global Kin_Model;
    global ui_controls;
    global Light_Colours;
    global Dark_Colours;
    global Between_Colours;
    drawlater();

    Origin = [0 0 0];

    Vec_End = [1,0,0;0,1,0;0,0,1];

    Line_Thickness = 4;
    Arrow_Size_Factor = 0.375;


    for i = 1 : Num_Triads
        Vec_Length = Kin_Model.Joints(i).Triad_Data.Vec_Len;
        for j = 1 : 3
            // We select the colour based on the current selected joint
            if (i ~= Joint_Index.Current) then
                //Light_Colours + Dark_Colours initialized in Forward_Kinematics
                Current_Colour = Light_Colours(j);
            else //if (i ~= Joint_Index.Current)
                Current_Colour = Dark_Colours(j);
            end //if (i ~= Joint_Index.Current)

            // If the triad vectors exist we want to change the colour
            if ( size(Kin_Model.Joints(i).Triad_Data.Origin_Handles) == 4 ) then
                //means we have handles to all triad graphics
                //disp("Changing Triad Focus!")
                Polyline_Hndl = Kin_Model.Joints(i).Triad_Data.Origin_Handles(j+1);
                Polyline_Hndl.foreground = Current_Colour;
                Vec = Polyline_Hndl.data;

                // Unitize then scale vectors!
                Scaled_Vec = [Vec(2,1) - Vec(1,1), Vec(2,2) - Vec(1,2), ...
                              Vec(2,3) - Vec(1,3)];
                Old_Length = ( Scaled_Vec(1)^2 + Scaled_Vec(2)^2 + ...
                              Scaled_Vec(3)^2 )^0.5;
                Scaled_Vec = (Vec_Length/Old_Length)*Scaled_Vec;
                Vec(2, :) = [Scaled_Vec(1) + Vec(1,1), Scaled_Vec(2) + ...
                             Vec(1,2), Scaled_Vec(3) + Vec(1,3)];
                Polyline_Hndl.data = Vec;


            else // if ( size(Kin_Model.Joints(i).Triad_Data.Origin_Handles) == 4 )
                // If not we create them!
                // But first draw the links
                if (j == 1) then//create link first
                    Link_Length = Kin_Model.Joints(i).Link_Length;
                    x = Link_Length*[Origin(1), Vec_End(1, 1)];
                    y = Link_Length*[Origin(2), Vec_End(1, 2)];
                    z = Link_Length*[Origin(3), Vec_End(1, 3)];

                    param3d(x, y, z);
                    Polyline_Hndl = get("hdl");
                    Polyline_Hndl.polyline_style = 1;
                    // 1-Solid; 3-Dash dot style; 4-ends in Arrow
                    Polyline_Hndl.line_style = 1;
                    Polyline_Hndl.thickness = 2; //Line_Thickness;
                    Polyline_Hndl.mark_style = 9;
                    Polyline_Hndl.mark_mode = "on";
                    Polyline_Hndl.foreground = -1; //Black //Current_Colour;
                    //get handle on current entity (here the polyline entity)
                    Kin_Model.Joints(i).Triad_Data.Link_Handle = Polyline_Hndl;
                    if (i == 7) then
                        Kin_Model.Joints(7).Triad_Data.Link_Handle.visible = "off";
                    end //if (i == 7)
                end //if (j == 1)

                // Now create the Triad vectors
```

```
                x = Vec_Length*[Origin(1), Vec_End(j, 1)];
                y = Vec_Length*[Origin(2), Vec_End(j, 2)];
                z = Vec_Length*[Origin(3), Vec_End(j, 3)];

                param3d(x, y, z);
                Polyline_Hndl = get("hdl");
                Polyline_Hndl.polyline_style = 4;
                // 1-Solid; 3-Dash dot style; 4-ends in Arrow
                Polyline_Hndl.line_style = 1;
                Polyline_Hndl.thickness = Line_Thickness;
                Polyline_Hndl.arrow_size_factor = Arrow_Size_Factor;
                Polyline_Hndl.mark_mode = "off";
                Polyline_Hndl.foreground = Current_Colour;
                Kin_Model.Joints(i).Triad_Data.Origin_Handles(j+1) = Polyline_Hndl;

            end //if ( size(Kin_Model.Joints(i).Triad_Data.Origin_Handles) == 4 )
        end //for j = 1 : 3
    end // for i = 1 : Num_Triads

    // Adjust colour of Seventh Triad (and maybe style)
    for j = 1 : 3 // seventh triad
        Polyline_Hndl = Kin_Model.Joints(7).Triad_Data.Origin_Handles(j+1);
        Polyline_Hndl.foreground = Between_Colours(j);
    end // for j = 1 : 3 -- seventh triad


    // All triad vetors have been created, now we just need to get the
    // orientation right!
    Temp_Joint_Current = Joint_Index.Current;
    Joint_Index.Current = 1; //Temporary reset, the algorithm is
    // efficient it only corrects joints
    // > i when i has been adjusted, by setting it to 1 we force a
    // redraw and update!
    Update_Graphics_Model();
    Joint_Index.Current = Temp_Joint_Current; // set to original value


    Result = [1]; //Successful!
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.5 **File name:** Triad_Visibility.sce

Purpose: Allows the user to select how the coordinate frames are displayed, this is done by pressing the H key repeatedly.

```
// ADJUSTS GRAPHIC TRIAD VISIBILITY BASED ON SELECTION MODE
// disp("-Debug line 92: Triad_Visibility")
function [Result, Msg] = Triad_Visibility ()
    This_Fun_Name = 'Triad_Visibility:';
    Err_Msg = '';

    //GLOBAL VARIABLES *********************************
    global Joint_Index;
    global Kin_Model;
    global Display_Switches;
    global Display_Modes;
    drawlater();

    if (Display_Modes == 1) then
        // Show all Triads
        // [triadX; triadY; triadZ; link];
        Display_Switches.Majority = ["on"; "on"; "on"; "on"];
        Display_Switches.Current_Triad = ["on"; "on"; "on"; "on"];
        Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "off"];

    elseif (Display_Modes == 2) then
        // Show Current Triad and End effector Triad, with linkage
        // [triadX; triadY; triadZ; link];
        Display_Switches.Majority = ["off"; "off"; "off"; "on"];
```

```
    Display_Switches.Current_Triad = ["on"; "on"; "on"; "on"];
    Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "off"];

elseif (Display_Modes == 3) then
    // Show Current Triad only, with linkage
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["off"; "off"; "off"; "on"];
    Display_Switches.Current_Triad = ["on"; "on"; "on"; "on"];
    Display_Switches.Seventh_Triad = ["off"; "off"; "off"; "off"];

elseif (Display_Modes == 4) then
    // Show End Effector Triad only, with linkage
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["off"; "off"; "off"; "on"];
    Display_Switches.Current_Triad = ["off"; "off"; "off"; "on"];
    Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "off"];

elseif (Display_Modes == 5) then
    // Show End Effector Triad only, with linkage -
    // angle reference will be shown in forward kinematics
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["off"; "off"; "off"; "on"];
    Display_Switches.Current_Triad = ["on"; "off"; "off"; "on"];
    Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "off"];

elseif (Display_Modes == 6) then
    // Show Linkage only No triads
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["off"; "off"; "off"; "on"];
    Display_Switches.Current_Triad = ["off"; "off"; "off"; "on"];
    Display_Switches.Seventh_Triad = ["off"; "off"; "off"; "off"];

elseif (Display_Modes == 7) then
    // Show End Effector Triad only with translation vector.
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["off"; "off"; "off"; "off"];
    Display_Switches.Current_Triad = ["off"; "off"; "off"; "off"];
    Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "on"];

elseif (Display_Modes >= 8) then
    Display_Modes = 1;
    // Show all Triads
    // [triadX; triadY; triadZ; link];
    Display_Switches.Majority = ["on"; "on"; "on"; "on"];
    Display_Switches.Current_Triad = ["on"; "on"; "on"; "on"];
    Display_Switches.Seventh_Triad = ["on"; "on"; "on"; "off"];

end


for i = 1 : 6//end_i
    Kin_Model.Joints(i).Triad_Data.Origin_Handles.X_dir.visible = ...
                                Display_Switches.Majority(1);
    Kin_Model.Joints(i).Triad_Data.Origin_Handles.Y_dir.visible = ...
                                Display_Switches.Majority(2);
    Kin_Model.Joints(i).Triad_Data.Origin_Handles.Z_dir.visible = ...
                                Display_Switches.Majority(3);
    Kin_Model.Joints(i).Triad_Data.Link_Handle.visible = ...
                                Display_Switches.Majority(4);
end

Kin_Model.Joints(Joint_Index.Current).Triad_Data.Origin_Handles.X_dir.visible = ...
                                Display_Switches.Current_Triad(1);
Kin_Model.Joints(Joint_Index.Current).Triad_Data.Origin_Handles.Y_dir.visible = ...
                                Display_Switches.Current_Triad(2);
Kin_Model.Joints(Joint_Index.Current).Triad_Data.Origin_Handles.Z_dir.visible = ...
                                Display_Switches.Current_Triad(3);
Kin_Model.Joints(Joint_Index.Current).Triad_Data.Link_Handle.visible = ...
                                Display_Switches.Current_Triad(4);


Kin_Model.Joints(7).Triad_Data.Origin_Handles.X_dir.visible = ...
                                Display_Switches.Seventh_Triad(1);
Kin_Model.Joints(7).Triad_Data.Origin_Handles.Y_dir.visible = ...
                                Display_Switches.Seventh_Triad(2);
Kin_Model.Joints(7).Triad_Data.Origin_Handles.Z_dir.visible = ...
                                Display_Switches.Seventh_Triad(3);
```

```
    Kin_Model.Joints(7).Triad_Data.Link_Handle.visible = ...
                                    Display_Switches.Seventh_Triad(4);

    Result = 1;
endfunction
```

3.6 **File name:** Highlight_Triad.sci

Purpose: Highlights a specific triad and greys out the others.

```
//HIGHLIGHT GRAPHIC TRIAD ****************************
function [Result, Msg] = Highlight_Triad (triad_data_in)
    This_Fun_Name = 'Highlight_Triad:';
    Err_Msg = '';
    //Check input data for correct Tlist structure
    //tlist(["Triad_Data", "Vec_Len", "Vec_Handles"]
    Check_Input = typeof(triad_data_in);
    //disp(triad_data_in)

    if ~strcmp(Check_Input, "Triad_Data") then //returns 0 is true hence the not!
        drawlater();
        global Joint_Index;


        Red = name2rgb("red");
        Green = name2rgb("green");
        Blue = name2rgb("blue");
        Colours = [Red; Green; Blue];
        Grey = name2rgb("lightslategray");
        for j = 1 : 3
            triad_data_in.Vec_Handles.X_dir.foreground = Grey;
            triad_data_in.Vec_Handles.Y_dir.foreground = Grey;
            triad_data_in.Vec_Handles.Z_dir.foreground = Grey;
        end

    else // Wrong input data type
        Handles = []; //Empty vector of handles
        Result = [0]; //Unsuccessful!
        Err_Msg = "Function input data has to be a Triad_Data tlist data structure!";
    end

    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.7 **File name:** Rotate_XYZ.sce

Purpose: Rotates a given frame about the X, Y then Z axes in that order.

```
// Rotation Matrix - ORDER X, Y, Z
// disp("-Debug line 173: A_sin_cos_d")
function [Rmat, Result, Msg] = Rotate_XYZ(AngX, AngY, AngZ)
    This_Fun_Name = "Rotate_XYZ:";
    Err_Msg = "";

        Rx = [1 0          0; ...
              0 cosd(AngX) -sind(AngX); ...
              0 sind(AngX) cosd(AngX)];

        Ry = [cosd(AngY)  0 sind(AngY) ; ...
              0           1 0          ; ...
              -sind(AngY) 0 cosd(AngY)];
```

```
        Rz = [cosd(AngZ) -sind(AngZ) 0; ...
              sind(AngZ)  cosd(AngZ) 0; ...
              0           0          1];


    Rmat = Rx*Ry*Rz;

    Result = 1;
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.8 **File name:** Rotate_ZYX.sce

Purpose: Rotates a given frame about the Z, Y then X axes in that order.

```
// Rotation Matrix reverse order Z, Y, X
// disp("-Debug line 173: A_sin_cos_d")
function [Rmat, Result, Msg] = Rotate_ZYX(AngX, AngY, AngZ)
    This_Fun_Name = "Rotation_Matrix ZYX:";
    Err_Msg = "";

        Rx = [1 0          0; ...
              0 cosd(AngX) -sind(AngX); ...
              0 sind(AngX) cosd(AngX)];

        Ry = [cosd(AngY)  0 sind(AngY) ; ...
              0           1 0          ; ...
              -sind(AngY) 0 cosd(AngY)];

        Rz = [cosd(AngZ) -sind(AngZ) 0; ...
              sind(AngZ)  cosd(AngZ) 0; ...
              0           0          1];


    Rmat = Rz*Ry*Rx;

    Result = 1;
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.9 **File name:** Inverse_Kinematics.sce

Purpose: Calculates the joint angles for a given pose. It calls the forward kinematics script to generate the graphics if it doesn't already exist. The user must then put the robot model in a particular desired pose. So at this point we already know the angles used, and we have a reachable end effector pose. The inverse kinematics script is then run again, it takes the end effector pose and calculates all joint angle solution sets that will give the robot that particular end effector pose. The user can check the solution sets by setting 'Display_Check = 1' close to the end of the script file, and compare the angles displayed in the graphic window to the calculated solutions. It must equal one set.

```
// INVERSE KINEMATICS
// CALLS FORWARD KINEMATICS FOR FIGURE INITIALIZATION AND GRAPHICS
global ik;
ik = 1;

if ~exists('LinkageTester') then // Execute Forward_Kinematics
    Script_Path = ...
         "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\" + ...
         "Forward_Kinematics.sce";
    exec(Script_Path, -1);
```

```
    disp("Executing Inverse Kinematics!");

    //*****************************************************************
    // ADJUST TRIAD VISIBILITY
    // Display_Modes = 4 - Show Main Triad only, with linkage
    // Display_Modes = 5 - Show Linkage only No triads
    // Display_Modes = 6 - Show Main Triad only with translation vector
    //Triad_Visibility ();

end // if ~exists('LinkageTester')


drawlater();
if ~exists('Ref_Triad') then // Create Reference Triad
    //Create A Reference Triad
    global Ref_Triad;
    Ref_Triad = tlist(["Ref_Triad", "X_vec", "Y_vec", "Z_vec"], [], [], []);

    global L1a_len;
    Vec_Length = 0.3*L1a_len;
    Line_Thickness = 4;
    Arrow_Size_Factor = 0.375;
    Origin = [0 0 0];
    Vec_End = [1,0,0;0,1,0;0,0,1];

    for j = 1 : 3 // Creating Reference Triad Vectors
        // Now create the Triad vectors
        x = Vec_Length*[Origin(1), Vec_End(j, 1)];
        y = Vec_Length*[Origin(2), Vec_End(j, 2)];
        z = Vec_Length*[Origin(3), Vec_End(j, 3)];

        param3d(x, y, z);
        Polyline_Hndl = get("hdl");
        Polyline_Hndl.polyline_style = 4;
        // 1-Solid; 3-Dash dot style; 4-ends in Arrow
        Polyline_Hndl.line_style = 1;
        Polyline_Hndl.thickness = Line_Thickness;
        Polyline_Hndl.arrow_size_factor = Arrow_Size_Factor;
        Polyline_Hndl.foreground = Dark_Colours(j);
        Tmp_Handles(j+1) = Polyline_Hndl;
    end // for j = 1 : 3  // Creating Reference Triad Vectors
    Ref_Triad = Tmp_Handles;
    Ref_Triad(2).visible = "off";
    Ref_Triad(3).visible = "off";
    Ref_Triad(4).visible = "off";
end // if ~exists('Ref_Triad') -- // Create Reference Triad

global Current_View_Point;
LinkageTester.children($).rotation_angles = Current_View_Point;
LinkageTester.children($).box = "back_half";
drawnow();


////////////////////////////////////////////////////////////////////////
// *******************************
// These variables exist in memory
// LinkageTester - figure handle
// DisplayModes
// global Display_Modes - just an integer;
// global Display_Switches;
// Display_Switches = tlist(["Display_Switches", "Majority",...
//             "Current_Triad", "Seventh_Triad"],  [], [], []);
// global Light_Colours;
// global Dark_Colours;
// global RotMatAll;
// global TransVecAll;
//
//Initial_Joint_Angles = [0  0   0;...
//                        90 0   0;...
//                        0  0   0;...
//                        0  90  0;...
//                        0  -90 0;...
//                        0  90  0;...
//                        0  -90 0];


// Link_Lengths = [0, 0,  L1a_len,  L1a_len,   0,   L3_len, 0];
```

```
//         // [F0-F1, F1-F2,   F2-F3, F3-F4,  F4-F5,  F5-F6,  F6-F7]
global L1a_len;
global L1b_len;

Rtp = RotMatAll(7);
Ttp = TransVecAll(7);

// The 6th coordinate frame can be determined from the Tool frame
// The x-axis of the 6th coordinate frame is in the opposite direction
// as the z-axis of the tool frame
// The y-axis of the 6th coordinate frame is in the same direction
// as the y-axis of the tool frame
// The z-axis of the 6th coordinate frame is in the same direction
// as the x axis of the tool frame
 RotMatAll(6) = [-RotMatAll(7)(:, 3), RotMatAll(7)(:, 2), RotMatAll(7)(:, 1)];


// We've redesigned the wrist so that it is spherical, and all wrist axes
// are concurrent. This simplifies the inverse kinematics, and we can
// find the wrist origin easier as we can now decouple the wrist orientation
// from the wrist (origin) position!

// The Wrist origin whose coordinates we're trying to find lies on a plane
// perpendicular to the XY plane passing through the Z axis (due to the
// orientation of the first 3 axes). The equation of this plane is y = Mx,
// as z is independent of x and y; y is dependent on x i.e. for each x there
// is only 1 y; and for each x there is an infintie number of z;

// First we need to find the Wrist origin, which is the intersection of the
// 3 wrist axes, this is simple we translate the given tool point back along
// its x-axis by -L2_len.


// We multiply the vector [-L2_len; 0; 0] with the rotation matrix Rtp which
// is the orientation matrix of the tool frame, that resulting vector is
// then added to the tool point coordinates given, to give the origin of the
// 6th coordinate frame CF6 or CF6_origin (spherical wrist
// origin).

// The first condition is that the distance from the base origin to the
// CF6_origin must be less than or equal to 2*L1_len, if not then we cannot
// reach that position. if it equals 2*L1_len then there's only 2 possible
// solutions for the set of angles ang1, ang2 and ang3 that position the
// wrist at that point. There would be 2 solutions for ang1 and 1 set of
// ang2 and ang3 for each ang1. There could still be multiple solutions to
// the angles ang4, ang5 and ang6 that give the required wrist orientation
// in the inverse kinematics.



CF6_origin = Rtp*[-L3_len; 0; 0] + Ttp;

My_Marker.data = [CF6_origin, CF6_origin]';

// Now that we have the origin of the 6th coordinate frame we can
// calculate the first 3 angles that position the wrist at that point.


drawlater();

global All_Angles;

Dist1 = (CF6_origin(1)^2 + CF6_origin(2)^2 + CF6_origin(3)^2)^0.5;


// Test length make sure its less than L1a_len + L1b_len
if (Dist1 > L1a_len + L1b_len) then
    disp("No solution exists as the linkage cannot reach that point!");
    disp("Wrist Origin to Base Origin is greater than 2x Link1 length.");

else // Now we can solve for all coordinates ...
    //Keeping track of solutions, there should be 8 sets of solutions,
    //creating storage space.
    Tot_Num_Sols = 8;
    for j = 1 : Tot_Num_Sols
        //'u' - unsolved; 's' - solved; 'i' - infinite number of solutions
        Angle_solved(j, :) = ['u', 'u', 'u', 'u', 'u', 'u'];
```

```
    All_Angles(j, :) = [0, 0, 0, 0, 0, 0,];
end

// Now we need to calculate the Knee coordinates.
// For each set of wrist coordinates there will be 2 values of Ang1.
// Ang1 is determined from the x-y pair, the projection onto the base
// frame. Will use function  A_sin_cos_d  to determine the
// angle, but first we need to find the sin and cos, given by the
// x-y pair.
Dist1 = (CF6_origin(1)^2 + CF6_origin(2)^2)^0.5;
if Dist1 == 0 then
    // Means the wrist origin lies on the z-axis...
    // There are an infinite number of solutions for angle 1, and
    // there's no need for any projection onto xz-axis
    // as these coordinates will not change.
    for i = 1:Tot_Num_Sols
       Angle_solved(i, 1) = 'i'; //Solved
       // Set to 0 or previous angle for convenience
       All_Angles(i, 1) = 0;
    end
    disp("There are an infinte number of solutions!");

else
    // SINGULARITY FOR ANG1 IS WHEN THE WRIST ORIGIN LIES ON THE
    // BASE FRAME Z AXIS - THIS IMPLIES INFINTE NUMBER OF
    // SOLUTIONS FOR ANG 1!
    CAng1a = CF6_origin(1)/Dist1;
    SAng1a = CF6_origin(2)/Dist1;
    Ang1a = pmodulo(A_sin_cos_d(SAng1a, CAng1a), 360);
    Ang1b = pmodulo(Ang1a + 180, 360);

    //We've Solved for Ang1
    for i = 1:(Tot_Num_Sols/2)
        Angle_solved(i, 1) = 's'; //Solved
        All_Angles(i, 1) = Ang1a;
    end
    for i = (Tot_Num_Sols/2 + 1):Tot_Num_Sols
        Angle_solved(i, 1) = 's'; //Solved
        All_Angles(i, 1) = Ang1b;
    end

end // Checking if Dist1 = (CF6_origin(1)^2 + CF6_origin(2)^2)^0.5 == 0

// Since we measured the angle from the positive x-axis to the line,
// we rotate the position vector by -Ang1 so that the linkage model
// lies on the xz plane, this makes calculating Ang2 and Ang3 easier,
// once we're done we rotate back.

R1 = Rotate_ZYX(0, 0, -Ang1a); // Rotate onto xz plane to solve easier
CF6_origin_xz = R1*CF6_origin; //The y coordinates should be 0;

// We can solve for the elbow or knee position, the knee coordinates
// are actually the intersection of 2 circles one centred at the base
// origin and the other at the wrist coordinates we've just rotated
// to lie on the xz-plane of the base frame.

// The equation of a circle lying at the centre of the base origin on
// the xz-plane, with radius L1_len is x^2 + z^2 = L1_len^2; the equation
// of the other circle with rotated wrist origin on xz plane is (remember
// the y coordinate is now 0)...

// We determine the actual angles by computing the sin and cosine of the
// angles and then passing them to the A_sin_cos_d function. Do this
// before rotating back. Now rotate the solutions found on the XZ plane
// back to the original plane;
// R1 = Rotation_Matrix(0, 0, Ang1);
// Elbow_2a = R1*[x_1; 0; z_1];
// Elbow_2b = R1*[x_2; 0; z_2];

//Find the angle  CF6_origin_xz to Base_Origin to +X_Axis
Hypotenuse = (CF6_origin_xz(1)^2 + CF6_origin_xz(3)^2)^0.5;
tmp_Sind = CF6_origin_xz(3)/Hypotenuse;
tmp_Cosd = CF6_origin_xz(1)/Hypotenuse;
tmp_Ang = A_sin_cos_d(tmp_Sind, tmp_Cosd);

// Now rotate the line  CF6_origin_xz to Base_Origin onto the X-Axis
// about the Y-Axis, this will make the 2 solutions for the knee
```

```
// coordinates mirrored about the X-Axis...
// makes solving geometry simpler!!!
R2 = Rotate_ZYX(0, tmp_Ang, 0);
//Now both the X and Y coordinates should be 0;
CF6_origin_xz = R2*CF6_origin_xz;


// From Cosine rule
CAng2a = ((L1a_len^2)+(Hypotenuse^2)-(L1b_len^2))/(2*L1a_len*Hypotenuse);
// From Heorn's Formula - Calculation of triangle area from side lengths
SP = (L1a_len + L1b_len + Hypotenuse)/2;
Triangle_Area = (SP*(SP - L1a_len)*(SP - L1b_len)*(SP - Hypotenuse))^0.5;
SAng2a = Triangle_Area/(0.5*L1a_len*Hypotenuse);
x_1 = L1a_len*CAng2a;
z_1 = L1a_len*SAng2a;
// ************************************************************
// Now find the angles, remember that in the joints coordinate
// frame (not the base frame in which we're calculating) the
// joint angle is between the x-axis vector of the adjacent joint
// coordinate frames.
// will have to check if x_1 == []; x_2 == []; z_1 == []; z_2 == [];


Ang2a =  pmodulo(A_sin_cos_d(SAng2a, CAng2a) + tmp_Ang, 360);
Ang2b =  pmodulo(-A_sin_cos_d(SAng2a, CAng2a) + tmp_Ang, 360);

//Elbow or Knee Joint Angle
V1 = [x_1; 0; z_1];
V1_mag = (V1'*V1)^0.5;
V1 = (1/V1_mag)*V1;
V2 = [CF6_origin_xz(1)-x_1; 0; CF6_origin_xz(3)-z_1];
V2_mag = (V2'*V2)^0.5;
V2 = (1/V2_mag)*V2;

Cross_V1_V2 = CrossMult(V1, V2);
// magnitude of cross product of unit vectors to give sine of
// angle between em  //(CF6_origin_xz(3) - z_1)/L1_len;;
SAng3a =  (Cross_V1_V2*Cross_V1_V2')^0.5;
// Dot product between unit vectors to give cosine of
// angle between em  //(CF6_origin_xz(1) - x_1)/L1_len;
CAng3a =  V1'*V2;

//z_1 > 0) so
Ang3a =  360 - A_sin_cos_d(SAng3a, CAng3a);
Ang3b =  A_sin_cos_d(SAng3a, CAng3a);

///// ANGs 2 TO 5
if (Ang2a == [])&(Ang2b == [])&(Ang3a == [])&(Ang3b == []) then
        // No solutions

else
    for i = 1:Tot_Num_Sols
        Angle_solved(i, 2) = 's'; //Solved
        Angle_solved(i, 3) = 's'; //Solved
    end

    for i=1:(Tot_Num_Sols/4)
        All_Angles(i, 2) = Ang2a;
        All_Angles(i, 3) = Ang3a;
    end

    for i=(Tot_Num_Sols/4 + 1):(Tot_Num_Sols/2)
        All_Angles(i, 2) = Ang2b;
        All_Angles(i, 3) = Ang3b;
    end

    for i=(Tot_Num_Sols/2 + 1):(3*Tot_Num_Sols/4)
        //pmodulo(ang, 360);
        All_Angles(i, 2) = pmodulo(180 - Ang2a, 360);
        All_Angles(i, 3) = 360 - Ang3a; //pmodulo(180 + Ang3a, 360);
    end

    for i=(3*Tot_Num_Sols/4 + 1):Tot_Num_Sols
        All_Angles(i, 2) = pmodulo(180 - Ang2b, 360);
        All_Angles(i, 3) = 360 - Ang3b; //pmodulo(180 + Ang3b, 360);
    end
```

```
// SINCE THE INITIAL ANGLES CHANGED THIS WRIST ORIENTATION ALSO CHANGES
// Now we solve for the wrist orientation angles.
// Symbolic multiplication of the last 3 sets of angles for joint
// angles yields...


// FOR CURRENT SPHERICAL WRIST
//!-s5.s6           -s5.c6           c5     !
//!                                          !
//!s4.c6+c4.c5.s6   -s4.s6+c4.c5.c6  c4.s5  !
//!                                          !
//!-c4.c6+s4.c5.s6  c4.s6+s4.c5.c6   s4.s5  !

for j = 1: 2: Tot_Num_Sols
    // Find reverse order multiplication matrices for
    // pre-multiplying and separating out the matrices responcible
    // for wrist orientation from wrist prigin position.
    RotMat321 = [1 0 0; 0 1 0; 0 0 1];
    //RotMat123 = [1 0 0; 0 1 0; 0 0 1];
    for i = 1 : 3
        angX = -Kin_Model.Joints(i).Joint_Angle.angXd;
        angY = -Kin_Model.Joints(i).Joint_Angle.angYd;
        angZ = -All_Angles(j, i);
        RotMat321 = Rotate_ZYX(angX, angY, angZ)*RotMat321;   //Note order
        //RotMat123 = RotMat123*Rotate_XYZ(-angX, -angY, -angZ);
    end

    RotMat456 = RotMat321*RotMatAll(6);
    // Using RotMat456 we can now solve for angles 4,5 and 6!

    //!-s5.s6           -s5.c6           c5     !
    //!                                          !
    //!s4.c6+c4.c5.s6   -s4.s6+c4.c5.c6  c4.s5  !
    //!                                          !
    //!-c4.c6+s4.c5.s6  c4.s6+s4.c5.c6   s4.s5  !

    // From the symbolic matirx above we can solve for ang5 as
    // we have c5 ( sind(Ang5) )...
    CAng5 = RotMat456(1, 3);
    if abs(CAng5) == 1 then
        SAng5a = 0;
        SAng5b = 0;
        if (CAng5 == 1) then
            Ang5a = 0;
            Ang5b = 0;
            // -> RotMat456(2, 1) = s4.c6+c4.c5.s6  = ...
            //            s4.c6+c4.s6 = sin(Ang4 + Ang6)
            // -> RotMat456(2, 2) = -s4.s6+c4.c5.c6 = ...
            //            -s4.s6+c4.c6 = cos(Ang4 + Ang6)
            Ang4_plus_Ang6 = A_sin_cos_d(RotMat456(2, 1), RotMat456(2, 2));
            Ang4a = Ang4_plus_Ang6
            Ang4b = 0;
            Ang6a = 0;
            Ang6b = Ang4_plus_Ang6;

        elseif (CAng5 == -1) then
            Ang5a = 180;
            Ang5b = 180;
            // -> RotMat456(2, 1) = s4.c6+c4.c5.s6 = ...
            //                s4.c6-c4.s6 = sin(Ang4 - Ang6)
            // -> RotMat456(2, 2) = -s4.s6+c4.c5.c6 = ...
            //                -s4.s6-c4.c6 = - cos(Ang4 - Ang6)
            Ang4_minus_Ang6 = A_sin_cos_d(RotMat456(2, 1), -RotMat456(2, 2));
            Ang4a = Ang4_minus_Ang6;
            Ang4b = 0;
            Ang6a = 0;
            Ang6b = Ang4_minus_Ang6;

        end
        // There are an infinite number of solutions for angs 4
        // and 6 as Axis 4 and Axis 6 are collinear
        disp('There are an infinite number of solutions'+ ...
            ' for angs 4 and 6 as Axis 4 and Axis 6 are collinear!');
        // Leave at previous angle or set to new


    else
```

```
                // FIRST SOLUTION
                SAng5a = (1 - CAng5^2)^0.5;
                CAng4a = RotMat456(2, 3)/SAng5a;
                SAng4a = RotMat456(3, 3)/SAng5a;
                SAng6a = -RotMat456(1, 1)/SAng5a;
                CAng6a = -RotMat456(1, 2)/SAng5a;

                Ang4a =  A_sin_cos_d(SAng4a, CAng4a);
                Ang5a =  A_sin_cos_d(SAng5a, CAng5);
                Ang6a =  A_sin_cos_d(SAng6a, CAng6a);


                // SECOND SOLUTION
                SAng5b = -(1 - CAng5^2)^0.5;
                CAng4b = RotMat456(2, 3)/SAng5b;
                SAng4b = RotMat456(3, 3)/SAng5b;
                SAng6b = -RotMat456(1, 1)/SAng5b;
                CAng6b = -RotMat456(1, 2)/SAng5b;

                Ang4b =  A_sin_cos_d(SAng4b, CAng4b);
                Ang5b =  A_sin_cos_d(SAng5b, CAng5);
                Ang6b =  A_sin_cos_d(SAng6b, CAng6b);

            end


            Angle_solved(j, 4:6)   = ['s', 's', 's']; //Solved
            Angle_solved(j+1, 4:6) = ['s', 's', 's']; //Solved

            All_Angles(j, 4:6)   = [Ang4a, Ang5a, Ang6a];
            All_Angles(j+1, 4:6) = [Ang4b, Ang5b, Ang6b];

        end // for j = 1:1
        // Rotate_XYZ(0, 0, Ang1)*Rotate_XYZ(0, -90, Ang2)*...
        // Rotate_XYZ(0, 0, Ang3)*Rotate_XYZ(0, 90, Ang4)*...
        // Rotate_XYZ(0, -90, Ang5)*Rotate_XYZ(90, 0, Ang6)


    end // checking if (Ang2a == [])&(Ang2b == [])&(Ang3a == [])&(Ang3b == [])

end // checking if (Dist1 > 2*L1_len)


////////////////////////////////////////////////////////////////////////////
//Setting the angles in the kinematic model to the ones we've just calculated
if ~exists('Sol_Nr') then
    global Sol_Nr;
    Sol_Nr = 1;
end //...if ~exists('Sol_Nr') then

if ~exists('Ani_idx') then // if it doesn't exist create it
    global Ani_idx;
end //...if ~exists('Ani_idx') then

for k = 1 : 6
    // if it exists do this to select the correct angle set!!!!
    if exists('Animation_Angs') then
        Sol_Nr = Min_Wrist_Angs();
    end // exists('Animation_Angs')
    Kin_Model.Joints(k).Joint_Angle.angZd = All_Angles(Sol_Nr, k);
end //...for k = 1 : 6

ik = 1; //Make sure we're in inverse kinematics mode!
//just update graphics, since we've changed the angles for display!



if exists('Dont_show') then
    // Don't update grpahics!!!!
else //...if exists('Dont_show') then
    Update_Graphics_Model();
end //...if exists('Dont_show') then
```

```
//*********************************************************************
// Testing Kinematic Solutions and Displaying Results ******************
// 1- to display the 8 solutions to the inverse kinematics; 0-discards
Display_Check = 0;

if (Display_Check == 1) then

    global ikSol_RotMatAll;
    ikSol_RotMatAll = list(0);
    global ikSol_TransVecAll;
    ikSol_TransVecAll = list(0);


    End_J_Indx = 6;

    Format = '%4.3f  ';
    str_TP_Coord = sprintf(Format, Ttp(1)) + sprintf(Format, Ttp(2)) + ...
                   sprintf(Format, Ttp(3));
    MyCheckString = "Original Tool Point Coordinates: " + str_TP_Coord;
    disp(MyCheckString);


    MyCheckString = "Original Input Angles: ";
    str_CurrentCoord = "";
    for i = 1 : End_J_Indx
        MyCheckString = MyCheckString + ...
                 sprintf(Format, Kin_Model.Joints(i).Joint_Angle.angZd);
    end

    disp(MyCheckString);
    i=3;
    str_CurrentCoord = sprintf(Format, TransVecAll(i)(1)) + ...
                       sprintf(Format, TransVecAll(i)(2)) + ...
                       sprintf(Format, TransVecAll(i)(3));
    disp('Coordinates of Wrist Origin: ' + str_CurrentCoord);

    disp ('Wrist Orientation Matrix...');
    disp(RotMatAll(6));


    for j = 1 : 8

        str_All_Angles = "";
        for i = 1 : End_J_Indx
            angX = Kin_Model.Joints(i).Joint_Angle.angXd;
            angY = Kin_Model.Joints(i).Joint_Angle.angYd;
            angZ = All_Angles(j, i);
            Link_Length = Kin_Model.Joints(i).Link_Length;
            X_vec = Link_Length*[1; 0; 0];

            if (i == 1)
                ikSol_RotMatAll(i) = Rotate_XYZ(angX, angY, angZ);
                ikSol_TransVecAll(i) = ikSol_RotMatAll(i)*X_vec;
                ikSol_Trans_Vec = [0 0; 0 0; 0 0]';
            else // from joints 2 to 6
                ikSol_RotMatAll(i) = ...
                    ikSol_RotMatAll(i-1)*Rotate_XYZ(angX, angY, angZ);
                ikSol_TransVecAll(i) = ikSol_TransVecAll(i-1) + ...
                    ikSol_RotMatAll(i)*X_vec;
                ikSol_Trans_Vec = [ikSol_TransVecAll(i-1), ...
                    ikSol_TransVecAll(i-1)]';
            end

            str_All_Angles = str_All_Angles + sprintf('%3.3f ', ...
                                                  All_Angles(j, i));
        end

        // The end position of all solutions should be the same!
        str_Wrist_Position = sprintf(Format, ikSol_Trans_Vec(1,1)) + ...
                             sprintf(Format, ikSol_Trans_Vec(1,2)) + ...
                             sprintf(Format, ikSol_Trans_Vec(1,3));
        MyCheckString = "Angle Set " + string(j) + ": " + str_All_Angles + ...
                        "  Wrist Position: " + str_Wrist_Position;
        disp(MyCheckString);

        disp('Wrist Orientation Matrix for this Solution:')
        disp(ikSol_RotMatAll(6))
```

```
    end //End for j = 1 : 8
    // ****************************************************************
end // Checking if Display_Check == 1
drawlater();
```

3.10 **File name:** CrossMult.sci

Purpose: Performs the cross multiplication of 2 vectors.

```
// Perfoms cross multiplication of 2 vectors.
// disp("-Debug line 173: CrossMult")
function [CVec, Result, Msg] = CrossMult(A, B)
    This_Fun_Name = "CrossMult:";
    Err_Msg = "";
    Size_A = size(A);
    Size_B = size(B);

    if ((Size_A(1)== 1) | (Size_A(2)== 1)) & ((Size_B(1) == 1) | (Size_B(2) == 1)) then
        // Both are vectors we can proceed!

        // A = Ax.i + Ay.j + Az.k
        // B = Bx.i + By.j + Bz.k

        AxBi = A(2)*B(3) - A(3)*B(2);
        AxBj = A(3)*B(1) - A(1)*B(3);
        AxBk = A(1)*B(2) - A(2)*B(1);

        Result = 1;
        CVec = [AxBi, AxBj, AxBk];

    elseif (Size_A(1)~= 1) | (Size_A(2)~= 1) then
        Err_Msg = "The first entry is not a 1 dimensional vector!";
        disp(Err_Msg);
        Result = 0;
        CVec = [];

    elseif (Size_B(1)~= 1) | (Size_B(2)~= 1) then
        Err_Msg = "The second entry is not a 1 dimensional vector!";
        disp(Err_Msg);
        Result = 0;
        CVec = [];

    end

    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.11 **File name:** Path_Follow_Select.sce

Purpose: Allows the user to select a path (square, circle, circle with changing orientation where the end effector points to a specific point, and a text output where the end effector traces out 'hello world') and watch as the inverse kinematics solves for the joint angles as the end effector is moved with the required orientation along the selected path. The forward kinematics script is called to animate the motion. The path is selected by setting the variable 'This_Path_Select' to a number between 1 and 5 inclusive.

```
global delete_and_clear;
delete_and_clear = 0;
```

```
if delete_and_clear then
    clc;
    xdel(winsid()); //Destroy all existing figures, this will be
    //reinitialized in the forward kinematics routine!
    clearglobal();
    clear;

    global delete_and_clear;
    delete_and_clear = 0;
end

// ALL PATH FOLLOWING
global Show_All_Paths;
global Current_Path;
global Const_Tool_Orient;
global Save_Animate_Angs;
global Step_Res;

// Adjusts resolution on step size of path folowing;
// 1 - accurate; 10 - fast
Step_Res = 10;
Save_Animate_Angs = 0; // 1 - save, 0 - discard

Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
PF_Script_Path = Kin_Scr_Path + "Path_Following.sce";
This_Path_Select = 5;

if (This_Path_Select == 0) then
    DispThisState = "on";
    DispState = "off"
    if exists('SqPath') then
       if (SqPath.Graphic_handles.visible ~= []) then
           if strcmp(SqPath.Graphic_handles.visible, DispThisState) == 0 then
               SqPath.Graphic_handles.visible = DispState;
           end // if strcmp(SqPath.Graphic_handles.visible, "on") == 0
       end // if (SqPath.Graphic_handles.visible ~= [])
    end // if exists('SqPath')

    if exists('CircPath') then
        if (CircPath.Graphic_handles.visible ~= []) then
            if strcmp(CircPath.Graphic_handles.visible, DispThisState) == 0 then
                CircPath.Graphic_handles.visible = DispState;
            end  //if strcmp(CircPath.Graphic_handles.visible, "on") == 0
        end // if (CircPath.Graphic_handles.visible ~= [])
    end // if ('CircPath')

    if exists('TxtPath') then
        if (TxtPath.Graphic_handles.visible ~= []) then
            if strcmp(TxtPath.Graphic_handles(1).visible, DispThisState) == 0 then
                for i = 1 : 10
                    TxtPath.Graphic_handles(i).visible = DispState;
                end  // for i = 1 : 10
                drawnow();
                drawlater();
            end  // strcmp(TxtPath.Graphic_handles(1).visible, "on") == 0
        end // if (TxtPath.Graphic_handles.visible ~= []
    end // if ('TxtPath')


elseif (This_Path_Select == 1) then
    //Square Path Constant Orientation
    Current_Path = "Square";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT

elseif (This_Path_Select == 2) then
    //Circle Path Constant Orientation
    Current_Path = "Circle";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT

elseif (This_Path_Select == 3) then
    //Circle Path Changing Orientation
    Current_Path = "Circle";
    Const_Tool_Orient = 0;
    exec(PF_Script_Path, -1); // UNCOMMENT
```

```
elseif (This_Path_Select == 4) then
    //Text Path Constant Orientation
    Current_Path = "Text";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT

elseif (This_Path_Select == 5) then
    // Execute all paths
    Current_Path = "Square";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT
    Current_Path = "Circle";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT
    Current_Path = "Circle";
    Const_Tool_Orient = 0;
    exec(PF_Script_Path, -1); // UNCOMMENT
    Current_Path = "Text";
    Const_Tool_Orient = 1;
    exec(PF_Script_Path, -1); // UNCOMMENT

end
```

3.12 **File name:** Path_Following.sce

Purpose: Shows and hides the selected path, and runs all the code for following a path as mentioned in 'Path_Follow_Select.sce'.

```
// PATH FOLLOWING - Circular, square, and text writing paths...
if ~exists('delete_and_clear') then
    global delete_and_clear;
    delete_and_clear = 1;
end

if delete_and_clear then
    clc;
    xdel(winsid()); //Destroy all existing figures, this will be
    // reinitialized in the forward kinematics routine!
    clearglobal();
    clear;

    global delete_and_clear;
    delete_and_clear = 0;
end



if ~exists('LinkageTester') then // Execute Inverse_Kinematics
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";
    exec(IK_Script_Path, -1);
    disp("Executing Path Following!");
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";

else
    disp("Figure Exists!");
    //UPDATES THE REFERENCE ANGLE GRAPHIC
    Script_Path = Kin_Scr_Path + "Angle_Reference_Display.sce";
    exec(Script_Path, -1);

end


//////////////////////////////////////////////////////////////////////
drawlater();
show_window([LinkageTester]) // Raises a figure handle

Line_Thickness = 2;

// *********************************************************************
```

```
global L1a_len;
global L1b_len;
global L3_len;


Vec_Length = 0.3*L1a_len;
// Setting the tool frame Orientation
Tool_Frame_Orient = Rotate_XYZ(-90, 0, 0);
// want a frame whose x-axis is perpendicular to the XZ plane;
Tool_Frame = [0 0 1; 0 1 0; -1 0 0]';
Tool_Frame = Tool_Frame_Orient*Tool_Frame;

Origin = [0; 0; 0];
Vec_End = Vec_Length*Tool_Frame + [Origin, Origin, Origin];

if exists('Path_Triad') then // Modify triad position or create it....
    //disp("Path_Triad exists!");
    for j = 1 : 3
        // Now create the Triad vectors
        x = [Origin(1), Vec_End(1, j)];
        y = [Origin(2), Vec_End(2, j)];
        z = [Origin(3), Vec_End(3, j)];
        Path_Triad(j+1).data = [x; y; z]';
    end

else
    disp("Path_Triad being created!");
    //Create A Visualisation Triad
    Path_Triad = tlist(["Path_Triad", "X_vec", "Y_vec", "Z_vec"], [], [], []);
    Line_Thickness = 2;
    Arrow_Size_Factor = 0.375;

    for j = 1 : 3
        // Now create the Triad vectors
        x = [Origin(1), Vec_End(1, j)];
        y = [Origin(2), Vec_End(2, j)];
        z = [Origin(3), Vec_End(3, j)];
        param3d(x, y, z);
        Polyline_Hndl = get("hdl");
        Polyline_Hndl.polyline_style = 4;
        // 1-Solid; 2-Dash; 3-Dash dot style; 4-ends in Arrow
        Polyline_Hndl.line_style = 2;
        Polyline_Hndl.thickness = Line_Thickness;
        Polyline_Hndl.arrow_size_factor = Arrow_Size_Factor;
        Polyline_Hndl.foreground = Dark_Colours(j);
        Path_Triad(j+1) = Polyline_Hndl;
    end

end //checks if exists('Path_Triad')
// *********************************************************************

// SET THE CURRENT PATH.......
if ~exists('Current_Path') then
    global Current_Path;
    Current_Path = "Text";
    Current_Path = "Circle";
    Current_Path = "Square";
end // if ~exists('Current_Path')



// This creates the different types of paths...
if ~exists('SqPath') then // If the paths do not exist then create them
    global SqPath;
    global CircPath;
    global TxtPath;

    drawlater;
    Script_Path = Kin_Scr_Path + "Path_Creation.sce";
    exec(Script_Path, -1);

elseif (sum(size(SqPath)) == 0) then
    drawlater;
    Script_Path = Kin_Scr_Path + "Path_Creation.sce";
    exec(Script_Path, -1);
```

```
end // Checking if ~exists('SqPath') -


// Adjusts visibility of paths based on selection
if (strcmp(Current_Path, "Square") == 0) then
    // Show path
    SqPath.Graphic_handles.visible = "on";
    CircPath.Graphic_handles.visible = "off";
    for i = 1:10
        TxtPath.Graphic_handles(i).visible = "off";
    end
    CP_Data = SqPath.Data;
    Total_Play_Time = 10;


elseif (strcmp(Current_Path, "Circle") == 0)
    // Show path
    CircPath.Graphic_handles.visible = "on";
    SqPath.Graphic_handles.visible = "off";
    for i = 1:10
        TxtPath.Graphic_handles(i).visible = "off";
    end
    CP_Data = CircPath.Data;
    Total_Play_Time = 10;


elseif (strcmp(Current_Path, "Text") == 0)
    // Show path
    // TxtPath = tlist(["Path", "Characters", "Positions", ...
    //                  "Graphic_handles"], list(0), [], list(0));
    CP_Data = [];
    Number_of_Letters = size(TxtPath.Characters);
    //Number_of_Letters = 4;
    for i = 1:Number_of_Letters
        TxtPath.Graphic_handles(i).visible = "on";
        CP_Data = [CP_Data, TxtPath.Characters(i).Data];
    end

    CircPath.Graphic_handles.visible = "off";
    SqPath.Graphic_handles.visible = "off";
    Time_per_Letter = 6;
    Total_Play_Time = Time_per_Letter*Number_of_Letters;


else
    disp("Selected path does not exist!")
    disp(Current_Path)
end


//Improve resolution on the path
if ~exists('Step_Res') then
    global Step_Res;
    Step_Res = 10;
end // if ~exists('Step_Res')


// Only do this for linear paths
if ~strcmp(Current_Path, "Circle") == 0 then
    // 5mm - Changed this to make it faster.....
    Min_Step_Len = Step_Res*0.005*L1a_len;
    // improves the resolution on the PathFollowing
    CP_Data = Improve_Path_Res(CP_Data, Min_Step_Len);
end


tmpBnd = (L1a_len + L1b_len + 1.5*L3_len);
//[xmin ymin; xmax ymax]->must be 2 dimensional
LinkageTester.children($).data_bounds = ...
                [-tmpBnd, -tmpBnd, -4; tmpBnd, tmpBnd, tmpBnd+5];
global Current_View_Point;
LinkageTester.children($).rotation_angles = Current_View_Point;
LinkageTester.children($).box = "back_half";
drawnow;
```

```
// *******************************************************
// SOLVING INVERSE KINEMATICS
// Now we're going to move the Path_Triad along the path....
global TransVecAll;
global RotMatAll;
global All_Angles;
global Sol_Nr;

IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";
Path_Size = size(CP_Data);
Animation_Angs = [];
Ani_idx = 1;


if ~exists('Const_Tool_Orient') then // if it doesn't exist create it
    global Const_Tool_Orient;
    //constant orientation - 1, Changing orientation - 0
    Const_Tool_Orient = 1;
end

if (Const_Tool_Orient == 0) then //Changing orientation along tool path
    // follow circle path with changing orientation
    if (strcmp(Current_Path, "Circle") == 0) then
        global Circ_Tool_Frames
        for p = 1 : Path_Size(2)-1  // -- Const_Tool_Orient = 0

            Origin = CP_Data(:, p);

            Vec_End = Vec_Length*Circ_Tool_Frames(p) + ...
                           [Origin, Origin, Origin];

            TransVecAll(7) = Origin;
            RotMatAll(7) = Circ_Tool_Frames(p);
            exec(IK_Script_Path, -1); // UNCOMMENT
            // At this point we have all the angles
            // We can now save these angles with time stamps
            // for the Solid Works 3D Animation...
            Animation_Angs(Ani_idx, :) = All_Angles(Sol_Nr, :);
            Ani_idx = Ani_idx + 1;

            for k = 1 : 3
                // Now adjust the path following Triad vectors position
                x = [Origin(1), Vec_End(1, k)];
                y = [Origin(2), Vec_End(2, k)];
                z = [Origin(3), Vec_End(3, k)];
                Path_Triad(k+1).data = [x; y; z]';
            end
            drawnow();

        end // for p = 1 : Path_Size(2)-1 -- Const_Tool_Orient = 0
    //else
        //Const_Tool_Orient = 1;  //So that it can enter the next loop
        //for constatn orientation path following
    end //if (strcmp(Current_Path, "Circle") == 0)
    // follow circle path with changing orientation
end // if (Const_Tool_Orient == 0)



if (Const_Tool_Orient == 1) then
    for p = 1 : Path_Size(2)-1 // --Const_Tool_Orient = 1

        Origin = CP_Data(:, p);

        Vec_End = Vec_Length*Tool_Frame + [Origin, Origin, Origin];

        TransVecAll(7) = Origin;
        RotMatAll(7) = Tool_Frame;
        exec(IK_Script_Path, -1); // UNCOMMENT
        // At this point we have all the angles
        // We can now save these angles with time stamps for the
        // Solid Works 3D Animation...
        Animation_Angs(Ani_idx, :) = All_Angles(Sol_Nr, :);
        Ani_idx = Ani_idx + 1;
```

```
        for k = 1 : 3
            // Now adjust the path following Triad vectors position
            x = [Origin(1), Vec_End(1, k)];
            y = [Origin(2), Vec_End(2, k)];
            z = [Origin(3), Vec_End(3, k)];
            Path_Triad(k+1).data = [x; y; z]';
        end
        drawnow();

    end // for p = 1 : Path_Size(2)-1  -- Const_Tool_Orient = 1
end //if (Const_Tool_Orient == 1)



Update_Graphics_Model(); //just update graphics one last time
                         // before exiting
drawnow();


// SAVE ANGLES FOR 3D ANIMATION
// Save Animation Angles
if ~exists('Save_Animate_Angs') then
    global Save_Animate_Angs;
    Save_Animate_Angs = 0;
end

if (Save_Animate_Angs == 1) then

    // Just make sure each start angle is 0!
    Animation_Angs = [0, 0, 0, 0, 0, 0; Animation_Angs];

    SZ_Animation_Angs = size(Animation_Angs);
    //Total_Play_Time = 5; //Remove
    Time_inc = Total_Play_Time/(SZ_Animation_Angs(1)-1);

    TimeVec = [];
    for i = 1:SZ_Animation_Angs(1)
        TimeVec(i) = (i-1)*Time_inc;
    end // for i = 1:SZ_Animation_Angs(1)
    Animation_Angs = [Animation_Angs, TimeVec];

    Location = Kin_Scr_Path + "IK Path Data\";
    Location = Location + Current_Path + '\';
    // Const_Tool_Orient = 1, Appending file name
    if Const_Tool_Orient == 1 then
        Location = Location + 'Constant_Orientation' + '\';
    else // Const_Tool_Orient = 0 --
        Location = Location + 'Changing_Orientation' + '\';
    end // if Const_Tool_Orient == 1 then -- Appending file name

    for i = 1:6
        File_Name = 'Ang' + string(i) + '.txt';
        File_Name_and_Location = Location + File_Name;
        fd = mopen(File_Name_and_Location, 'wt');
        // Creates a string vector out of a matrix, where each element
        // in the vector, is a tab separated string of the column of the
        // matrix.
        txt = strcat(string([Animation_Angs(:, 7), ...
                             Animation_Angs(:, i)]'), "\t", "r");
        // Creates a single string from the vector string, with a new
        // line character separating the elements of the vetor.
        txt = strcat(txt, "\n");
        mfprintf(fd, txt);
        mclose(fd);
    end // for i = 1:2
end // Checking if (Save_Animate_Angs == 1)
```

3.13 **File name:** Path_Creation.sce

Purpose: This code creates all the paths mentioned previously (circle, square, and text).

```
drawlater();
```

```
if exists('LinkageTester') then
    show_window([LinkageTester]) // Raises a figure handle
end

//global Robot_Paths;
global SqPath;
global CircPath;
global TxtPath;

SqPath = tlist(["Path", "Data", "Graphic_handles"], [], []);
CircPath = tlist(["Path", "Data", "Graphic_handles"], [], []);
TxtPath = tlist(["Path", "Characters", "Positions", "Graphic_handles"], list(0), [], list(0));

global L1a_len;
global L1b_len;
global L3_len;

All_Centres = [0; L1b_len; L1b_len];

Line_Thickness = 2;
// CREATING PATHS ........
// Creating Square path, its graphics and data
//only enter if the graphics handle object is empty!
if (SqPath.Graphic_handles == [])
    disp('Creating square path graphics.')
    Sq_Len = 0.3*L1b_len;
    SqCentre_Pos = All_Centres;
    SqCentre_Orient = Rotate_XYZ(-90, 0, 0); //Rotate_XYZ(90, 0, 0);

    // Scales the size of the square path
    SqData = (Sq_Len/2)*[1 1 0; -1 1 0; -1 -1 0; 1 -1 0; 1 1 0]';

    // orients the square and then translates its centre to
    // the point we want...
    SqData = SqCentre_Orient*SqData + [SqCentre_Pos, SqCentre_Pos, ...
                            SqCentre_Pos, SqCentre_Pos, SqCentre_Pos];
    SqPath.Data = SqData;

    // Determine size of data, for the number of lines that need to
    // be drawn....
    SizeDataPath = size(SqData);
    // The columns equals the number of lines that need to be drawn.

    // Draw Square Path Now create the Triad vectors
    x = SqData(1, :);
    y = SqData(2, :);
    z = SqData(3, :);

    param3d(x, y, z);
    Polyline_Hndl = get("hdl");
    Polyline_Hndl.polyline_style = 1;
    // 1-Solid; 3-Dash dot style; 4-ends in Arrow
    Polyline_Hndl.line_style = 1;
    Polyline_Hndl.thickness = Line_Thickness;
    Polyline_Hndl.foreground = 38;
    Polyline_Hndl.visible = "off";
    SqPath.Graphic_handles = Polyline_Hndl;
end


// Creating Circular Path, its graphics and data
// only enter if the graphics handle object is empty!
if (CircPath.Graphic_handles == [])
    disp('Creating circle path graphics.')
    Radius = 0.3*L1b_len/2;
    Centre_Pos = All_Centres;
    Centre_Orient = Rotate_XYZ(-90, 0, 0); //Rotate_XYZ(90, 0, 0);

    if exists('Step_Res') then
        // 8mm - Changed this to make it faster.....
        Min_Step_Len = Step_Res*0.008*L1a_len;
        d_polyg_radn = Min_Step_Len/Radius;
        d_polyg_ang = d_polyg_radn*180/(%pi);
    else
        Num_Edges = 100;
        d_polyg_ang = 360/Num_Edges;
    end
```

```
    off_set_ang = 0;
    idx = 1;
    for polyg_ang = (0+off_set_ang) : d_polyg_ang : (360+off_set_ang)
        // drawing circle on xy-plane then orients and shifts it
        // //Orient the data and position the circle properly
        polyg_data(:,idx) = Centre_Orient*[Radius*cosd(polyg_ang);...
                            Radius*sind(polyg_ang); 0] + Centre_Pos;
        idx = idx+1;
    end

    // Draw the graphic entity
    x = polyg_data(1, :);
    y = polyg_data(2, :);
    z = polyg_data(3, :);

    param3d(x, y, z);
    Polyline_Hndl = get("hdl");
    Polyline_Hndl.polyline_style = 1;
    // 1-Solid; 3-Dash dot style; 4-ends in Arrow
    Polyline_Hndl.line_style = 1;
    Polyline_Hndl.thickness = Line_Thickness;
    Polyline_Hndl.foreground = 38;
    Polyline_Hndl.visible = "off";
    CircPath.Graphic_handles = Polyline_Hndl;
    CircPath.Data = polyg_data;


    // ***************************************************************
    //Now we want to create an orientation matrix dataset for the
    // circle following path
    // Going to create frames that point to the centre only!
    global Circ_Tool_Frames;
    Circ_Tool_Frames = list(0);
    Sz_polyg_data = size(polyg_data);
    // Setting the tool frame Orientation
    // want a frame whose x-axis is perpendicular to the YZ plane;
    CTool_Frame = [-1 0 0; 0 0 -1; 0 -1 0]';

    idx = 1;
    for polyg_ang = (0+off_set_ang) : d_polyg_ang : (360+off_set_ang)
        // Creating a set of orientations for the circle trajectory
        //CONE
        Circ_Tool_Frames(idx) = Rotate_XYZ(0, polyg_ang, -60)*CTool_Frame;
        idx = idx + 1;
    end
    // HAVE TO KILL THE FIRST LIST ELEMENT


end



// Creating Text Path, its graphics and data
// TxtPath = tlist(["Path", "Characters", "Positions", ...
//                "Graphic_handles"], list(0), [], list(0));
//only enter if the graphics handle object is empty!
if (TxtPath.Graphic_handles == list(0))
    disp('Creating text path graphics.')
    Bx_Width = 0.08*L1a_len;
    Bx_Height = 2.25*Bx_Width;
    Letter_Centre = [0; 0; 0];

    // H ***************************
    Txt_H = tlist(["H", "Data", "Centre"], [], []);
    Letter_Data = [-Bx_Width/2, Bx_Height/2, 0; ...
                   -Bx_Width/2, -Bx_Height/2, 0; ...
                   -Bx_Width/2, 0, 0; ...
                    Bx_Width/2, 0, 0; ...
                    Bx_Width/2, Bx_Height/2, 0; ...
                    Bx_Width/2, -Bx_Height/2, 0;]';

    Txt_H.Data = Letter_Data;


    // E ***************************
    Txt_E = tlist(["E", "Data", "Centre"], [], []);
```

```
Letter_Data = [-Bx_Width/2, -Bx_Height/2, 0; ...
               -Bx_Width/2, Bx_Height/2, 0; ...
                Bx_Width/2, Bx_Height/2, 0; ...
               -Bx_Width/2, Bx_Height/2, 0; ...
               -Bx_Width/2, 0, 0; ...
                0, 0, 0; ...
               -Bx_Width/2, 0, 0;...
               -Bx_Width/2, -Bx_Height/2, 0;...
                Bx_Width/2, -Bx_Height/2, 0]';

Txt_E.Data = Letter_Data;


// L ***************************
Txt_L = tlist(["L", "Data", "Centre"], [], []);
Letter_Data = [-Bx_Width/2, -Bx_Height/2, 0; ...
               -Bx_Width/2, Bx_Height/2, 0; ...
               -Bx_Width/2, -Bx_Height/2, 0; ...
                Bx_Width/2, -Bx_Height/2, 0;]';
Txt_L.Data = Letter_Data;


// O ****************************
Txt_O = tlist(["O", "Data", "Centre"], [], []);
clear Letter_Data;

Num_Circ_Points = 10;
D_circ_ang = 90/Num_Circ_Points;
Circ_Radius = Bx_Width/2;

// Lower Right corner first
Circ_Centre = [Bx_Width/2-Circ_Radius; -Bx_Height/2+Circ_Radius; 0];
idx = 1;
for Circ_ang = 270 : D_circ_ang : 360
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                  Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end

// Upper Right corner
Circ_Centre = [Bx_Width/2-Circ_Radius; Bx_Height/2-Circ_Radius; 0];
for Circ_ang = 0 : D_circ_ang : 90
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                  Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end

// Upper Left corner
Circ_Centre = [-Bx_Width/2+Circ_Radius; Bx_Height/2-Circ_Radius; 0];
for Circ_ang = 90 : D_circ_ang : 180
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                  Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end

// Lower Left corner
Circ_Centre = [-Bx_Width/2+Circ_Radius; -Bx_Height/2+Circ_Radius; 0];
for Circ_ang = 180 : D_circ_ang : 270
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                  Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end
Txt_O.Data = Letter_Data;



// W ***************************
Txt_W = tlist(["W", "Data", "Centre"], [], []);
Letter_Data = [-Bx_Width/2, Bx_Height/2, 0; ...
               -Bx_Width/4, -Bx_Height/2, 0; ...
                0, 0, 0; ...
                Bx_Width/4, -Bx_Height/2, 0; ...
                Bx_Width/2, Bx_Height/2, 0; ...
                Bx_Width/4, -Bx_Height/2, 0]';

Txt_W.Data = Letter_Data;
```

```
// R ******************************
Txt_R = tlist(["R", "Data", "Centre"], [], []);
Letter_Data = [-Bx_Width/2, -Bx_Height/2, 0; ...
               -Bx_Width/2, Bx_Height/2, 0]';

Num_Circ_Points = 20;
D_circ_ang = 90/20;
Circ_Radius = 0.5*Bx_Width;

// From +90 to -90 clockwise
Circ_Centre = [Bx_Width/2-Circ_Radius; Bx_Height/2-Circ_Radius; 0];
idx = 3;
for Circ_ang = 90 : -D_circ_ang : -90
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                          Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end
Letter_Data(:,idx) = [Bx_Width/2, -Bx_Height/2, 0]';
Txt_R.Data = Letter_Data;



// D ******************************
Txt_D = tlist(["D", "Data", "Centre"], [], []);
Letter_Data = [-Bx_Width/2, -Bx_Height/2, 0; ...
               -Bx_Width/2, Bx_Height/2, 0]';

Num_Circ_Points = 20;
D_circ_ang = 90/20;
Circ_Radius = 0.65*Bx_Width;

// From +90 to 0 clockwise
Circ_Centre = [Bx_Width/2-Circ_Radius; Bx_Height/2-Circ_Radius; 0];
idx = 3;
for Circ_ang = 90 : -D_circ_ang : 0
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang);...
                         Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end

// From 0 to -90 clockwise
Circ_Centre = [Bx_Width/2-Circ_Radius; -Bx_Height/2+Circ_Radius; 0];
for Circ_ang = 0 : -D_circ_ang : -90
    Letter_Data(:,idx) = [Circ_Radius*cosd(Circ_ang); ...
                          Circ_Radius*sind(Circ_ang); 0] + Circ_Centre;
    idx = idx+1;
end
Letter_Data(:,idx) = [-Bx_Width/2, -Bx_Height/2, 0]';
Txt_D.Data = Letter_Data;



// ****************************************************************
// TEXT POSITIONING ON XY PLANE
// HELLO ; WORLD - Both 5 letters each the centre point of the first
// L from hello will be on the mid line, same for the R of world.
Letter_Hori_Space = 1.35*Bx_Width;
Letter_Vert_Space = 1.55*Bx_Height;

k = 1;
for j = 0:1
    for i = 0:4
        Character_Positions(:, k) = ...
                    [i*Letter_Hori_Space, -j*Letter_Vert_Space, 0]';
        k = k+1;
    end
end

TxtPath.Characters(1) = Txt_H;
TxtPath.Characters($+1) = Txt_E;
TxtPath.Characters($+1) = Txt_L;
TxtPath.Characters($+1) = Txt_L;
TxtPath.Characters($+1) = Txt_O;
TxtPath.Characters($+1) = Txt_W;
TxtPath.Characters($+1) = Txt_O;
```

```
    TxtPath.Characters($+1) = Txt_R;
    TxtPath.Characters($+1) = Txt_L;
    TxtPath.Characters($+1) = Txt_D;

    TxtPath.Positions = Character_Positions;
    Shift = [-2*Letter_Hori_Space, Letter_Vert_Space/2, 0]';
    Translate = All_Centres;//[0, L1b_len, L1b_len]';
    //Rotate_XYZ(-90, 0, 0); //Rotate_XYZ(90, 0, 0);
    TxtPath_Orient = Rotate_XYZ(-90, 0, 180);

    for i = 1:10
        Letter_Data = TxtPath.Characters(i).Data;
        size_Letter_Data = size(Letter_Data);
        for j = 1:size_Letter_Data(2)
            Letter_Data(:, j) = TxtPath_Orient*(Letter_Data(:, j) + ...
                                TxtPath.Positions(:, i) + Shift) + Translate;
        end
        TxtPath.Characters(i).Data = Letter_Data;

        // DRAW THE LETTER ENTITIES
        x = Letter_Data(1, :);
        y = Letter_Data(2, :);
        z = Letter_Data(3, :);

        param3d(x, y, z);
        Polyline_Hndl = get("hdl");
        Polyline_Hndl.polyline_style = 1;
        // 1-Solid; 3-Dash dot style; 4-ends in Arrow
        Polyline_Hndl.line_style = 1;
        Polyline_Hndl.thickness = 2;
        Polyline_Hndl.foreground = color('purple');
        Polyline_Hndl.visible = "off";
        TxtPath.Graphic_handles(i) = Polyline_Hndl;
    end

    //for i = 1:10
        //delete(LinkageTester.children($).children(1))
    //end
end

// Save Paths
Save_Paths = 0;
if (Save_Paths == 1) then
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    Location =  Kin_Scr_Path + "IK Path Data\";
    File_Name = 'Square Path.sldcrv';
    File_Name_and_Location = Location + File_Name;
    fd = mopen(File_Name_and_Location, 'wt');
    // Creates a string vector out of a matrix, where each element in
    // the vector, is a tab separated string of the column of the matrix.
    txt = strcat(string(SqPath.Data), "\t", "r");
    // Creates a single string from the vector string, with a new line
    // character separating the elements of the vetor.
    txt = strcat(txt, "\n");
    mfprintf(fd, txt);
    mclose(fd);
    SqPath_Data = SqPath.Data;
    save(File_Name_and_Location+'.dat', SqPath_Data);


    File_Name = 'Circle Path.sldcrv';
    File_Name_and_Location = Location + File_Name;
    fd = mopen(File_Name_and_Location, 'wt');
    // Creates a string vector out of a matrix, where each element in
    // the vector, is a tab separated string of the column of the matrix.
    txt = strcat(string(CircPath.Data), "\t", "r");
    // Creates a single string from the vector string, with a new line
    // character separating the elements of the vetor.
    txt = strcat(txt, "\n");
    mfprintf(fd, txt);
    mclose(fd);
    CircPath_Data = CircPath.Data;
    save(File_Name_and_Location+'.dat', CircPath_Data);


    File_Name = 'Text Path.sldcrv';
    File_Name_and_Location = Location + File_Name;
```

```
    fd = mopen(File_Name_and_Location, 'wt');

    TxtPath_Data = [];
    for i = 1:10
        TxtPath_Data = [TxtPath_Data, TxtPath.Characters(i).Data];
    end
    // Creates a string vector out of a matrix, where each element in
    // the vector, is a tab separated string of the column of the matrix.
    txt = strcat(string(TxtPath_Data), "\t", "r");
    // Creates a single string from the vector string, with a new line
    // character separating the elements of the vetor.
    txt = strcat(txt, "\n");
    mfprintf(fd, txt);
    mclose(fd);
    save(File_Name_and_Location+'.dat', TxtPath_Data);
end
```

3.14 **File name:** Improve_Path_Res.sci

Purpose: Improves the path resolution for straight line paths. Straight lines can be set by just the end coordinates of the line. This code then created data points between those end points according to the resolution desired.

```
// IMPROVING PATH FOLLOWING RESOLUTION
// disp("-Debug line 173: Improve_Path_Res")
function [new_PathData, Result, Msg] = Improve_Path_Res(PathData, Min_Step_Dist)
    This_Fun_Name = "Improve_Path_Res:";
    Err_Msg = "";

    new_PathData = [];
    SZ_PathData = size(PathData)
    for i = 2:SZ_PathData(2)
        dVec = PathData(:, i) - PathData(:, i-1);
        dl = ceil( ((dVec'*dVec)^0.5)/Min_Step_Dist );

        Added_Data = []; //Reset
        // we will be increasing the number of points between
        // SqPath_Data points...
        if (dl > 1) then
            //Check if dl is odd
            if pmodulo(dl,2) then
                // means dl is odd, so make it even, this will then force
                // the length of added data to be odd...
                dl = dl+1;
            end

            Fraction = 1/dl;
            Added_Data(:, 1) = PathData(:, i-1) + Fraction*dVec;
            for j = 2:dl
                // The last value in added data must take you to within
                // a fraction of SqPath_Data(:, i)
                Added_Data(:, j) = Added_Data(:, j-1) + Fraction*dVec;
            end

        end // Checking if (dl > 1)
        new_PathData = [new_PathData, PathData(:, i-1), Added_Data];

    end
    // at this point i = SZ_PathData(2);
    // new_PathData = [new_PathData, PathData(:, i)];
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.15 **File name:** Min_Wrist_Angs.sci

Purpose: This code uses a weighting system to select the wrist angles that use a minimum

rotation when going from one configuration to the next.

```
// Min_Wrist_Angs - finds the minimum wrist rotation angles by
// using a weighting system.
// disp("-Debug line 173: Min_Wrist_Angs")
function [Solution_nr, Result, Msg] = Min_Wrist_Angs()
    global All_Angles;
    global Animation_Angs;
    global Ani_idx;

    This_Fun_Name = "Min_Wrist_Angs:";
    Err_Msg = "";

    if (Ani_idx > 1) then
        // We need to select the correct solution cause the wrist
        // rotation angles flip drastically (0 to 180)
        // We use a minimum criterion, sum of the changes in
        // wrist angles must be a minimum
        Pre_Ani_idx = Ani_idx - 1;
        ax = Animation_Angs(Pre_Ani_idx, 4);
        bx = Animation_Angs(Pre_Ani_idx, 5);
        cx = Animation_Angs(Pre_Ani_idx, 6);

    else //chooses the first solution
        ax = 0;
        bx = 0;
        cx = 0;
    end

    a1 = All_Angles(1, 4); b1 = All_Angles(1, 5); c1 = All_Angles(1, 6);
    a2 = All_Angles(2, 4); b2 = All_Angles(2, 5); c2 = All_Angles(2, 6);

    a11 = abs(a1-ax); a12 = abs(360-a11);
    if (a11 > a12) then
        a1 = a12;
    else
        a1 = a11;
    end

    b11 = abs(b1-bx); b12 = abs(360-b11);
    if (b11 > b12) then
        b1 = b12;
    else
        b1 = b11;
    end

    c11 = abs(c1-cx); c12 = abs(360-c11);
    if (c11 > c12) then
        c1 = c12;
    else
        c1 = c11;
    end

    a21 = abs(a2-ax); a22 = abs(360-a21);
    if (a21 > a22) then
        a2 = a22;
    else
        a2 = a21;
    end

    b21 = abs(b2-bx); b22 = abs(360-b21);
    if (b21 > b22) then
        b2 = b22;
    else
        b2 = b21;
    end

    c21 = abs(c2-cx); c22 = abs(360-c21);
    if (c21 > c22) then
        c2 = c22;
    else
        c2 = c21;
    end
```

```
    fa = 100;
    fb = 10000;
    min_ang_sum1 = fa*a1 + fb*b1 + c1 + fb*b1*a1;
    min_ang_sum2 = fa*a2 + fb*b2 + c2 + fb*b2*a2;
    if (min_ang_sum1 > min_ang_sum2) then
        Solution_nr = 2;
    else
        Solution_nr = 1;
    end

    Result = 1;
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

3.16 **File name:** Angle_Reference_Display.sce

Purpose: This allows the user to see the selected angle in the forward kinematics, as a grey disc between the x vectors that define it according to the Denavit-Hartenberg notation.

```
// Angle Reference display...
global LinkageTester;
global Kin_Model;
global Joint_Index;

global AngInc;
AngInc = 5;
global RotMatAll;
global TransVecAll;
global Display_Modes;
global ik;
global Current_View_Point;

global AngRef_CircData;
global AngRef_Circ;
global AngRef_Vec;


    if (AngRef_CircData == [])  then //- does not exist
        //Create the data
        global L1a_len;
        x = 0.65*0.3*L1a_len;
        j = 1;
        // for polyline first point must be the origin
        AngRef_CircData(j, :) = [0, 0, 0];
        j=j+1;

        for i = 0 : AngInc : 360
            AngRef_CircData(j, :)= x*[cosd(i), sind(i), 0];
            j=j+1;
        end // for i = 0 : AngInc : 360
    end // if ~exists('AngRef_CircData')



    // change visibility and orientation - must track the current
    // selected angle. Only do this if in forward kinemtaics mode
    if (ik == 0)&(Display_Modes == 5) then
        drawlater();
        Current_AngZ = Kin_Model.Joints(Joint_Index.Current).Joint_Angle.angZd;
        end_AngRef_idx = Current_AngZ/AngInc + 1 + 1;
        FPolyData = [AngRef_CircData(1:end_AngRef_idx, :)];
        // Now create the Refence X vector
        Vec_Length = 0.3*L1a_len;
        RefVecData = Vec_Length*[0, 0, 0; 1, 0, 0];

        if (AngRef_Circ == []) then // does not exist
            cx = FPolyData(:, 1);
            cy = FPolyData(:, 2);
            cz = FPolyData(:, 3);
            xfpoly(cx, cy, 1);
            AngRef_Circ = get("hdl");
```

```
        AngRef_Circ.thickness = 2; //Line_Thickness;
        AngRef_Circ.background = color("grey"); // filled colour
        AngRef_Circ.foreground = color("blue"); // line colour

        // Now create the Refence X vector
        Vec_Length = 0.3*L1a_len;
        x = Vec_Length*[0, 1];
        y = Vec_Length*[0, 0];
        z = Vec_Length*[0, 0];

        global Dark_Colours;
        Line_Thickness = 4;
        Arrow_Size_Factor = 0.375;

        param3d(x, y, z);
        AngRef_Vec = get("hdl");
        AngRef_Vec.polyline_style = 4; // 4-ends in Arrow
        // 1-Solid; 3-Dash dot style; 8-Double Dot
        AngRef_Vec.line_style = 8;
        AngRef_Vec.thickness = Line_Thickness;
        AngRef_Vec.arrow_size_factor = Arrow_Size_Factor;
        AngRef_Vec.mark_mode = "off";
        AngRef_Vec.foreground = Dark_Colours(1);

        LinkageTester.children($).rotation_angles = Current_View_Point;
        LinkageTester.children($).box = "off";

    end //if ~exists('AngRef_Circ')

    // Manipulate and assign data
    Rm1 = RotMatAll(Joint_Index.Current)*Rotate_ZYX(0, 0, -Current_AngZ);
    FPolyData = (Rm1*FPolyData')';
    Sz_FPolyData = size(FPolyData);

    for i = 1 : Sz_FPolyData(1, 1)
        FPolyData(i, :) = FPolyData(i, :) + ...
                  TransVecAll(Joint_Index.Current)';
    end //for i = 1 : Sz_FPolyData(1, 1)

    AngRef_Circ.visible = "on";
    AngRef_Circ.data = FPolyData;

    AngRef_Vec.visible = "on";
    //Rotate and translate reference vector to correct position
    AngRef_Vec.data = (Rm1*(RefVecData'))' + ...
                      [TransVecAll(Joint_Index.Current)'; ...
                       TransVecAll(Joint_Index.Current)'];


else //if (ik == 0)&(Display_Modes == 5)
    if exists('AngRef_Circ') then
        if (AngRef_Circ ~= []) then // make sure its defined!
            AngRef_Circ.visible = "off";
            AngRef_Vec.visible = "off";
        end // if (AngRef_Circ ~= []) -- // make sure its defined!
    end //if exists('AngRef_Circ')


end //if (ik == 0)&(Display_Modes == 5)
```

4 HKM Dynamics

4.1 **File name:** Dyn_Path_Select.sce

Purpose: Allows the user to select a vertical, horizontal or radial path (2 for each, centred on different points). The paths are created, the inverse kinematics is called, which in turn calls the forward kinematics, and the angles for each pose are solved and animated for the pre-configured orientation. Once the angles are obtained the time vector is created using the duration set with the variable 'Total_Play_Time'. The velocities and accelerations are then

calculated for each pose. The inverse dynamics script is then called which creates the dynamics models and then solves for the joint torques.

```
// Will allow the user to select a path, and a duration for motion, and
// then run the scripts for the dynamics.


if ~exists('delete_and_clear') then
    global delete_and_clear;
    delete_and_clear = 1;
end //...if ~exists('delete_and_clear') then


if delete_and_clear then
    clc;
    xdel(winsid()); //Destroy all existing figures
    clearglobal();
    clear;
    lines(0); //disables vertical paging
    disp('Deleting all variables and clearing workspace... (at Dyn_Path_Select)');

    global delete_and_clear;
    delete_and_clear = 0;
end


Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";

// SET THE CURRENT PATH.......
if ~exists('Current_Path') then
    global All_paths;
    All_paths = ["H_line1";...
                 "H_line2";...
                 "V_line1";...
                 "V_line2";...
                 "R_line1";...
                 "R_line2"];

    global Current_Path;
    Current_Path = All_paths(1);

    Saved_Path = Dyn_Scr_Path + "Dynamics Data\Current_Path.dat";
    save(Saved_Path, Current_Path);
end // if ~exists('Current_Path')


// ********************************************************************
// SELECT A PATH
Current_Path = All_paths(1);  //Changes the path


// ********************************************************************
// SET A DURATION FOR PATH FOLLOWING
if ~exists('Total_Play_Time') then
    global Total_Play_Time
end
Total_Play_Time = 5; // /20


// ********************************************************************
// SAVE ALL DATA
if ~exists('Save_Data') then
    global Save_Data;
end //...if ~exists('Save_Data') then
Save_Data = 0; // 0 - don't save; 1 - save


// ********************************************************************
// SELECTS HEAVY OR LIGHT MOTORS FOR THE SERIAL DYNAMIC MODEL
if ~exists('Heavy_motors') then
    global Heavy_motors;
end
Heavy_motors = 0; //1 -> heavy motors are used
```

```
// ********************************************************************
// FIND THE JOINT ANGLES FOR EACH POSE ALONG THE PATH
exec(Dyn_Scr_Path + 'Dyn_Path_Fol_Lin.sce');


// ********************************************************************
// CREATE THE TIME VECTOR AND FIND VELOCITIES AND ACCELERATIONS
exec(Dyn_Scr_Path + 'Calc_Vel_Acc.sce');


// ********************************************************************
// RUN THE DYNAMICS SIMULATION
exec(Dyn_Scr_Path + 'Inverse_Dynamics.sce');


// ********************************************************************
// CALCULATE THE ENERGY USED
exec(Dyn_Scr_Path + 'Calc_Energy_Used.sce');
```

4.2 **File name:** Dyn_Path_Fol_Lin.sce

Purpose: Creates paths and solves for joint angles along those paths. This script is called by 'Dyn_Path_Select.sce'.

```
// DYNAMICS PATH FOLLOWING - Vertical and horizontal lines...
// INVERSE KINEMATICS - To get angles
// FORWARD KINEMATICS - For figure initialization and graphics


if ~exists('LinkageTester') then // Execute Inverse_Kinematics
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\"
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";
    exec(IK_Script_Path, -1);
    disp("Executing Path Following!");

    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";

    Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";

else //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics

    //UPDATES THE REFERENCE ANGLE GRAPHIC
    Script_Path = Kin_Scr_Path + "Angle_Reference_Display.sce";
    exec(Script_Path, -1);

end //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics


//////////////////////////////////////////////////////////////////////
drawlater();
//show_window([LinkageTester]) // Raises a figure handle
Line_Thickness = 2;
// ********************************************************************


global L1a_len;
global L1b_len;
global L3_len;


Vec_Length = 0.3*L1a_len;
// Setting the tool frame Orientation
Tool_Frame_Orient = Rotate_XYZ(-90, 0, 0);
// want a frame whose x-axis is perpendicular to the XZ plane;
Tool_Frame = [0 0 1; 0 1 0; -1 0 0]';
Tool_Frame = Tool_Frame_Orient*Tool_Frame;
```

```
Origin = [0; 0; 0];
Vec_End = Vec_Length*Tool_Frame + [Origin, Origin, Origin];


if ~exists('Dont_show') then
    global Dont_show;
    Dont_show = 0; //0-> show graphics
end



if exists('Path_Triad') then // Modify triad position or create it....
    //disp("Path_Triad exists!");
    if (Dont_show == 0) then
        for j = 1 : 3 // Modify triad position or create it....
            // Now create the Triad vectors
            x = [Origin(1), Vec_End(1, j)];
            y = [Origin(2), Vec_End(2, j)];
            z = [Origin(3), Vec_End(3, j)];
            Path_Triad(j+1).data = [x; y; z]';
        end //...for j = 1 : 3 // Modify triad position or create it....
    end //...if (Dont_show == 0) then

else //...if exists('Path_Triad') then // Modify triad position or create it....
    disp("Path_Triad being created!");
    //Create A Visualisation Triad
    Path_Triad = tlist(["Path_Triad", "X_vec", "Y_vec", "Z_vec"], [], [], []);
    Line_Thickness = 2;
    Arrow_Size_Factor = 0.375;

    for j = 1 : 3  // creating triad vectors
        // Now create the Triad vectors
        x = [Origin(1), Vec_End(1, j)];
        y = [Origin(2), Vec_End(2, j)];
        z = [Origin(3), Vec_End(3, j)];
        param3d(x, y, z);
        Polyline_Hndl = get("hdl");
        Polyline_Hndl.polyline_style = 4;
        // 1-Solid; 2-Dash; 3-Dash dot style; 4-ends in Arrow
        Polyline_Hndl.line_style = 2;
        Polyline_Hndl.thickness = Line_Thickness;
        Polyline_Hndl.arrow_size_factor = Arrow_Size_Factor;
        Polyline_Hndl.foreground = Dark_Colours(j);
        Path_Triad(j+1) = Polyline_Hndl;
    end //...for j = 1 : 3  // creating triad vector

end //checks if exists('Path_Triad')
// *********************************************************************


// SET THE CURRENT PATH.......
if ~exists('Current_Path') then
    global All_paths;
    All_paths = ["H_line1";...
                 "H_line2";...
                 "V_line1";...
                 "V_line2";...
                 "R_line1";...
                 "R_line2"];

    global Current_Path;
    Current_Path = All_paths(6);

    Saved_Path = Dyn_Scr_Path + "Dynamics Data\Current_Path.dat";
    save(Saved_Path, Current_Path);
end // if ~exists('Current_Path')



global LP_indx;
// By default in the Dyn_Path_Create script we can make all paths invisible.
if (strcmp(Current_Path, "V_line1") == 0) then
    LP_indx = 1;

elseif (strcmp(Current_Path, "V_line2") == 0) then
    LP_indx = 2;
```

```
elseif (strcmp(Current_Path, "H_line1") == 0) then
    LP_indx = 3;

elseif (strcmp(Current_Path, "H_line2") == 0) then
    LP_indx = 4;

elseif (strcmp(Current_Path, "R_line1") == 0) then
    LP_indx = 5;

elseif (strcmp(Current_Path, "R_line2") == 0) then
    LP_indx = 6;

else
    disp("Selected path does not exist!")
    disp(Current_Path);
    abort();

end //...if (strcmp(Current_Path, "H_line1") == 0) then



// This creates the different types of paths...
drawlater;
Script_Path = Dyn_Scr_Path + "Dyn_Path_Create.sce";
exec(Script_Path, -1);


CP_Data = [];
CP_Data = L_Path(LP_indx).Data;





//Improve resolution on the path
if ~exists('Step_Res') then
    global Step_Res;
    Step_Res = 5;
end // if ~exists('Step_Res')


// Only do this for linear paths, all paths are linear
Min_Step_Len = Step_Res*0.005*L1a_len;
// 5mm - Changed this to make it faster.....
CP_Data = Improve_Path_Res(CP_Data, Min_Step_Len);
// improves the resolution on the PathFollowing



tmpBnd = (L1a_len + L1b_len + 1.5*L3_len);
LinkageTester.children($).data_bounds = ...
                        [-tmpBnd, -tmpBnd, -4; tmpBnd, tmpBnd, tmpBnd+5];
//[xmin ymin; xmax ymax]->must be 2 dimensional
global Current_View_Point;
LinkageTester.children($).rotation_angles = Current_View_Point;
LinkageTester.children($).box = "off";
drawnow;


// *******************************************************
// SOLVING INVERSE KINEMATICS
// Now we're going to move the Path_Triad along the path....
global TransVecAll;
global RotMatAll;
global All_Angles;
global Sol_Nr;

    // *****************************************************************
    // START: CHANGE CP_DATA SIZE - Added point at end
    // To have zero velocity on the end of the path we need to add a data point
    // at the end that is the same as the current last point! Doing this here
    // should not affect all the code that follows, as the size of animation_angs
    // will depend on this data!
    CP_Data(:, $+1) = CP_Data(:, $);
```

```
    Path_Size = size(CP_Data);
    // END: CHANGE CP_DATA SIZE - Added point at end
    // *********************************************************************

if ~exists('Animation_Angs') then // if it doesn't exist create it
    global Animation_Angs;
end //...if ~exists('Animation_Angs') then // if it doesn't exist create it
Animation_Angs = [];

if ~exists('Ani_idx') then // if it doesn't exist create it
    global Ani_idx;
end //...if ~exists('Ani_idx') then // if it doesn't exist create it
Ani_idx = 1; //Need to reset each time it comes in here!


if ~exists('Const_Tool_Orient') then // if it doesn't exist create it
    global Const_Tool_Orient;
    Const_Tool_Orient = 1; //constant orientation - 1, Changing orientation - 0
end //...if ~exists('Const_Tool_Orient') then // if it doesn't exist create it




drawlater();

if (Const_Tool_Orient == 1) then
    for p = 1 : Path_Size(2)//-1 // --Const_Tool_Orient = 1
        Origin(1) = CP_Data(1, p);
        Origin(2) = CP_Data(2, p);
        Origin(3) = CP_Data(3, p);

        Vec_End = Vec_Length*Tool_Frame + [Origin, Origin, Origin];

        TransVecAll(7) = Origin;
        RotMatAll(7) = Tool_Frame;
        exec(IK_Script_Path, -1); // UNCOMMENT
        // At this point we have all the angles
        Sol_Nr = Min_Wrist_Angs();

//        //********START: REMOVE THIS CODE********************
//            if( p >= 26) then
//                disp(All_Angles(1, :));
//                disp(All_Angles(2, :));
//            end //...if( p >= 26) then
//        //********END: REMOVE THIS CODE********************


        //Finds the set of angles for minimum rotation!
        // Save the correct solution, minimum change in wrist angles
        Animation_Angs(Ani_idx, :) = All_Angles(Sol_Nr, :);
        Ani_idx = Ani_idx + 1;

        if (Dont_show == 0) then
            for k = 1 : 3
                // Now adjust the path following Triad vectors position
                x = [Origin(1), Vec_End(1, k)];
                y = [Origin(2), Vec_End(2, k)];
                z = [Origin(3), Vec_End(3, k)];
                Path_Triad(k+1).data = [x; y; z]';
            end //...for k = 1 : 3
        end //...if (Dont_show == 0) then
    end // ...for p = 1 : Path_Size(2)-1  -- Const_Tool_Orient = 1
end //...if (Const_Tool_Orient == 1)



if (Dont_show == 0) then
    Update_Graphics_Model();
    //just update graphics one last time before exiting
end //...if (Dont_show == 0) then

drawnow();


// CONVERT TO METERS NOW
CP_Data = (1/1000)*CP_Data; //Convert to meters
```

```
// ********************************************************************
// SAVE ANGLES FOR DYNAMICS SCRIPT
// Save Animation Angles
if ~exists('Save_Data') then
    global Save_Data;
    Save_Data = 0;
end //...if ~exists('Save_Data') then


if (Save_Data == 1) & ~exists('Multi_Path') then

    Motor_Mass = [0, 1];
    lbl_Motor_Mass = ["LM", "HM"];
    lbl_Full_MM = ["light motors, ", "heavy motors, "]
    Tot_Time_P = [5, 5/20];

    // SCRIPT PATHS
    Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";
    Scr_Lin_Fol = Dyn_Scr_Path + "Dyn_Path_Fol_Lin.sce";
    Scr_Calc_VA = Dyn_Scr_Path + "Calc_Vel_Acc.sce";
    Scr_Inv_Dyn = Dyn_Scr_Path + "Inverse_Dynamics.sce";
    Scr_Calc_EU = Dyn_Scr_Path + "Calc_Energy_Used.sce";

    Location = Dyn_Scr_Path + "Dynamics Data\";
    Filename = Location + Current_Path + 'Angles_CP_Data.dat';
    save(Filename, Animation_Angs, CP_Data, Current_Path);
    // Saves the joint angles and the ToolEnd point locations.

    for i_MM = 1 : 2
        for i_TTP = 1 : 2

            global Heavy_motors;
            Heavy_motors = Motor_Mass(i_MM); //0 - light; 1 - heavy
            global Total_Play_Time;
            Total_Play_Time = Tot_Time_P(i_TTP); //5/20; // /20

            exec(Scr_Calc_VA, -1);
            exec(Scr_Inv_Dyn, -1);
            exec(Scr_Calc_EU, -1);

        end//...for i_TTP = 1 : 2
    end//...for i_MM = 1 : 2

end // Checking if (Save_Data == 1)
```

4.3 **File name:** Dyn_Path_Create.sce

Purpose: Creates the graphics for straight line dynamics paths.

```
// CREATE THE LINES FOR DYNAMICS PATHS
drawlater();
if exists('LinkageTester') then
    //show_window([LinkageTester]) // Raises a figure handle
else
    disp("Creating Figure!");
    ScreenSizePX = get(0, "screensize_px");
    //Returns screen size in pixels, on this PC it s 1920 by 1200
    //ScreenSizePT = get(0, "screensize_pt");
    //Returns screen size in points, on this PC it s 960 by 600
    global L1a_len;
    global L1b_len;
    global L3_len;
```

```
    L1a_len = 1000;
    L1b_len = 1066.2938; //1066.2937611; //1000;
    L3_len =  181.2938; //181.2937611;//  101.2937611; //100;

    //global LinkageTester;
    LinkageTester = figure(1); // Create figure having figure_id==1
    LinkageTester.figure_name = "Linkage Designer"
    LinkageTester.info_message = ""
    Win_Width = ScreenSizePX(3);
    Win_Height = ScreenSizePX(4);
    LinkageTester.figure_position = [0, 0];
    LinkageTester.figure_size = [Win_Width, Win_Height];
    LinkageTester.children(1).data_bounds = [-3, -3, 0; 3, 3, 3];

    close(figure(0));
end //...if exists('LinkageTester') then




if ~exists('Centre_Pos') then
    Centre_Pos = [0; L1b_len; 1.1*L1b_len];
    // Used [0; L1b_len; L1b_len] for vertical lines
    Centre_Pos = [Centre_Pos, [0; 1.7*L1b_len; 0.5*L1b_len]];
end //...if ~exists('Centre_Pos') then


sz_Centre_Pos = size(Centre_Pos);
end_vli =  sz_Centre_Pos(2);
end_hli = end_vli;



if ~exists('Half_Line_Disp') then
    Disp_Len = 0.6*L1b_len;
    // Half line displacement along Z, vertical lines
    Half_Line_Disp = [0; 0; 0.5*Disp_Len];
    // Adds half line displacement along X, horizontal lines
    Half_Line_Disp = [Half_Line_Disp, [0.5*Disp_Len; 0; 0]];
    // Adds half line displacement along Y, radial lines
    Half_Line_Disp = [Half_Line_Disp, [0; 0.5*Disp_Len; 0]];
end // ... if ~exists('Half_Line_Disp') then

sz_HLD = size(Half_Line_Disp);
end_hld_i = sz_HLD(2);


if ~exists('L_Path') then
    global L_Path;
    L_Path = list(0);
    path_data_struct = tlist(["Path", "Data", "Time", ...
                        "Graphic_handles"], [], [], []);

    sz_All_paths = size(All_paths);
    for vl_indx = 1 : sz_All_paths(1) // 1st one
        L_Path(vl_indx) = path_data_struct;
    end //...for vl_indx = 1 : sz_All_paths(1) // 1st one
end //...if ~exists() then




//Sets the visibility of the paths
see_paths = "off";

// CREATING PATHS ........
// Creating horizontal line path (same Z coordinate),
// its graphics and data
// Only enter if the graphics handle object is empty!
if (L_Path(1).Graphic_handles == [])
    //disp('Creating VL path graphics.')

    // Draw Line Path
    x = [1 2]; //Vl_Data(1, :);
    y = [3 4]; //Vl_Data(2, :);
```

```
    z = [5 6]; //Vl_Data(3, :);

    sz_All_paths = size(All_paths);
    for vl_indx = 1 : sz_All_paths(1)
        param3d(x, y, z);
        Polyline_Hndl = get("hdl");
        Polyline_Hndl.polyline_style = 1;
        // 1-Solid; 3-Dash dot style; 4-ends in Arrow
        Polyline_Hndl.line_style = 3;
        Polyline_Hndl.thickness = 2;//Line_Thickness;
        //9 - dark blue, 38 - black color('gray50');
        Polyline_Hndl.foreground = 9;
        Polyline_Hndl.visible = see_paths;
        L_Path(vl_indx).Graphic_handles = Polyline_Hndl;
    end //...for vl_indx = 1 : sz_All_paths(1)

end //...if (Vl_Path(1).Graphic_handles == [])



//sz_Centre_Pos = size(Centre_Pos);
//end_vli =  sz_Centre_Pos(2);
if ((~exists('LP_indx')) | (end_vli == 2)) then
    for hld_i = 1 : end_hld_i
        for vl_i = 1 : end_vli
            Vl_Centre_Pos = Centre_Pos(:, vl_i);
            Displacement = Half_Line_Disp(:, hld_i);
            Vl_Data = [Vl_Centre_Pos - Displacement,...
                       Vl_Centre_Pos + Displacement];

            vl_indx = (hld_i - 1)*end_vli + vl_i;
            L_Path(vl_indx).Data = Vl_Data;
            //convert to 2x3
            L_Path(vl_indx).Graphic_handles.data = Vl_Data';
        end //...for vl_i = 1 : end_vli
    end //...for hld_i = 1 : end_hld_i

else

    Vl_Data = [Centre_Pos - Half_Line_Disp, Centre_Pos + Half_Line_Disp];
    L_Path(LP_indx).Data = Vl_Data;
    L_Path(LP_indx).Graphic_handles.data = Vl_Data'; //convert to 2x3


end //... if ~exists('LP_indx') then



if (Dont_show == 0) then

    // Need to make the previous path invisible!
    if exists('Prev_ghdl') then
        Prev_ghdl.visible = "off";
    end
    // Show path
    L_Path(LP_indx).Graphic_handles.visible = "on";
    Prev_ghdl = L_Path(LP_indx).Graphic_handles;

end
```

4.4 **File name:** Calc_Vel_Acc.sce

Purpose: Creates the time vector and solves for velocity and acceleration (both end-effector and joints).

```
//Make all angles continuous about the 0-360 boundary, this prevents
```

```
//large spikes when calculating velocity and acceleration
Animation_Angs = Ang_Continuity(Animation_Angs);


SZ_Animation_Angs = size(Animation_Angs); //Animation_Angs has 6 columns
//disp('SZ_Animation_Angs = ');
//disp(SZ_Animation_Angs);

SZ_CP_Data = size(CP_Data); // CP_Data has 3 rows
if (SZ_Animation_Angs(1) ~= SZ_CP_Data(2)) then
    //Cannot continue as there wont be a 1 to 1 matching of data!
    err_msg = 'Rows of Animation_Angs ~= to columns of CP_Data;' + ...
              ' no 1-to-1 matching of data, cannot' + ...
              ' continue for path ' + Current_Path + '!';
    disp(err_msg);
    abort();
end //(SZ_Animation_Angs(1) ~= SZ_CP_Data(2)) then


// tHE CONVERSION TO METERS MUST HAPPEN IN DYN_PATH_FOL_LIN
// CP_Data = (1/1000)*CP_Data; //Convert to meters
Total_Dist = 0; //Resets total distance
for i = 1 : SZ_CP_Data(2)-1
    dx = CP_Data(1, i+1) - CP_Data(1, i);
    dy = CP_Data(2, i+1) - CP_Data(2, i);
    dz = CP_Data(3, i+1) - CP_Data(3, i);
    dl = (dx^2 + dy^2 + dz^2)^0.5;
    Total_Dist = Total_Dist + dl;
end
Half_Tot_Dist = 0.5*Total_Dist;




//*********************************************************************
// FINDING TIME; END-EFFECTOR VELOCITY AND END-EFFECTOR ACCELERATION
// VECTORS; NEED TO FIND CORRECT TIME VECTOR FOR STRAIGHT LINE PATHS

if ~exists('Total_Play_Time') then
    global Total_Play_Time
    Total_Play_Time = 5; // /20
elseif (Total_Play_Time == 0) then
    disp('Total_Play_Time = 0; cannot continue, at Cal_Vel_Acc!');
    abort();
end

Half_Time = Total_Play_Time/2;
Average_Vel = Total_Dist/Total_Play_Time;
Start_Vel = 0;
Mid_Vel = 2*Average_Vel;
End_Vel = 0;
Acc_FH = (Mid_Vel - Start_Vel)/Half_Time; //First half acceleration
Acc_SH = (End_Vel - Mid_Vel)/Half_Time; //Second half acceleration

//TIME VECTOR
Time_Vec = zeros(SZ_Animation_Angs(1), 1); //creates a vector of zeroes;
Time_Vec(1) = 0;
//Time_Vec(SZ_Animation_Angs(1)) = Total_Play_Time;

//VELOCITY VECTOR
Vel_Vec = zeros(SZ_Animation_Angs(1), 1); //creates a vector of zeroes;
Vel_Vec(1) = Start_Vel;
//Vel_Vec(SZ_Animation_Angs(1)) = End_Vel;

//ACCELERATION VECTOR
Acc_Vec = zeros(SZ_Animation_Angs(1), 1); //creates a vector of zeroes;
Acc_Vec(1) = Acc_FH;
//Acc_Vec(SZ_Animation_Angs(1)) = 0;


// s = ut +0.5at^2, but since u=0 s = 0.5at^2
// so for the first half t = (2s/a)^0.5
Dist = 0; //Reset Dist
//For second hald of curve, these don't change!!!
A = -0.5*Acc_SH;
B = -Mid_Vel;
Ax2 = 2*A;
```

```
end_i = SZ_Animation_Angs(1) - 1;
Circ_Tym_Inc = Total_Play_Time/(end_i - 1);

for i = 1: end_i
    dx = CP_Data(1, i+1) - CP_Data(1, i);
    dy = CP_Data(2, i+1) - CP_Data(2, i);
    dz = CP_Data(3, i+1) - CP_Data(3, i);
    dl = (dx^2 + dy^2 + dz^2)^0.5;
    Dist = Dist + dl;

    if (Total_Dist == 0) then
        //Could be that we're just rotating the end effector.
        Time_Vec(i+1) = i*Circ_Tym_Inc;
        Acc_Vec(i) = 0;  //End-effector position not moving
        Vel_Vec(i+1) = 0; //End-effector position not moving

    elseif (Dist == Total_Dist)  then
        Time_Vec(i+1) = Total_Play_Time;
        Acc_Vec(i) = Acc_SH;
        //v = u+at
        Vel_Vec(i+1) = Mid_Vel + Acc_SH*(Time_Vec(i+1) - Half_Time);
        //At full time

    elseif (0 < Dist)&(Dist < Half_Tot_Dist) then
        //s = Dist = u*t + 0.5*a*t^2, but u is 0...
        Time_Vec(i+1) = (2*Dist/Acc_FH)^0.5;
        Acc_Vec(i) = Acc_FH;
        Vel_Vec(i+1) = Acc_FH*Time_Vec(i+1); //v = u+at

    elseif (Half_Tot_Dist <= Dist)&(Dist <= Total_Dist) then
        // Now we have a quadratic that needs to be solved to
        // get the time
        // Dist - Half_Tot_Dist = Mid_Vel*t + 0.5*Acc_SH*t^2
        C = Dist - Half_Tot_Dist;
        Discriminant = (B^2 - 4*A*C)^0.5;
        //Remember time starts from Half_Time now
        t1 = (-B + Discriminant)/Ax2 + Half_Time;
        t2 = (-B - Discriminant)/Ax2 + Half_Time;


        if (t1>Total_Play_Time & t2>Total_Play_Time)|...
                        (t1<Half_Time & t2<Half_Time) then
            // Cannot continue as both can't be more than
            // Total_Time or less than Half_Time!
            err_msg = 'Both time values are greater than Total_Time' + ...
                      ' or less than Half_Time for i=' + string(i) + ...
                      'on path ' + Current_Path + '!';
            disp(err_msg);
            abort();
        elseif (t1>Half_Time & t2>Half_Time)&...
                        (t1<Total_Play_Time & t2<Total_Play_Time) then
            //Both solutions are valid and we can't choose!
            err_msg = 'Both solutions are valid for i=' + string(i) + ...
                      'on path ' + Current_Path + '!';
            disp(err_msg);
            abort();
        else
            if ((Half_Time <= t1) & (t1 <= Total_Play_Time)) then
                Time_Vec(i+1) = t1;
            elseif ((Half_Time <= t2) & (t2 <= Total_Play_Time)) then
                Time_Vec(i+1) = t2;
            else
                err_msg = 'Unknown error!';
                disp(err_msg);
                disp('t1 = ' + string(t1) + '; t2 =' + string(t2));
                abort();
            end
        end //... if (t1>Total_Play_Time & t2>Total_Play_Time)|...
                                    //(t1<Half_Time & t2<Half_Time) then

        Acc_Vec(i) = Acc_SH;
        //v = u+at
        Vel_Vec(i+1) = Mid_Vel + Acc_SH*(Time_Vec(i+1) - Half_Time);
        //Remember this graph goes from Half_Time to Full_Time

    end //...if (Total_Dist == 0) then
end //... for i = 1: end_i
```

```
// The last time value has been repeated we need to add some time
// here so that the software knows that it has stopped moving, same
// position value but different time values!!!!!
Time_Vec($) = Time_Vec($) + Total_Play_Time/100;
// FOUND TIME; END-EFFECTOR VELOCITY AND END-EFFECTOR ACCELERATION
// VECTORS
// ************************************************************************



// *******  FIND JOINT-ANGLE VELOCITY AND ACCELERATION VECTORS  *********
Ang_Vel_Mat = zeros(SZ_Animation_Angs(1),SZ_Animation_Angs(2));
//From this Ang_Vel_Mat(1,m) = [0 0 0 0 0 0]
//Ang_Vel_Mat2 = zeros(SZ_Animation_Angs(1),SZ_Animation_Angs(2));
Ang_Acc_Mat = zeros(SZ_Animation_Angs(1),SZ_Animation_Angs(2));


// First and last row must remain 0s!!! Start and end angular velocities
// are 0!!!
// Angular_Velocity = Change_in_Angle/Change_in_Time;

// ********** ANGULAR VELOCITIES !!!!!
end_k = SZ_Animation_Angs(1) - 1;
for k = 1: end_k // Finding Ang_Vel_Mat
    delta_t = Time_Vec(k+1)-Time_Vec(k);
    for m = 1:SZ_Animation_Angs(2)
        delta_ang = Animation_Angs(k+1,m) - Animation_Angs(k,m);
        // The change in angular velocity is given by delta_ang/delta_t,
        // we need to add this to the previous calculation to get the
        // current angular velocity!
        // Ang_Vel_Mat(k+1,m) = delta_ang/delta_t;
        if (delta_ang == 0) then //this take priority over delta_t = 0
            //No change in angle hence 0 angular velocity
            Ang_Vel_Mat(k+1,m) = 0;
        elseif (delta_t == 0) then
            // No time has past so the velocity should be the same as
            // the previous instant
            Ang_Vel_Mat(k+1,m) = Ang_Vel_Mat(k,m);
        else
            Ang_Vel_Mat(k+1,m) = 2*(delta_ang/delta_t) - Ang_Vel_Mat(k,m);
        end  // ...if (delta_ang == 0) then //this take priority over
             // delta_ang = 0
    end // ...for m = 1:SZ_Animation_Angs(2)
end // ... for k = 1: end_k // Finding Ang_Vel_Mat



// ********** ANGULAR ACCELERATIONS !!!!!
// Angular_Acceleration = Change_in_Angular_Acceleration/Change_in_Time;
for p = 1:(SZ_Animation_Angs(1)-1) //Angular acceleration
    delta_t = Time_Vec(p+1)-Time_Vec(p);
    for n = 1:SZ_Animation_Angs(2) //Angular acceleration
        delta_ang = Animation_Angs(p+1,n) - Animation_Angs(p,n);
        delta_vel = Ang_Vel_Mat(p+1,n) - Ang_Vel_Mat(p,n);

        if (delta_vel == 0) then ////this take priority over delta_t = 0
            //No change in angular velocity hence 0 angular acceleration.
            Ang_Acc_Mat(p,n) = 0;

        elseif (delta_t == 0) then
            // No time has past so the acceleration should be the same
            // as the previous instant
            Ang_Acc_Mat(p,n) = Ang_Acc_Mat(p-1,n);

        else //...if (delta_vel == 0) then ////this take priority over
                                            //delta_ang = 0
            Ang_Acc_Mat(p,n) = delta_vel/delta_t;

        end // ... if (delta_vel == 0) then ////this take priority over
                                            //delta_ang = 0

    end //for n = 1:SZ_Animation_Angs(2)  //Angular acceleration
end //for p = 1:(SZ_Animation_Angs(1)-1) //Angular acceleration


// FIX THE DATA FOR THE ACCELERATION CURVE SHOULD BE A STEP CURVE!!!!
```

```
// FOR PLOTTING IN LIBRE-OFFICE
nRows = 2*SZ_Animation_Angs(1)-1;
Time_Vec2 = zeros(nRows);
Ang_Acc_Mat2 = zeros(nRows, SZ_Animation_Angs(2));

Time_Vec2(1) = Time_Vec(1);
Ang_Acc_Mat2(1,:) = Ang_Acc_Mat(1,:);

for g = 1:SZ_Animation_Angs(1)-1
    Ang_Acc_Mat2(2*g,:) = Ang_Acc_Mat(g,:);
    Ang_Acc_Mat2(2*g+1,:) = Ang_Acc_Mat(g+1,:);
    Time_Vec2(2*g-1) = Time_Vec(g);
    Time_Vec2(2*g) = Time_Vec(g+1);
end

Time_Vec2(nRows) = Time_Vec(SZ_Animation_Angs(1));

// ****** FOUND JOINT-ANGLE VELOCITY AND ACCELERATION VECTORS  *********



// ********************************************************************
// ************    ORGANIZE ALL DATA FOR PLOTTING    ******************

// Make all angles continuous about the 0-360 boundary for plotting
// Ang_Continuity = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics...
// \Ang_Continuity_abt_0.sce";
//exec(Ang_Continuity, -1);
Animation_Angs = Ang_Continuity(Animation_Angs);


XYZ_Col_Vec = ['red', 'green', 'blue'];
Ang_Col_Vec = ['darkgray', 'blue', 'turquoise', 'green', 'orange', 'maroon'];
All_Plot_Data = list(0);
G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data = CP_Data;
G_Data.Colors = XYZ_Col_Vec;
G_Data.Legend = ['X', 'Y', 'Z'];
All_Plot_Data(1) = G_Data;

G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data = Animation_Angs'; //Note its the transpose of Animation_Angs
G_Data.Colors = Ang_Col_Vec;
G_Data.Legend = ['a1', 'a2', 'a3', 'a4', 'a5', 'a6'];
All_Plot_Data(2) = G_Data;

G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data =  Vel_Vec';
G_Data.Colors = 'blue';
G_Data.Legend = ['Z'];
All_Plot_Data(3) = G_Data;

G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data = Ang_Vel_Mat'; //Note its the transpose of Ang_Vel_Mat
G_Data.Colors = Ang_Col_Vec;
G_Data.Legend = ['a1', 'a2', 'a3', 'a4', 'a5', 'a6'];
All_Plot_Data(4) = G_Data;

G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data =  Acc_Vec';
G_Data.Colors = 'blue';
G_Data.Legend = ['Z'];
All_Plot_Data(5) = G_Data;

G_Data = tlist(["Graph_Data", "Data", "Line_Handles", "Colors", ...
                              "Leg_Hdl", "Legend"], [], [], [], [], []);
G_Data.Data = Ang_Acc_Mat'; //Note its the transpose of Ang_Acc_Mat
G_Data.Colors = Ang_Col_Vec;
G_Data.Legend = ['a1', 'a2', 'a3', 'a4', 'a5', 'a6'];
All_Plot_Data(6) = G_Data;
```

```
// ***************  ORGANIZED ALL DATA FOR PLOTTING   *******************


//*********************************************************************
//*********************** PLOT ALL DATA  ******************************
if ~exists('plot_data') then
    global plot_data;
    plot_data = 0;
end //...if ~exists('plot_data') then

if (plot_data == 1) then
    if(~exists('View_PVA'))then
        // CREATE FIGURE WITH SUB-PLOTS
        FigCre_Scpt = Dyn_Scr_Path + "View_Pos_Vel_Acc.sce";
        exec(FigCre_Scpt, -1);
    end //if(~exists('View_PVA'))then

    drawlater();
    end_i = size(All_Plot_Data)
    //PLOT POSITION CURVES
    L_Hdls = [];
    for axes_i = 1: end_i // selecting axis
        sca(View_PVA.children(axes_i))
        Col_Vec = All_Plot_Data(axes_i).Colors
        Sz_Data_Mat = size(All_Plot_Data(axes_i).Data);
        Line_Thickness = 2;

        for j = 1 : Sz_Data_Mat(1)
            if (axes_i == 6) then
                //Plots a step curve acceleration vector!
                plot(Time_Vec2, Ang_Acc_Mat2(:,j)');
            else //...if (j == 6) then
                plot(Time_Vec, All_Plot_Data(axes_i).Data(j,:));
            end  //...if (j == 6) then

            L_Hdls($+1) = get("hdl"); //Save line handle
            hdl = L_Hdls($);
            //hdl.children(1) gives access to the polyline entity.
            // 1-Solid; 3-Dash dot style; 4-ends in Arrow
            hdl.children(1).line_style = 1;
            hdl.children(1).thickness = Line_Thickness;
            hdl.children(1).foreground = color(Col_Vec(j));
            //9 - dark blue, 38 - black color('gray50');
            hdl.children(1).visible = "on";
        end //...for j = 1 : Sz_Data_Mat(1)

        All_Plot_Data(axes_i).Line_Handles = L_Hdls; //Save line handles
        Legd = legend(View_PVA.children(axes_i), ...
                      All_Plot_Data(axes_i).Legend, -4);
        All_Plot_Data(axes_i).Leg_Hdl = Legd;
        Legd.font_size = Axes_Fnt_Sz;
        Legd.visible = 'on';
        Legd.foreground = -1; //Black
        Legd.background = -2; //White
        Legd.font_style = 6; //Sans Serif close to arial
    end //...for axes_i = 1:end_i // selecting axis

    drawnow();

end //...if (plot_data == 1) then
//********************** PLOTTED ALL DATA  ****************************




// *************** SAVE DATA FOR PLOTTING LATER **********************
// SAVE POSITIONS, VELOCITIES AND ACCELERATIONS IN TEXT FORMAT TO BE
// DRAWN IN LIBRE-OFFICE
if ~exists('Save_Data') then
    global Save_Data;
    Save_Data = 0;
end //...if ~exists('Save_Data') then

if (Save_Data == 1) & ~exists('Multi_Path') then
```

```
All_paths = ["H_line1";...
             "H_line2";...
             "V_line1";...
             "V_line2";...
             "R_line1";...
             "R_line2"];

global Current_Path;
if ~strcmp(Current_Path, "H_line1") then
    lbl_heading = "Horizontal line 1"
    lbl_axis = 'X';
elseif ~strcmp(Current_Path, "H_line2") then
    lbl_heading = "Horizontal line 2"
    lbl_axis = 'X';
elseif ~strcmp(Current_Path, "V_line1") then
    lbl_heading = "Vertical line 1"
    lbl_axis = 'Z';
elseif ~strcmp(Current_Path, "V_line2") then
    lbl_heading = "Vertical line 2"
    lbl_axis = 'Z';
elseif ~strcmp(Current_Path, "R_line1") then
    lbl_heading = "Radial line 1"
    lbl_axis = 'Y';
elseif ~strcmp(Current_Path, "R_line2") then
    lbl_heading = "Radial line 2"
    lbl_axis = 'Y';
end


Headings = '';
Labels = '';

Labels = "Time\tX\tY\tZ\t";
Matrix_Data = [Time_Vec, CP_Data'];
Headings = Headings + lbl_heading + " XYZ positions (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t\t\t';
Labels = Labels + "Time\t" + lbl_axis + "_Vel\t";
Matrix_Data = [Matrix_Data, Time_Vec,  Vel_Vec];
Headings = Headings + lbl_heading + " XYZ velocities (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t';
Labels = Labels + "Time\t" + lbl_axis + "_Acc\t";
Matrix_Data = [Matrix_Data, Time_Vec,  Acc_Vec];
Headings = Headings + lbl_heading + " XYZ accelerations (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t';


Labels = Labels + "Time\tJ1\tJ2\tJ3\tJ4\tJ5\tJ6\t";
Matrix_Data = [Matrix_Data, Time_Vec, Animation_Angs];
Headings = Headings + lbl_heading + " joint angle positions (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t\t\t\t\t\t';

Labels = Labels + "Time\tJ1_Vel\tJ2_Vel\tJ3_Vel\tJ4_Vel\tJ5_Vel\tJ6_Vel\t";
Matrix_Data = [Matrix_Data, Time_Vec, Ang_Vel_Mat];
Headings = Headings + lbl_heading + " joint angle velocities (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t\t\t\t\t\t';

Labels = Labels + "Time\tJ1_Acc\tJ2_Acc\tJ3_Acc\tJ4_Acc\tJ5_Acc\tJ6_Acc\n";
Matrix_Data = [Matrix_Data, Time_Vec, Ang_Acc_Mat];
Headings = Headings + lbl_heading + " joint angle accelerations (" + ...
            lbl_Full_MM (i_MM) + ...
            string(Total_Play_Time) + 's)\t\t\t\t\t\t\n';


Labels = Headings + Labels;
File_Location = Dyn_Scr_Path + 'Dynamics Data\'+ Current_Path + ...
                '\Constant_Orientation_' + string(lbl_Motor_Mass(i_MM)) +...
                 '_' + string(Total_Play_Time) + 's\';
File_Name = 'Pos_Vel_Acc_Data.txt';
Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);
```

```
    File_Name = 'Ang_Acc Data.txt';
    Labels = "Time\tJoint1\tJoint2\tJoint3\tJoint4\tJoint5\tJoint6\n";
    Matrix_Data = [Time_Vec2, Ang_Acc_Mat2];
    Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);

//    //  SAVE VARIABLES ...
//    VL1 = File_Location + 'VelAcc.dat';
//    save (VL1, Time_Vec, Ang_Vel_Mat, Ang_Acc_Mat);
//    // Loads the joint angles and the ToolEnd point locations.


end //...if (Save_Data == 1) then

// ******************* SAVED DATA FOR PLOTTING *************************




// ***************** START: DISPLAY DATA ******************************
if ~exists('Display_Data') then
    Display_Data = 0;
end //...if ~exists('Display_Data') then

if (Display_Data == 1) then
    end_k = SZ_Animation_Angs(1) - 1;
    tmp_vec = [0 0 0 0 0 0];
    for k = 1: end_k // Finding Ang_Vel_Mat
        delta_t = Time_Vec(k+1)-Time_Vec(k);
        delta_ang = Animation_Angs(k+1,:) - Animation_Angs(k,:);
        tmp_vec = (2/delta_t)*delta_ang - tmp_vec ;
        txt_space = '   ';
        ThisString = string(delta_t) + txt_space + string(tmp_vec(1)) + ...
                                     txt_space + string(tmp_vec(2)) + ...
                                     txt_space + string(tmp_vec(3)) + ...
                                     txt_space + string(tmp_vec(4)) + ...
                                     txt_space + string(tmp_vec(5)) + ...
                                     txt_space + string(tmp_vec(6));
        disp (ThisString)
    end // ... for k = 1: end_k // Finding
end //...if (Display_Data == 1) then
// **************** END: DISPLAY DATA ***********************************
```

4.5 **File name:** View_Pos_Vel_Acc.sce

Purpose: Creates a figure with axes to display position, velocity and acceleration data.

```
// CREATE WINDOW TO VIEW POSIIONS VELOCITIES AND ACCELERATIONS
disp("Executing View_Pos_Vel_Acc!");
//Returns screen size in pixels, on this PC it s 1920 by 1200
ScreenSizePX = get(0, "screensize_px")
//Returns screen size in points, on this PC it s 960 by 600
//ScreenSizePT = get(0, "screensize_pt")

L1_len = 20;
L2_len = 1;
L1_len_B = L1_len + 0.9*L2_len;

global Current_Path;
if (Current_Path == []) then
    Current_Path = "V_line1";
end


View_PVA = figure(); // Create figure having figure_id==1
View_PVA.figure_name = 'Positions, velocities and accelerations for '+ ...
                       Current_Path;
View_PVA.info_message = ""
Win_Width = ScreenSizePX(3);
Win_Height = ScreenSizePX(4);
View_PVA.figure_position = [0, 0];
```

```
View_PVA.figure_size = [Win_Width, Win_Height];
View_PVA.background = 8; //White
toolbar(View_PVA,'off');

//GRAPH LABELS
//FIRST GRAPH OF END EFFECTOR POSITIONS, XYZ
//SECOND GRAPH OF JOINT ANGLE POSITIONS
Graph_Labels = list(0);
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "End-Effector Position, XYZ", "Time(s)", ...
               "Distance(m)", "");
Graph_Labels(1) = g_lbl;
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "Joint-Angle Positions", "Time(s)", ...
               "Angle(Degrees)", "");
Graph_Labels(2) = g_lbl;
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "End-Effector Velocity", "Time(s)", ...
               "Distance/Time(m/s)", "");
Graph_Labels(3) = g_lbl;
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "Joint-Angle Velocities", "Time(s)", ...
               "Angle/Time(Degrees/s)", "");
Graph_Labels(4) = g_lbl;
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "End-Effector Acceleration", "Time(s)", ...
               "Distance/Time^2(m/s^2)", "");
Graph_Labels(5) = g_lbl;
g_lbl = tlist(["Graph_Labels", "Axes_Title", "X_Label", "Y_Label", ...
               "Z_Label"], "Joint-Angle Accelerations", "Time(s)", ...
               "Angle/Time^2(Degrees/s^2)", "");
Graph_Labels(6) = g_lbl;


// Figure and Axes defaults
Axes_Fnt_Sz = 3;
Title_Fnt_Sz = 4;
Rows = 3;
Cols = 2;
Num_of_Axes = Rows*Cols;

drawlater();
// CREATE ALL AXES SUBPLOTS - If you don't do this it plots all axes
// on the same space!!!!
for i = 1 : (Num_of_Axes)
    drawnow();
    subplot(Rows, Cols, i);
end
// *********************************************************************


//SETSUP ALL AXES SUBPLOTS
drawlater();
for Axes_i = 1 : Num_of_Axes
    ///subplot(Rows, Cols, Axes_i);
    sca(View_PVA.children(Axes_i));
    View_PVA.children(Axes_i).axes_visible = ["on","on","off"];
    tmpBnd = (2*L1_len + 0.25*L2_len);
    View_PVA.children(Axes_i).grid = [1 1];
    View_PVA.children(Axes_i).font_size = Axes_Fnt_Sz;
    View_PVA.children(Axes_i).title.text = Graph_Labels(Axes_i).Axes_Title;
    View_PVA.children(Axes_i).title.font_size = Title_Fnt_Sz;
    View_PVA.children(Axes_i).x_label.font_size = Axes_Fnt_Sz
    View_PVA.children(Axes_i).x_label.text = Graph_Labels(Axes_i).X_Label;
    View_PVA.children(Axes_i).y_label.font_size = Axes_Fnt_Sz
    View_PVA.children(Axes_i).y_label.text = Graph_Labels(Axes_i).Y_Label;
    //[-0.0578526, 0.2];
    View_PVA.children(Axes_i).y_label.position = [-0.075, 0.2];
    //"on" Makes square axis look square
    View_PVA.children(Axes_i).isoview = "off";
    View_PVA.children(Axes_i).hidden_axis_color = 0;

end
drawnow();

drawlater();
// Setting the correct x position for the Y-Labels! Can only do this
```

```
// after it has been drawn
for Axes_i = 1 : Num_of_Axes
    View_PVA.children(Axes_i).y_label.position = [-0.075, 0.2];
end
drawnow();
```

4.6 **File name:** Save2_TXT.sci

Purpose: Formats and saves data to text files, in a tab separated list with new line characters at the end of each row. This is so that the data can be cut and pasted into Libre Office Base for plotting.

```
// Save data to text files....
// disp("-Debug line 173: Save2_TXT")
function [Result, Msg] = Save2_TXT(Location, File_Name, Matrix_Data, Labels)

    This_Fun_Name = "Save2_TXT:";
    Err_Msg = "";

    File_Name_and_Location = Location + File_Name;
    // Creates a string vector out of a matrix, where each element in the
    // vector, is a tab separated string of the column of the matrix.
    txt = strcat(string(Matrix_Data), "\t", "c");
    // txt = strsubst(txt, 'D', 'E');//  LibreOffice uses E for scientific
    // notation, and scilab saves it as D! The substitute string adds a
    // character that LibreOffice
    // reads as text.
    txt = Labels + strcat(txt, "\n");
    fd = mopen(File_Name_and_Location, 'wt');
    mfprintf(fd, txt);
    mclose(fd);

    Result = 1;
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

4.7 **File name:** Inverse_Dynamics.sce

Purpose: Calculates the joint angle torques for all models using the algorithm from Craig's 'Introduction to Robotics', which requires the joint angles, joint angular speed and joint angular acceleration for each pose in the path selected.

```
// KINEMATICS
Initial_Joint_Angles = [0  0   0;...
                        90 0   0;...
                        0  0   0;...
                        0  90  0;...
                        0  -90 0;...
                        0  90  0;...
                        0  -90 0];

global L1a_len;
global L1b_len;
global L3_len;

L1a_len = 1000;
L1b_len = 1066.2938; //1066.2937611;
L3_len =  181.2938; //181.2937611;
```

```
// IMPORTANT: Convert to meters
L1a_len = 1000;
L1_Meters = L1a_len/1000;
L1b_len = 1066.2938; //1066.2937611;
L2_Meters = L1b_len/1000;
L3_len =  181.2938; //181.2937611;
L3_Meters = L3_len/1000;


Link_Lengths = [0, 0, L1_Meters,  L2_Meters,   0, L3_Meters, 0];
// [F0-F1, F1-F2, F2-F3, F3-F4, F4-F5, F5-F6, F6-F7]

//Initialize Directions....
i_vec = [1; 0; 0];
j_vec = [0; 1; 0];
k_vec = [0; 0; 1];

// Setting the tool frame Orientation
Tool_Frame_Orient = Rotate_XYZ(-90, 0, 0);
// want a tool-frame whose x-axis is perpendicular to the XZ
// plane of the static base frame;
Tool_Frame = [0 0 1; 0 1 0; -1 0 0]';
Tool_Frame = Tool_Frame_Orient*Tool_Frame;
//disp('Tool_Frame = ');
//disp(Tool_Frame);

// Initialize structures
global Rrel_0;
Rrel_0 = list(0);


R = list(0);
R_list = list(0);
R_inv = list(0);
R_inv_list = list(0);
temp_Mat = tlist(["R", "Mat", "Rel_to"], [], 0);
w_list = list(0);


global PRel_0;
pVecRel_0 = list(0);
global P;
PVec = list(0);
temp_Loc = tlist(["PVec", "Vec", "Rel_to"], [], 0);



pos_chk_ind = []; //Matrix to store path indices which have an
// incorrect kinematic solution!
ori_chk_ind = [];
R0 = [1 0 0; 0 1 0; 0 0 1];
w0 = [0;0;0];
w0_dot = [0;0;0];
nu_dot_0 = [0; 0; 10]; //This term accounts for gravity, and
// acts in the opposite direction!
Deg_2_Rad = %pi/180;


//*********************************************************************
    // GET COM LOCATIONS, AND INERTIA TENSORS FOR DYNAMIC MODEL!!!!
    // Call individual dynamic model, to get relative location of
    // COMs in joint frames.


    // Best case: Scale_Factor=0.2
    // Most likely case: Scale_Factor=0.3
    // Worst case: Scale_Factor=0.45

    Scale_Factor = 0.15;
    // Material density
    steel_density = 7850; //kg/m^3
    copper_density = 8930; //kg/m^3
    ferrite_density = 4900; //kg/m^3
```

```
    // Run Script to generate common base and wrist modelling here!
    // Still to implement...

    //Display dynamic variables
    disp_dyn_vars = 0;


    global dyn_model;

    //****************************************
    //START TEST - REMOVE THIS PART OF THE CODE
        //Dyn_Pozzi.Mass(3) = Dyn_Pozzi.Mass(3) - delta_ml3;
    //END TEST - REMOVE THIS PART OF THE CODE
    //****************************************


    // GOT COMS AND INERTIA TENSORS FOR SELECTED DYNAMIC MODEL
//*************************************************************************

Dynmod_Torques = tlist(["Dynmod _Torques", "Pozzi", ...
                        "Serial", "Design2", "Design1"], [], [], [], []);
Dynmod_ni = tlist(["Dynmod _ni", "Pozzi", "Serial", ...
                   "Design2", "Design1"], [], [], [], []);
Dynmod_Ni = tlist(["Dynmod _Ni", "Pozzi", "Serial", ...
                   "Design2", "Design1"], [], [], [], []);
Dynmod_fi = tlist(["Dynmod _fi", "Pozzi", "Serial", ...
                   "Design2", "Design1"], [], [], [], []);
Dynmod_Fi = tlist(["Dynmod _Fi", "Pozzi", "Serial", ...
                   "Design2", "Design1"], [], [], [], []);


SZ_Animation_Angs = size(Animation_Angs); //Animation_Angs has 6 columns
end_k = SZ_Animation_Angs(1); //1
end_mod_cnt = 5;
for model_count = 1 : end_mod_cnt

    if (model_count == 1) then
        //disp(ascii(10)); //ascii(10) is the newline character
        //disp('DYNAMIC MODEL: Pozzi')
        DynMod = Dyn_Scr_Path + "DynMod_Pozzi.sce";
        exec(DynMod, -1);
        dyn_model = 'Pozzi';
        Mass = Dyn_Pozzi.Mass;
        PCom_Vec = Dyn_Pozzi.P_COM;
        I_tensor = Dyn_Pozzi.I_Tensor;

    elseif (model_count == 2) then
        Dynmod_Torques.Pozzi = jnt_torques;
        Dynmod_ni.Pozzi = n;
        Dynmod_Ni.Pozzi = N;
        Dynmod_fi.Pozzi = f;
        Dynmod_Fi.Pozzi = F;

        //disp(ascii(10)); //ascii(10) is the newline character
        //disp('DYNAMIC MODEL: Serial Arm')
        DynMod = Dyn_Scr_Path + "DynMod_Serial.sce";
        exec(DynMod, -1);
        dyn_model = 'Serial_arm';

        Mass = Dyn_Serial.Mass;
        PCom_Vec = Dyn_Serial.P_COM;
        I_tensor = Dyn_Serial.I_Tensor;

    elseif (model_count == 3) then
        Dynmod_Torques.Serial = jnt_torques;
        Dynmod_ni.Serial = n;
        Dynmod_Ni.Serial = N;
        Dynmod_fi.Serial = f;
        Dynmod_Fi.Serial = F;

        //disp(ascii(10)); //ascii(10) is the newline character
        //disp('DYNAMIC MODEL: Design_2')
        DynMod = Dyn_Scr_Path + "DynMod_Design2.sce";
        exec(DynMod, -1);
        dyn_model = 'Design_2';
        Mass = Dyn_Design2.Mass;
        PCom_Vec = Dyn_Design2.P_COM;
```

```
    I_tensor = Dyn_Design2.I_Tensor;

elseif (model_count == 4) then
    Dynmod_Torques.Design2 = jnt_torques;
    Dynmod_ni.Design2 = n;
    Dynmod_Ni.Design2 = N;
    Dynmod_fi.Design2 = f;
    Dynmod_Fi.Design2 = F;

    //disp(ascii(10)); //ascii(10) is the newline character
    //disp('DYNAMIC MODEL: Design_1')
    DynMod = Dyn_Scr_Path + "DynMod_Design1.sce";
    exec(DynMod, -1);
    dyn_model = 'Design_1';
    Mass = Dyn_Design1.Mass;
    PCom_Vec = Dyn_Design1.P_COM;
    I_tensor = Dyn_Design1.I_Tensor;

elseif (model_count == 5) then
    Dynmod_Torques.Design1 = jnt_torques;
    Dynmod_ni.Design1 = n;
    Dynmod_Ni.Design1 = N;
    Dynmod_fi.Design1 = f;
    Dynmod_Fi.Design1 = F;

    end_k = 0; //skip the next for loop

end //if (model_count == 1) then


//Reset main dynamics variables...
n = [];
N = [];
f = [];
F = [];
TT_params = list(0); // will be used to calc motor energy and torque!
for k = 1 : end_k

    // ************************************************************
    // CALCULATES ROTATION MATRICES RELATIVE TO BASE FRAME, ROTATION
    // MATRIX i+1 RELATIVE TO i, P VECTOR RELATIVE TO ORIGIN AND P
    // FOR FRAME i+1 RELATIVE TO i !!!!
    for i = 1 : 6
        angX = Initial_Joint_Angles(i,1); //Kinematic model params
        angY = Initial_Joint_Angles(i,2); //Kinematic model params
        angZ = Animation_Angs(k, i);

        temp_Mat.Mat = Rotate_XYZ(angX, angY, angZ);
        temp_Mat.Rel_to = i-1;
        R(i) = temp_Mat;

        temp_Loc.Vec = Link_Lengths(i)*i_vec;
        temp_Loc.Rel_to = i-1;
        PVec(i) = temp_Loc;

        if (i == 1) then
            Rrel_0(i) = R(i).Mat;
            pVecRel_0(i) = R0*PVec(i).Vec;

        else // if (i == 1) then
            // from joints 2 to 6
            Rrel_0(i) = Rrel_0(i-1)*R(i).Mat; //Rx*Ry*Rz;
            pVecRel_0(i) = pVecRel_0(i-1) + Rrel_0(i-1)*PVec(i).Vec;

        end //... if (i == 1) then


        if (i == 3) then // ...get knee coordinates
            Knee_Coord = pVecRel_0(i);
            //disp('Knee Coordinates = ');
            //disp(Knee_Coord);
        elseif (i == 4) then //...if (i == 3) then // ...get knee C
            Wrist_Org_Coord = pVecRel_0(i);
            //disp('Wrist Origin = ');
            //disp(Wrist_Org_Coord);
            //disp('CP_Data = ')
            //disp(CP_Data(:,1)')
```

```
    end //...if (i == 3) then // ...get knee coordinates


end //...for i = 1 : 6

//When i = 7
i = 7;
angX = Initial_Joint_Angles(7,1);
angY = Initial_Joint_Angles(7,2);
angZ = Initial_Joint_Angles(7,3);

temp_Mat.Mat = Rotate_XYZ(angX, angY, angZ);
temp_Mat.Rel_to = 6;
R(7) = temp_Mat;

temp_Loc.Vec = Link_Lengths(i)*i_vec;
temp_Loc.Rel_to = 6;
PVec(7) = temp_Loc;

Rrel_0(7) = Rrel_0(6)*Rotate_XYZ(angX, angY, angZ); //Rx*Ry*Rz;
pVecRel_0(7) = pVecRel_0(6) + Rrel_0(6)*PVec(7).Vec;

// CALCULATES ROTATION MATRICES RELATIVE TO BASE FRAME, ROTATION
// MATRIX i+1 RELATIVE TO i, P VECTOR RELATIVE TO ORIGIN AND P
// FOR FRAME i+1 RELATIVE TO i !!!!
// ************************************************************

chk_kinematics = 0;
if (chk_kinematics == 1) then //First
// ************************************************************
    // CHECKING THAT KINEMATIC SOLUTION IS RIGHT
    //Check end-effector position
    // convert CP_Data(:, k) to meters before subtracting
    d_vec = pVecRel_0(7) - 0.001*CP_Data(:, k); //column vector!
    dl_chk = (d_vec')*d_vec;
    if (dl_chk > 0.00001) then
        pos_chk_ind($+1) = k; //Add path position index!
    end //...if (dl_chk > 0.00001) then


    //Check end-effector orientation
    dl_chk = 0;
    for chk_row = 1:3
        for chk_col = 1:3
            dl_chk = dl_chk + (Rrel_0(7)(chk_row, chk_col) - ...
                               Tool_Frame(chk_row, chk_col))^2;
        end // for chk_col = i:3
    end //for chk_row = 1:3

    if (dl_chk > 0.00001) then
        ori_chk_ind($+1) = k; //Add path position index!
    end //(dl_chk > 0.00001) then
    // CHECKED THAT KINEMATIC SOLUTION WAS RIGHT
    // ********************************************************
end //if (chk_kinematics == 1) then //First


//************************************************************
// IF DYN_MODEL LINKS CHANGE MASS DISTRIBUTION THEN WE MUST CHANGE
// COM LOCATIONS, AND INERTIA TENSORS ACCORDINGLY!!!!
// THIS ONLY HAPPENS FOR dyn_model = 'Design_1';

if ~strcmp(dyn_model, 'Design_1') then
    DynMod_ScrPth = "D:\Program Files\scilab-5.3.3\SCI_Work"+ ...
                    "\HKM Dynamics\DynMod_Design1.sce";
    exec(DynMod_ScrPth, -1);
end
// ************************************************************




// GOT COM LOCATIONS, AND INERTIA TENSORS FOR DYNAMIC MODEL!!!!
//************************************************************
// Masses are stored in Mass() - A Vector
// Inertia tensors in I_tensor - A list object containing matrices
// Centre of Mass locations are stored in PCom_Vec - A list object
```

```
// containing vectors. IMPORTANT!!!!! - Remember your P and PCom
// vectors must be in meters! Also all angles must be measured in
// radians!

// ************************************************************
// STARTING OUTWARD ITERATIONS FOR DYNAMICS ALGORITHM
//'Ang_Vel_Mat', 'Ang_Acc_Mat'
end_i = 5; //Should be 5!
for i = 0 : end_i
    if (i == 0) then
        wi =  w0;
        wi_dot = w0_dot;
        //This initialization takes care of gravity!
        nu_dot_i = nu_dot_0;
    else //if (i == 0) then
        wi =  w(:, i);
        wi_dot = w_dot(:, i);
        nu_dot_i = nu_dot(:, i);
    end //if (i == 0) then

    // inverse of a rotation matirx is its transpose! And we want

    // R i with respect to i+1
    R_inv(i+1) = R(i+1).Mat';
    R_inv_w = R_inv(i+1)*wi;
    theta_dot_z = Deg_2_Rad*Ang_Vel_Mat(k, i+1)*k_vec;
    //Need to convert to radians
    theta_Ddot_z = Deg_2_Rad*Ang_Acc_Mat(k, i+1)*k_vec;

    w(:,i+1) =  R_inv_w + theta_dot_z;

    w_dot(:,i+1) = R_inv(i+1)*wi_dot + ...
                CrossMult(R_inv_w, theta_dot_z)' + theta_Ddot_z;


    wi_dot_x_P = CrossMult(wi_dot, PVec(i+1).Vec)';
    wi_x_P = CrossMult(wi, PVec(i+1).Vec)';
    wi_x_wi_x_P = CrossMult(wi, wi_x_P)';
    nu_dot(:, i+1) = R_inv(i+1)*(wi_dot_x_P + wi_x_wi_x_P + ...
                                                nu_dot_i);

    wi1_dot_x_PCi1 = CrossMult(w_dot(:,i+1), PCom_Vec(i+1))';
    wi1_x_PCi1 = CrossMult(w(:,i+1), PCom_Vec(i+1))';
    wi1_x_wi1_x_PCi1 = CrossMult(w(:,i+1), wi1_x_PCi1)';
    nu_C_dot(:, i+1) = wi1_dot_x_PCi1 + wi1_x_wi1_x_PCi1 + ...
                                                nu_dot(:, i+1);

    F(:,i+1) = Mass(i+1)*nu_C_dot(:, i+1);

    Ii1_wi1 = I_tensor(i+1)*w(:,i+1);
    wi1_x_Ii1_wi1 = CrossMult(w(:,i+1), Ii1_wi1)';
    N(:,i+1) = I_tensor(i+1)*w_dot(:,i+1) + wi1_x_Ii1_wi1;

end //for i = 0 : end_i

// COMPLETED OUTWARD ITERATIONS FOR DYNAMICS ALGORITHM
// ************************************************************




// ************************************************************
// STARTING INWARD ITERATIONS FOR DYNAMICS ALGORITHM
for i = (end_i+1) : -1:  1
    if (i == 6) then
        fi1 = [0; 0; 0]; //Force exerted by robot on environment!
        ni1 = [0; 0; 0]; //Moment exerted by robot on environment!
    else // if (i == 6) then
        fi1 = f(:, i+1);
        ni1 = n(:, i+1);
    end //(i == 6) then

    Ri1_fi1 = R(i+1).Mat*fi1;
    f(:,i) = Ri1_fi1 + F(:, i);

    PCi_x_Fi = CrossMult(PCom_Vec(i), F(:,i))';
    Ri1_ni1 =  R(i+1).Mat*ni1;
```

```
            Pi1_x_Ri1_fi1 = CrossMult(PVec(i+1).Vec, Ri1_fi1)';
            n(:,i) = N(:,i) + Ri1_ni1 + PCi_x_Fi + Pi1_x_Ri1_fi1;

            jnt_torques(k,i) = n(:,i)'*k_vec;


        end //i = (end_i+1) : -1:  1

        // COMPLETED INWARD ITERATIONS FOR DYNAMICS ALGORITHM
        // ************************************************************

    end // for k = 1:SZ_Animation_Angs(1)




    if (chk_kinematics == 1) then //Second
    // ********************************************************************
        // CHECKING THAT KINEMATIC SOLUTION IS RIGHT
        if (pos_chk_ind == []) then
            disp('All positions verified!!!')
        else //if (pos_chk_ind == []) then
            disp('All positions NOT verified!!!')
        end  //if (pos_chk_ind == []) then

        if (ori_chk_ind == []) then
            disp('All orientations verified!!!')
        else  //if (ori_chk_ind == []) then
            disp('All orientations NOT verified!!!')
        end //if (ori_chk_ind == []) then
    end //if (chk_kinematics == 1) then //Second

end // for model_count = 1 : end_mod_cnt



// ****************************************
// START: Save torque data to text files....
// Save Animation Angles
if ~exists('Save_Data') then
    global Save_Data;
    Save_Data = 0;
end //...if ~exists('Save_Data') then

if ~exists('test_static') then
    test_static = 0; //0 - > dynamic data
end

if (Save_Data == 1) & ~exists('Multi_Path') then

    if (test_static == 0) then // Only save if its dynamic data!

        end_jnt_cnt = 3; //6;
        Labels = '';
        Headings = '';
        Matrix_Data = [];
        End_Char = ["\t", "\t", "\n"];
        for jnt_cnt = 1 : end_jnt_cnt //...saving data
            //File_Name = 'J' + string(jnt_cnt) + '_Torques_Dynamic.txt';

            //Time_Vec, Dynmod_Torques
            PJ_i = Dynmod_Torques.Pozzi(:,jnt_cnt);
            SJ_i = Dynmod_Torques.Serial(:,jnt_cnt);
            D1J_i = Dynmod_Torques.Design1(:,jnt_cnt);
            D2J_i = Dynmod_Torques.Design2(:,jnt_cnt);
            Matrix_Data = [Matrix_Data , Time_Vec, PJ_i, SJ_i, D1J_i, D2J_i];
            Labels = Labels + "Time\t" + "Pozzi_JT" + string(jnt_cnt) +"\t" + ...
            + "Serial_JT" + string(jnt_cnt) +"\t" + ...
            + "Design1_JT" + string(jnt_cnt) +"\t" + ...
            + "Design2_JT" + string(jnt_cnt) + End_Char(jnt_cnt); //"\n";

            Headings = Headings + lbl_heading + " joint" + string(jnt_cnt) + ...
                       " torques (" + lbl_Full_MM (i_MM) + ...
                       string(Total_Play_Time) + 's)\t\t\t\t' + ...
                       End_Char(jnt_cnt);
```

```
        end // ...for model_count = 1 : end_mod_cnt //...saving data


        File_Location = Dyn_Scr_Path + 'Dynamics Data\'+ Current_Path + ...
                    '\Constant_Orientation_' + string(lbl_Motor_Mass(i_MM)) +...
                     '_' + string(Total_Play_Time) + 's\';
        Labels = Headings + Labels;
        File_Name = 'Joint_Torques.txt';
        Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);


    else //...if (test_static == 0) then // Only save if its dynamic
        disp('Will not save data as it is for static poses!');

    end //...if (test_static == 0) then // Only save if its dynamic


    // Now that we have all Joint torques, ni, Ni, fi and Fi, we can
    // store them and work on them later!!!!

//    if (test_static == 0) then
//        // Saves Timevec, Torques and Forces.
//        File_Name_and_Location = File_Location + ...
//                                          'Forces_Torques_Dynamic.dat';
//        save (File_Name_and_Location, Time_Vec, Dynmod_Torques, ...
//                        Dynmod_ni, Dynmod_Ni, Dynmod_fi, Dynmod_Fi);
//
//        File_Name_and_Location = File_Location + 'Design_Models.dat';
//        save (File_Name_and_Location, Dyn_Pozzi, Dyn_Serial, ...
//                        Dyn_Design2, Dyn_Design1);
//
//        File_Name_and_Location = File_Location + 'Design1_TT_Configs.dat';
//        save (File_Name_and_Location, Time_Vec, TT_params);
//
//    end //...if (test_static == 0) then
end //...if (Save_Data == 1) then
// END: Save torque data to text files....
// ****************************************
```

4.8 **File name:** DynMod_Pozzi.sce

Purpose: Finds the mass, inertia tensors and locates the centre of mass for each link in the Pozzi model.

```
//DynMod_Pozzi

if ~exists('Scale_Factor') then
    Scale_Factor = 0.15
end

if ~exists('L1_Meters') then
    L1a_len = 1000;
    L1_Meters = L1a_len/1000;
end



// Link 1
d1 = Scale_Factor*L1_Meters;
d2 = 2/3*d1;
d3 = 1/3*d1;


if ~exists('steel_density') then
    // Material density
```

```
    steel_density = 7850; //kg/m^3
    copper_density = 8930; //kg/m^3
    ferrite_density = 4900; //kg/m^3
end
rl1 = d1;
hl1 = d1/5;
ml1 = %pi*(rl1^2)*hl1*steel_density;



// Link 2, torque transfer tubes and elbow gearing

r_p = zeros(1,6);
r_pm = zeros(1,5);
r_p(6) = 0.5*(2/3)*rl1;
r_p(1) = (1/3)*r_p(6);

// Sets the radii of the prox link tubes
dr = (r_p(6) - r_p(1))/(6-1);
for pz_i = 2:5 //Sets r_p radii
    r_p(pz_i) = r_p(pz_i-1)+dr;
end //for pz_i = 2:5 //Sets r_p radii


// Sets the mass of the proximal link tubes
prx_TTI_Tensor = list(0);
temp_mass = 0;
for pz_i = 1:5 // Sets r_pm mass

      // Using the same material densitiy for all models
    r_pm(pz_i) = %pi*(r_p(pz_i+1)^2 - r_p(pz_i)^2)*L1_Meters*steel_density;

    Ixx = 6*(r_p(pz_i+1)^2 + r_p(pz_i)^2);
    Iyy = 3*(r_p(pz_i+1)^2 + r_p(pz_i)^2) + L1_Meters^2;
    Izz = Iyy;
    prx_TTI_Tensor(pz_i) = r_pm(pz_i)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

    temp_mass = temp_mass + r_pm(pz_i);
end //for pz_i = 1:5 // Sets r_pm mass
ml2 = temp_mass;



//ELBOW GEARING
h_eg = 0.8*hl1;
r_eg = zeros(1,5);
r_egm = zeros(1,4);

// Sets the radii of the elbow gearing
for pz_i = 1:4 //Sets r_eg radii
    r_eg(pz_i+1) = (d1-d2) - r_p(6-pz_i);
end //for pz_i = 1:4 //Sets r_eg radii

dr = r_eg(3) - r_eg(2);
r_eg(1) = r_eg(2) - dr;
if (r_eg(1) < 0) then
    r_eg(1) = 0
end //if (r_d(1) < 0) then //Elbow gearing

// Sets the mass of the elbow gearing
egm_TTI_Tensor = list(0);
temp_mass = 0
for pz_i = 1:4 // Sets r_egm mass

    // Using the same material densitiy for all models
    r_egm(pz_i) = %pi*(r_eg(pz_i+1)^2 - r_eg(pz_i)^2)*h_eg*steel_density;

    Ixx = 6*(r_eg(pz_i+1)^2 + r_eg(pz_i)^2);
    Iyy = 3*(r_eg(pz_i+1)^2 + r_eg(pz_i)^2) + h_eg^2;
    Izz = Iyy;
    egm_TTI_Tensor(pz_i) = r_egm(pz_i)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

    temp_mass = temp_mass + r_egm(pz_i);
end //for pz_i = 1:4 // Sets r_egm mass
meg = temp_mass;
```

```
// Link 3 and torque transfer tube params
r_d = zeros(1,5);
r_dm = zeros(1,4);

// Sets the radii of the distal link tubes
for pz_i = 2:5 //Sets r_d radii
    r_d(pz_i) = (d2-d3) - r_eg(7-pz_i);
end //for pz_i = 2:5 //Sets r_d radii

dr = r_d(3) - r_d(2);
r_d(1) = r_d(2) - dr;
if (r_d(1) < 0) then
    r_d(1) = 0
end //if (r_d(1) < 0) then // Link3


if ~exists('L2_Meters') then
    L1b_len = 1066.2938; //1066.2937611;
    L2_Meters = L1b_len/1000;
end //if ~exists('L2_Meters') then

// Sets the mass of the distal link tubes
dst_TTI_Tensor = list(0);
temp_mass = 0;
for pz_i = 1:4 // Sets r_dm mass

     // Using the same material densitiy for all models
    r_dm(pz_i) = %pi*(r_d(pz_i+1)^2 - r_d(pz_i)^2)*L2_Meters*steel_density;

    Ixx = 6*(r_d(pz_i+1)^2 + r_d(pz_i)^2);
    Iyy = 3*(r_d(pz_i+1)^2 + r_d(pz_i)^2) + L2_Meters^2;
    Izz = Iyy;
    dst_TTI_Tensor(pz_i) = r_dm(pz_i)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

    temp_mass = temp_mass + r_dm(pz_i);
end //for pz_i = 1:4 // Sets r_dm mass
ml3 = temp_mass;



    if ~exists('disp_dyn_vars') then
        disp_dyn_vars = 0;
    end //...if ~exists('disp_dyn_vars') then


    if (disp_dyn_vars == 1) then //Prox link vars
        //ascii(9) is the horizontal tab!
        tab_space = ascii(9);// + ascii(9);
        disp('POZZI - Dynamic Params...')
        disp('d1 = Scale_Factor*L1_Meters = ' + string(d1) + ...
                   tab_space + 'rl1 = d1 = ' +  string(rl1));
        disp('d2 = 2/3*d1 = ' + string(d2) + tab_space + ...
                   'd3 = 1/3*d1 = ' + string(d3));
        disp('hl1 = d1/5 = ' + string(hl1));

        disp('ml1 = %pi*(rl1^2)*hl1*steel_density = ' + string(ml1));
        disp('ml2 = ' + string(ml2) + '; ml3 = ' + string(ml3) + ...
                   '; meg = ' + string(meg));

        disp('Proximal link:  Link Ro = 0.5*(2/3)*rl1 = ' + ...
                   string(r_p(6)) + '; Link Ri = (1/3)*r_p(6) = ' + ...
                   string(r_p(1)) );
        disp('Distal link:  Link Ro = ' + string(r_d(5)) + ...
                   '; Link Ri = ' + string(r_d(1)) );
    end //... if (disp_dyn_vars == 1) then //Prox link vars



// ************************************************************
// START I_L1Complete
```

```
I_tensor = list(0);
PCom_Vec = list(0);
Mass = [];
Mass(1) = ml1;
PCom_Vec(1) = [0;0;0];

Ixx = 3*rl1^2 + hl1^2;
Iyy = Ixx;
Izz = 6*rl1^2;
Ibg1 = ml1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

I_L1Complete = Ibg1;
I_tensor(1) = I_L1Complete;
// END I_L1Complete
// ************************************************************




// ************************************************************
// START I_L2Complete
//*****************************
//START TEST: Remove this code!!!
//meg = 0;
//END TEST: Remove this code!!!!
//*****************************
Mass(2) = ml2 + meg;
dx = 0.5*meg*L1_Meters/Mass(2);
dy = 0;
dz = meg*(d1-d2)/Mass(2);

PCom_Vec(2) = [0.5*L1_Meters; 0; d1] + [dx; 0; -dz];


Ixx = 6*(r_p(6)^2 + r_p(1)^2);
Iyy = 3*(r_p(6)^2 + r_p(1)^2) + L1_Meters^2;
Izz = Iyy;
I_pls = ml2/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = PCom_Vec(2) - [0.5*L1_Meters; 0; d1];

if ~exists('Eye_3x3') then
    Eye_3x3 = [1 0 0; 0 1 0; 0 0 1];
end //...if ~exists('Eye_3x3') then
I_pls_at_COM = RotTrans_IT(I_pls, ml2, drr_vec, Eye_3x3);

Ixx = 6*(r_eg(5)^2 + r_eg(1)^2);
Iyy = 3*(r_eg(5)^2 + r_eg(1)^2) + h_eg^2;
Izz = Iyy;
I_egs = meg/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [L1_Meters; 0; d2] - PCom_Vec(2);
I_egs_at_COM = RotTrans_IT(I_egs, meg, drr_vec, Eye_3x3);


I_L2Complete = I_pls_at_COM + I_egs_at_COM;
I_tensor(2) = I_L2Complete;
// END I_L2Complete
// ************************************************************







// ************************************************************
// START I_L3Complete
//*****************************
//START TEST: Remove this code!!!
//meg = 0;
//ml3 = 0;
//END TEST: Remove this code!!!!
//*****************************
Mass(3) = ml3;
PCom_Vec(3) = [0.5*L2_Meters; 0; d3];
```

```
Ixx = 6*(r_d(5)^2 + r_d(1)^2);
Iyy = 3*(r_d(5)^2 + r_d(1)^2) + L2_Meters^2;
Izz = Iyy;
I_dls = ml3/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

I_L3Complete = I_dls;
I_tensor(3) = I_L3Complete;
// END I_L3Complete
// *************************************************************
```

```
// *************************************************************
// START I_L4,5 Complete
Mass(4) = 0;
Mass(5) = Mass(4);
PCom_Vec(4) = [0;0;0];
PCom_Vec(5) = PCom_Vec(4);
I_L4Complete = [0 0 0; 0 0 0; 0 0 0];
I_L5Complete = I_L4Complete;
I_tensor(4) = I_L4Complete;
I_tensor(5) = I_L5Complete;
// END I_L4,5 Complete
// *************************************************************
```

```
// *************************************************************
// START I_WRIST
//Wrist Params

if ~exists('L3_Meters') then
    L3_len =  181.2938; //181.2937611;
    L3_Meters = L3_len/1000;
end
rw = 0.25*L3_Meters;

m_wrist = %pi*(rw^2)*L3_Meters*steel_density; //10
//*******************************
//START TEST: Remove this code!!!
//m_wrist = 0;
//END TEST: Remove this code!!!!
//*******************************
Mass(6) = m_wrist


// In the kinematic solution we coded the 6th axis coordinate frame
// lies at the tool point. Hence to get its centre of mass we must
// travel 0.5*L3_Meters in the negative Z6 direction!!!
PCom_Vec(6) = [0; 0; -0.5*L3_Meters];
Ixx = 3*rw^2 + L3_Meters^2;
Iyy = Ixx
Izz = 6*rw^2;
I_L6Complete = Mass(6)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
I_tensor(6) = I_L6Complete;
// END I_WRIST
// *************************************************************
```

```
// Store Dynamic model parameters
Dyn_Pozzi = tlist(["Dyn_Model", "I_Tensor", "Mass", "P_COM", ...
                   "L2_components", "L3_components"], [], [], [], [], []);
L2_parts = tlist(["Pzzi_L2_Parts", "Tube_Radii", "Tube_Masses", ...
                  "prx_ttI_Tensor", "EG_Radii", "EG_Masses", ...
```

```
                "egm_ttI_Tensor"], [], [], [], [], [], []);
L3_parts = tlist(["Pzzi_L3_Parts", "Tube_Radii", "Tube_Masses", ...
                "dst_ttI_Tensor"], [], [], []);

L2_parts.Tube_Radii = r_p;
L2_parts.Tube_Masses = r_pm;
L2_parts.prx_ttI_Tensor = prx_TTI_Tensor;
L2_parts.EG_Radii = r_eg;
L2_parts.EG_Masses = r_egm;
L2_parts.egm_ttI_Tensor = egm_TTI_Tensor;


L3_parts.Tube_Radii = r_d;
L3_parts.Tube_Masses = r_dm;
L3_parts.dst_ttI_Tensor = dst_TTI_Tensor;



Dyn_Pozzi.I_Tensor = I_tensor;
Dyn_Pozzi.Mass = Mass;
Dyn_Pozzi.P_COM = PCom_Vec;
Dyn_Pozzi.L2_components = L2_parts;
Dyn_Pozzi.L3_components = L3_parts;
```

4.9 **File name:** DynMod_Serial.sce

Purpose: Finds the mass, inertia tensors and locates the centre of mass for each link of the Serial robot model.

```
if ~exists('Dyn_Pozzi') then

    Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";
    DynMod = Dyn_Scr_Path + "DynMod_Pozzi.sce";
    exec(DynMod, -1);
end


// Motor2 mass in kg; model this using copper, iron, steel and empty
// space densities.
pcntage_steel = 0.15;
pcntage_copper = 0.4;
pcntage_ferrite = 0.4;
pcntage_space = 0.05;
Motor_composition = pcntage_steel*steel_density + ...
                    pcntage_copper*copper_density +...
                    pcntage_ferrite*ferrite_density;

if ~exists('Heavy_motors') then
    global Heavy_motors;
    Heavy_motors = 0;
end

Heavy_motors = 1;
if (Heavy_motors == 1) then //HEAVY MOTORS

    // Here we set motor masses as a percentage of link masses...ml1
    // Link 1 and Motor 2 Params
    //L_m2 = d1; //Length of motor 2
    //L_m3 = 0.8*L_m2; //Length of motor 3
    //L_m4 = 0.8*L_m3; //Length of motor 4

    // Mass Motor 4
    mm4 = 0.35*ml3
    Vol_m4 = mm4/Motor_composition;
    // B_m4 = 0.4*L_m4; //Breadth of motor 4
    // W_m4 = 4*B_m4;
    // Vol_m4 = B_m4*W_m4*L_m4 = 0.64*L_m4^3;
    //=> L_m4 = (Vol_m4/0.64)^(1/3)
    L_m4 = (Vol_m4/0.64)^(1/3);
    B_m4 = 0.4*L_m4;//(0.25*Vol_m4/L_m4)^0.5;
    W_m4 = 4*B_m4;
```

```
    // Mass Motor 3
    //W_m3 = 0.4*L_m3; //Width of motor 3
    //Vol_m3 = W_m3*W_m3*L_m3 = 0.16*L_m3^3;
    //=> L_m3 = (Vol_m3/0.16)^(1/3)
    mm3 = 0.35*(ml3 + mm4);
    Vol_m3 = mm3/Motor_composition;
    L_m3 = (Vol_m3/0.16)^(1/3)
    W_m3 = 0.4*L_m3;//(Vol_m3/L_m3)^0.5;

    // Mass Motor 2
    //W_m2 = 0.4*L_m2; //Width of motor 2, width and breath are equal
    //Vol_m2 = W_m2*W_m2*L_m2 = 0.16*L_m2^3
    //=> L_m2 = (Vol_m2/0.16)^(1/3)
    mm2 = 0.35*(ml2 + ml3 + mm3 + mm4);
    Vol_m2 = mm2/Motor_composition;
    L_m2 = (Vol_m2/0.16)^(1/3);
    W_m2 = 0.4*L_m2;//(Vol_m2/L_m2)^0.5; //Width of motor 2


else //... LIGHT MOTORS

    // Link 1 and Motor 2 Params
    L_m2 = d1; //Length of motor 2
    W_m2 = 0.4*L_m2; //Width of motor 2, width and breath are equal
    Vol_m2 = W_m2*W_m2*L_m2;
    mm2 = Motor_composition*Vol_m2;

    // Link 2 and Motor 3 Params
    // ml2 = ml2 from Pozzi //50;
    L_m3 = 0.8*L_m2; //Length of motor 3
    W_m3 = 0.4*L_m3; //Width of motor 3
    Vol_m3 = W_m3*W_m3*L_m3;
    mm3 = Motor_composition*Vol_m3;

    // Link 3 and Motor 4 Params
    // ml3 = ml3 from Pozzi // 50;
    L_m4 = 0.8*L_m3; //Length of motor 4
    W_m4 = 4*0.4*L_m4; //Width of motor 4
    B_m4 = 0.4*L_m4; //Breadth of motor 4
    Vol_m4 = B_m4*W_m4*L_m4;
    mm4 = Motor_composition*Vol_m4;;

end //...if (Heavy_motors == 1) then



d2 = d1; // makes the proximal link the same distance from the midline,
// equivalent to Pozzi; d3 and d4 do not change they are specific to
// this serial model
d4 = L_m3/2;
rlo_2 = r_p(6); //Takes pozzi proximal link outer radius
rli_2 = r_p(1); //Takes pozzi proximal link inner radius
sr_p(1) = r_p(1);
sr_p(2) = r_p(6);
sr_pm = L_m2;


d3 = 0.5*L_m4;
rlo_3 = r_d(5);//Take pozzi distal link outer radius
rli_3 = r_d(1); //Takes pozzi distal link inner radius
sr_d = [r_d(1); r_d(2); r_d(3); r_d(4)];
sr_dm = [r_dm(1); r_dm(2); r_dm(3)];


tab_space = ascii(9);// + ascii(9); //ascii(9) is the horizontal tab!
if ~exists('disp_dyn_vars') then
    disp_dyn_vars = 0;
end //...if ~exists('disp_dyn_vars') then


if (disp_dyn_vars == 1) then //Prox link vars
    disp('SERIAL - Dynamic Params...')
    disp('Motor composition density = ' + string(Motor_composition) );
    disp('Motor2: L_M2 = ' + string(L_m2) + tab_space + 'W_M2 = ' + ...
            string(W_m2) + tab_space + 'Mass mm2 = ' + string(mm2));
    disp('Motor3: L_M3 = ' + string(L_m3) + tab_space + 'W_M2 = ' + ...
```

```
            string(W_m3) + tab_space + 'Mass mm3 = ' + string(mm3));
    disp('Motors4to6: L_M3 = ' + string(L_m4) + tab_space + 'W_M4 = ' +...
            string(W_m4) + tab_space + 'B_M4 = ' + string(B_m4) + ...
            tab_space + 'Mass mm4 = ' + string(mm4));


    disp('d2 = ' + string(d2) + tab_space + 'd3 = ' + string(d3) + ...
            tab_space + 'd4 = ' + string(d4));
    //disp('a2 = a1 = ' + string(a2));
    disp('Prox link Mass = ' + string(ml2) + tab_space + ...
            'Prox link Ro = ' + string(rlo_2) + tab_space + ...
            'Prox link Ri = ' + string(rli_2));
    disp('Dst link Mass = ' + string(ml3) + tab_space + ...
            'Dist link Ro = ' + string(rlo_3) + tab_space + ...
            'Dist link Ri = ' + string(rli_3));

    disp('Wrist torque transfer link: 1 Mass = ' + string(r_dm(1)) +...
            tab_space + 'Link Ro = ' + string(r_d(2)) + tab_space + ...
            'Link Ri = ' + string(r_d(1)));

    disp('Wrist torque transfer link: 2 Mass = ' + string(r_dm(2)) + ...
            tab_space + 'Link Ro = ' + string(r_d(3)) + tab_space + ...
            'Link Ri = ' + string(r_d(2)));

    disp('Wrist torque transfer link: 3 Mass = ' + string(r_dm(3)) + ...
            tab_space + 'Link Ro = ' + string(r_d(4)) + tab_space + ...
            'Link Ri = ' + string(r_d(3)));

end //... if (disp_dyn_vars == 1) then //Prox link vars



// ************************************************************
// START I_L1Complete
I_tensor = list(0);
PCom_Vec = list(0);
Mass = [];
Mass(1) = mm2 + ml1;
dx = 0;
dy = mm2*d1/Mass(1);
dz = 0;

PCom_Vec(1) = [0;0;0] + [0; -dy; 0];


Ixx = 3*rl1^2 + hl1^2;
Iyy = Iyy;
Izz = 6*rl1^2;
Ibg1 = ml1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = PCom_Vec(1) - [0;0;0];
if ~exists('Eye_3x3') then
    Eye_3x3 = [1 0 0; 0 1 0; 0 0 1];
end //...if ~exists('Eye_3x3') then
Ibg1_at_COM = RotTrans_IT(Ibg1, ml1, drr_vec, Eye_3x3)


Ixx = (W_m2^2 + L_m2^2);
Iyy = 2*W_m2^2;
Izz = Ixx;
I_motor2 = mm2/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [0;-d1;0] - PCom_Vec(1);
I_motor2_at_COM = RotTrans_IT(I_motor2, mm2, drr_vec, Eye_3x3)


I_L1Complete = Ibg1_at_COM + I_motor2_at_COM;
I_tensor(1) = I_L1Complete;
// END I_L1Complete
// ************************************************************




// ************************************************************
```

```
// START I_L2Complete
//*****************************
//START TEST: Remove this code!!!
//mm3 = 0;
//END TEST: Remove this code!!!!
//*****************************
Mass(2) = mm3 + ml2;
dx = 0.5*mm3*L1_Meters/Mass(2);
dy = 0;
dz = mm3*(d2+d4)/Mass(2);
Half_L1 = 0.5*L1_Meters;
PCom_Vec(2) = [Half_L1; 0; -d2] + [dx; 0; dz];


//Tube
Ixx = 6*(rlo_2^2 + rli_2^2);
Iyy = 3*(rlo_2^2 + rli_2^2) + L1_Meters^2;
Izz = Iyy;
I_pls = ml2/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = PCom_Vec(2) - [Half_L1; 0; -d2];
I_pls_at_COM = RotTrans_IT(I_pls, ml2, drr_vec, Eye_3x3);

//Motor
Ixx = (W_m3^2 + L_m3^2);
Iyy = Ixx;
Izz = 2*W_m3^2;
I_motor3 = mm3/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [L1_Meters; 0; d4] - PCom_Vec(2);
I_motor3_at_COM = RotTrans_IT(I_motor3, mm3, drr_vec, Eye_3x3);


I_L2Complete = I_pls_at_COM + I_motor3_at_COM;
I_tensor(2) = I_L2Complete;
// END I_L2Complete
// ************************************************************
```

```
// ************************************************************
// START I_L3Complete
//*****************************
//START TEST: Remove this code!!!
//mm4 = 0;
//ml3 = 0;
//END TEST: Remove this code!!!!
//*****************************
Mass(3) = mm4 + ml3;
if (Mass(3) == 0) then
    dx = 0;
else //if (Mass(3) == 0) then
    dx = mm4*(0.5*L2_Meters + d3)/Mass(3);
end  //if (Mass(3) == 0) then
dy = 0;
dz = 0;
Half_L2 = 0.5*L2_Meters;
PCom_Vec(3) = [Half_L2; 0; 0] + [-dx; 0; 0];


Ixx = 6*(rlo_3^2 + rli_3^2);
Iyy = 3*(rlo_2^2 + rli_3^2) + L2_Meters^2;
Izz = Iyy;
I_dls = ml3/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = PCom_Vec(3) - [Half_L2; 0; 0];
I_dls_at_COM = RotTrans_IT(I_dls, ml3, drr_vec, Eye_3x3);


Ixx = (W_m4^2 + B_m4^2);
Iyy = (B_m4^2 + L_m4^2);
Izz = (W_m4^2 + L_m4^2);
I_motor4 = mm4/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [-d3; 0; 0] - PCom_Vec(3);
I_motor4_at_COM = RotTrans_IT(I_motor4, mm4, drr_vec, Eye_3x3);
```

```
I_L3Complete = I_dls_at_COM + I_motor4_at_COM;
I_tensor(3) = I_L3Complete;


//Inertia tensors for torque transfer tubes
//sr_d = [r_d(1); r_d(2); r_d(3); r_d(4)];
//sr_dm = [r_dm(1); r_dm(2); r_dm(3)];
TTI_Tensor = list(0);
for ttSer_tmp_i = 1 : 3
    Ixx = 6*(sr_d(ttSer_tmp_i+1)^2 + sr_d(ttSer_tmp_i)^2);
    Iyy = 3*(sr_d(ttSer_tmp_i+1)^2 + sr_d(ttSer_tmp_i)^2) + L2_Meters^2;
    Izz = Iyy;
    TTI_Tensor(ttSer_tmp_i) = sr_dm(ttSer_tmp_i)/12*...
                                [Ixx 0 0; 0 Iyy 0; 0 0 Izz];
end


delta_ml3 = (2*mm4*d3)/L2_Meters;
// END I_L3Complete
// *************************************************************




// *************************************************************
// START I_L4,5 Complete
Mass(4) = 0;
Mass(5) = Mass(4);
PCom_Vec(4) = [0;0;0];
PCom_Vec(5) = PCom_Vec(4);
I_L4Complete = [0 0 0; 0 0 0; 0 0 0];
I_L5Complete = I_L4Complete;
I_tensor(4) = I_L4Complete;
I_tensor(5) = I_L5Complete;
// END I_L4,5 Complete
// *************************************************************




// *************************************************************
// START I_WRIST
//Wrist Params
if ~exists('m_wrist') then
     rw = 0.25*L3_Meters;
     m_wrist = %pi*(rw^2)*L3_Meters*steel_density;
end
//*******************************
//START TEST: Remove this code!!!
//m_wrist = 0;
//END TEST: Remove this code!!!!
//*******************************
Mass(6) = m_wrist;

// In the kinematic solution we coded the 6th axis coordinate frame lies
// at the tool point. Hence to get its centre of mass we must travel
// 0.5*L3_Meters in the negative Z6 direction!!!
PCom_Vec(6) = [0; 0; -0.5*L3_Meters];
Ixx = 3*rw^2 + L3_Meters^2;
Iyy = Ixx
Izz = 6*rw^2;
I_L6Complete = Mass(6)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
I_tensor(6) = I_L6Complete;
// END I_WRIST
// *************************************************************




// Store Dynamic model parameters
Dyn_Serial = tlist(["Dyn_Model", "I_Tensor", "Mass", "P_COM", ...
            "L2_components", "L3_components"], [], [], [], [], []);
L2_parts = tlist(["Serial_L2_Parts", "Tube_Radii", "Tube_Masses", ...
```

```
            "Motor3_mass"], [], [], []);
L3_parts = tlist(["Serial_L3_Parts", "Tube_Radii", "Tube_Masses", ...
            "ttI_Tensor", "Motor4_mass"], [], [], [], []);

L2_parts.Tube_Radii = sr_p;
L2_parts.Tube_Masses = sr_pm;
L2_parts.Motor3_mass = mm3;

L3_parts.Tube_Radii = sr_d;
L3_parts.Tube_Masses = sr_dm;
L3_parts.ttI_Tensor = TTI_Tensor
L3_parts.Motor4_mass = mm4;

Dyn_Serial.I_Tensor = I_tensor;
Dyn_Serial.Mass = Mass;
Dyn_Serial.P_COM = PCom_Vec;
Dyn_Serial.L2_components = L2_parts;
Dyn_Serial.L3_components = L3_parts;
```

4.10 **File name:** DynMod_Design2.sce

Purpose: Finds the mass, inertia tensors and locates the centre of mass for each link of the second design robot model.

```
//DynMod_Design2
if ~exists('Dyn_Pozzi') then
    Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";
    DynMod = Dyn_Scr_Path + "DynMod_Pozzi.sce";
    exec(DynMod, -1);
end

// ******************************************
// LINK 1
// CAREFUL: If you change anything here you need to recalculate and
// select the correct cube root!
w1 = 2*rl1; //w1 is known set to Scale_Factor*L1_Meters;
//Need to calculate a1, b1 and c
//b1 = 2*a1
//c = b1 = 2*a1
// Vol_Vbar = a1*b1*l1 = 2*a1^2*l1
// Vol_Hbar = (w1-a1)*b1*c = 4*a1^2*(w1-a1) = 4*a1^2*w1 - 4*a1^3
// R_tt = 0.25*a1
// Tt_vol = %pi*(0.25*a1)^2*l1 = %pi*(0.25^2)*a1^2*l1
// Mass/density = Total_Volume = 2*Vol_Vbar + Vol_Hbar + 4*Tt_vol
// 2*Vol_Vbar = 2*2*a1^2*l1 = 4*a1^2*l1
// 4*Tt_vol = 4*%pi*(0.25^2)*a1^2*l1 = %pi*(0.25)*a1^2*l1
// Total_Volume = 4*a1^2*l1 + 4*a1^2*w1 - 4*a1^3 + %pi*(0.25)*a1^2*l1
//              = -4*a1^3 + (4*l1 + 4*w1 + %pi*(0.25)*l1)*a1^2 + 0*a1
// 0 = -4*a1^3 + (4*l1 + 4*w1 + %pi*(0.25)*l1)*a1^2 + 0*a1 - Total_Volume
// 0 = -4*a1^3 + (4*L1_Meters + 4*w1 + %pi*(0.25)*L1_Meters)*a1^2 +...
//                                                  0*a1 - Total_Volume

PL_Tot_Vol = ml2/steel_density;
AA = -4;
BB = (4*L1_Meters + 4*w1 + %pi*(0.25)*L1_Meters);
CC = 0
DD = -PL_Tot_Vol;

Cube_Rts = Solve_Cubic(AA, BB, CC, DD);
a1 = Cube_Rts(3);
b1 = 2*a1;
c1 = 2*a1;
R_tt = 0.25*a1;

Vol_Vbar1 = a1*b1*L1_Meters;
Vol_Hbar1 = 4*a1^2*w1 - 4*a1^3;
PH_lnk_mass = (2*Vol_Vbar1 + Vol_Hbar1)*steel_density;
Vbar1_mass = Vol_Vbar1*steel_density;
Hbar1_mass = Vol_Hbar1*steel_density;
```

```
TT_mass1 = (%pi*R_tt^2*L1_Meters)*steel_density;


// PH_lnk_mass + 4*TT_mass1



// ********************************************************************
// LINK 2
TT_mass2 = (%pi*R_tt^2*L2_Meters)*steel_density;
DH_lnk_mass = ml3 - 3*TT_mass2;
DL_Vol = DH_lnk_mass/steel_density;

// w2+a2 = w1-a1
// w2 - a2 = w1-a1 - 2*a2;
// b2 = 2*a2;
// c2 = 2*a2;
// Vol_Vbar2 = a2*b2*l2 = 2*a2^2*l2
// Vol_Hbar2 = (w2-a2)*b2*c2 = 4*a2^2*(w1-a1 - 2*a2) = ...
//                                      4*a2^2*(w1-a1) - 8*a2^3
// DL_Vol = 2*Vol_Vbar2 + Vol_Hbar2
//        = 4*a2^2*l2 + 4*a2^2*(w1-a1) - 8*a2^3
//        = - 8*a2^3 + (4*l2 + 4*(w1-a1))*a2^2 + 0*a2
// 0 = - 8*a2^3 + (4*l2 + 4*(w1-a1))*a2^2 + 0*a2 - DL_Vol

AA = -8;
BB = (4*L2_Meters + 4*(w1-a1));
CC = 0
DD = -DL_Vol;

Cube_Rts = Solve_Cubic(AA, BB, CC, DD);
a2 = Cube_Rts(3);
b2 = 2*a2;
c2 = 2*a2;
w2 = w1-a1-a2;

Vol_Vbar2 = a2*b2*L2_Meters;
Vol_Hbar2 = 4*a2^2*(w1-a1) - 8*a2^3;
DH_lnk_mass = (2*Vol_Vbar2 + Vol_Hbar2)*steel_density;
Vbar2_mass = Vol_Vbar2*steel_density;
Hbar2_mass = Vol_Hbar2*steel_density;


    if ~exists('disp_dyn_vars') then
        disp_dyn_vars = 0;
    end //...if ~exists('disp_dyn_vars') then


    if (disp_dyn_vars == 1) then //Prox link vars
        disp('DESIGN_2 - Dynamic Params...')
        disp('w1 = 2*rl1 = ' + string(w1));
        disp('a1 = ' + string(a1) + tab_space + 'b1 = 2*a1 = ' + ...
             string(b1) + tab_space + 'c1 = 2*a1 = ' + string(c1));
        disp('R_tt = 0.25*a1 = ' + string(R_tt));
        disp('w2 = w1-a1-a2 = ' + string(w2));
        disp('a2 = ' + string(a2) + tab_space + 'b2 = 2*a2 = ' + ...
             string(b2) + tab_space + 'c2 = 2*a2 = ' + string(c2));

        disp('Proximal torque transfer link: Mass = ' + ...
             string(TT_mass1) + tab_space + 'Link Ro = ' + string(R_tt));
        disp('Distal torque transfer link: Mass = ' + ...
             string(TT_mass2) + tab_space + 'Link Ro = ' + string(R_tt));

        disp('PH_lnk_mass = ml2 - 4*TT_mass1 = ' + string(ml2-4*TT_mass1));
        disp('PH_lnk_mass = (2*Vol_Vbar1 + Vol_Hbar1)*steel_density = ' + ...
             string((2*Vol_Vbar1 + Vol_Hbar1)*steel_density));

        disp('DH_lnk_mass = ml3 - 3*TT_mass2 = ' + string(ml3-3*TT_mass2));
        disp('DH_lnk_mass = (2*Vol_Vbar2 + Vol_Hbar2)*steel_density = ' + ...
             string((2*Vol_Vbar2 + Vol_Hbar2)*steel_density));
    end //... if (disp_dyn_vars == 1) then //Prox link vars
```

```
// ************************************************************
// START I_L1Complete
I_tensor = list(0);
PCom_Vec = list(0);
Mass = [];
Mass(1) = ml1;
PCom_Vec(1) = [0;0;0];

Ixx = 3*rl1^2 + hl1^2;
Iyy = Ixx;
Izz = 6*rl1^2;
Ibg1 = ml1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

I_L1Complete = Ibg1;
I_tensor(1) = I_L1Complete;
// END I_L1Complete
// ************************************************************




// ************************************************************
// START I_L2Complete
Mass(2) = ml2;
Half_L1 = 0.5*L1_Meters;
PCom_Vec(2) = [Half_L1; 0; 0]; //Machine is symmetric


Ixx = (a1^2 + b1^2);
Iyy = (a1^2 + L1_Meters^2);
Izz = (b1^2 + L1_Meters^2);
I_vbar1 = Vbar1_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [Half_L1;0;-w1/2] - PCom_Vec(2);
I_vbar1a_at_COM = RotTrans_IT(I_vbar1, Vbar1_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L1;0;w1/2] - PCom_Vec(2);
I_vbar1b_at_COM = RotTrans_IT(I_vbar1, Vbar1_mass, drr_vec, Eye_3x3);


Ixx = ((w1-a1)^2 + c1^2);
Iyy = ((w1-a1)^2 + b1^2);
Izz = (b1^2 + c1^2);
I_hbar1 = Hbar1_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
I_hbar1_at_COM = I_hbar1


I_phl_at_COM = I_vbar1a_at_COM + I_vbar1b_at_COM +I_hbar1_at_COM;


Ixx = 6*R_tt^2;;
Iyy = 3*R_tt^2 + L1_Meters^2;;
Izz = Iyy;
I_ttbar1 = TT_mass1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];


tt_dz = 1.5*(w1/5);
drr_vec = [Half_L1;0;tt_dz] - PCom_Vec(2);
I_ttbar1a_at_COM = RotTrans_IT(I_ttbar1, TT_mass1, drr_vec, Eye_3x3);
drr_vec = [Half_L1;0;-tt_dz] - PCom_Vec(2);
I_ttbar1d_at_COM = RotTrans_IT(I_ttbar1, TT_mass1, drr_vec, Eye_3x3);



tt_dz = 0.5*(w1/5);
drr_vec = [Half_L1;0;tt_dz] - PCom_Vec(2);
I_ttbar1b_at_COM = RotTrans_IT(I_ttbar1, TT_mass1, drr_vec, Eye_3x3);
drr_vec = [Half_L1;0;-tt_dz] - PCom_Vec(2);
I_ttbar1c_at_COM = RotTrans_IT(I_ttbar1, TT_mass1, drr_vec, Eye_3x3);


I_L2Complete = I_phl_at_COM + I_ttbar1a_at_COM + I_ttbar1d_at_COM + ...
               I_ttbar1b_at_COM + I_ttbar1c_at_COM;
I_tensor(2) = I_L2Complete;

// END I_L2Complete
// ************************************************************
```

```
// ************************************************************
// START I_L3Complete
Mass(3) = ml3;
Half_L2 = 0.5*L2_Meters;
PCom_Vec(3) = [Half_L2; 0; 0]; //Machine is symmetric

Ixx = (a2^2 + b2^2);
Iyy = (a2^2 + L2_Meters^2);
Izz = (b2^2 + L2_Meters^2);
I_vbar2 = Vbar2_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [Half_L2;0;-w2/2] - PCom_Vec(3);
I_vbar2a_at_COM = RotTrans_IT(I_vbar2, Vbar2_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L2;0;w2/2] - PCom_Vec(3);
I_vbar2b_at_COM = RotTrans_IT(I_vbar2, Vbar2_mass, drr_vec, Eye_3x3);

Ixx = ((w2-a2)^2 + b2^2);
Iyy = ((w2-a2)^2 + c2^2);
Izz = (b2^2 + c2^2);
I_hbar2 = Hbar2_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
I_hbar2_at_COM = I_hbar2


I_dhl_at_COM = I_vbar2a_at_COM + I_vbar2b_at_COM + I_hbar2_at_COM;


Ixx = 6*R_tt^2;
Iyy = 3*R_tt^2 + L2_Meters^2;;
Izz = Iyy;
I_ttbar2 = TT_mass2/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
tt_dz = w2/4;
drr_vec = [Half_L2;0;tt_dz] - PCom_Vec(3);
I_ttbar2a_at_COM = RotTrans_IT(I_ttbar2, TT_mass2, drr_vec, Eye_3x3);
drr_vec = [Half_L2;0;-tt_dz] - PCom_Vec(3);
I_ttbar2c_at_COM = RotTrans_IT(I_ttbar2, TT_mass2, drr_vec, Eye_3x3);



I_L3Complete = I_dhl_at_COM + I_ttbar2a_at_COM + I_ttbar2 + ...
               I_ttbar2c_at_COM;
I_tensor(3) = I_L3Complete;
// END I_L3Complete
// ************************************************************
```

```
// ************************************************************
// START I_L4,5 Complete
Mass(4) = 0;
Mass(5) = Mass(4);
PCom_Vec(4) = [0;0;0];
PCom_Vec(5) = PCom_Vec(4);
I_L4Complete = [0 0 0; 0 0 0; 0 0 0];
I_L5Complete = I_L4Complete;
I_tensor(4) = I_L4Complete;
I_tensor(5) = I_L5Complete;
// END I_L4,5 Complete
// ************************************************************
```

```
// ************************************************************
// START I_WRIST
//Wrist Params

if ~exists('L3_Meters') then
    L3_len =  181.2938; //181.2937611;
    L3_Meters = L3_len/1000;
end
rw = 0.25*L3_Meters;

m_wrist = %pi*(rw^2)*L3_Meters*steel_density; //10
Mass(6) = m_wrist


// In the kinematic solution we coded the 6th axis coordinate frame
// lies at the tool point. Hence to get its centre of mass we must
// travel 0.5*L3_Meters in the negative Z6 direction!!!
PCom_Vec(6) = [0; 0; -0.5*L3_Meters];
Ixx = 3*rw^2 + L3_Meters^2;
Iyy = Ixx
Izz = 6*rw^2;
I_L6Complete = Mass(6)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
I_tensor(6) = I_L6Complete;
// END I_WRIST
// ************************************************************




//// Store Dynamic model parameters
Dyn_Design2 = tlist(["Dyn_Model", "I_Tensor", "Mass", "P_COM", ...
                     "L2_components", "L3_components"], [], [], [], [], []);
L2_parts = tlist(["Desn2_L2_Parts", "Tube_Radii", "Tube_Masses", ...
                  "ttI_Tensor"], [], [], []);
L3_parts = tlist(["Desn2_L3_Parts", "Tube_Radii", "Tube_Masses", ...
                  "ttI_Tensor"], [], [], []);

L2_parts.Tube_Radii = R_tt;
L2_parts.Tube_Masses = TT_mass1;
L2_parts.ttI_Tensor = I_ttbar1;

L3_parts.Tube_Radii = R_tt;
L3_parts.Tube_Masses = TT_mass2;
L3_parts.ttI_Tensor = I_ttbar2;


Dyn_Design2.I_Tensor = I_tensor;
Dyn_Design2.Mass = Mass;
Dyn_Design2.P_COM = PCom_Vec;
Dyn_Design2.L2_components = L2_parts;
Dyn_Design2.L3_components = L3_parts;
```

4.11 **File name:** DynMod_Design1.sce

Purpose: Finds the mass, inertia tensors and locates the centre of mass for each link of the first design robot model.

```
//DynMod_Design1

if ~exists('Dyn_Pozzi') then
    Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";

    DynMod = Dyn_Scr_Path + "DynMod_Pozzi.sce";
    exec(DynMod, -1);

    // Also call second design as we will use the mass of the torque
    // transfer rod to determine the dimensions of the torque transfer
    // links here!
    DynMod = Dyn_Scr_Path + "DynMod_Design2.sce";
```

```
    exec(DynMod, -1);
    //disp_dyn_vars = 1;
end //...if ~exists('Dyn_Pozzi') then


if ~exists('Dyn_Design1') then
    // ******************************************
    // LINK 1
    // CAREFUL: If you change anything here you need to recalculate and
    // select the correct
    // cube root!
    w1 = 2*rl1; //rl1 is known set to Scale_Factor*L1_Meters;
    //Need to calculate a1, a2, b2 and c1

    // Will calculate the width and breath of the torque transfer bars
    // first. The area must be the same as the area for the torque
    // transfer tubes of the second design.
    // width = p; breath = q = 2*p;
    // Area = 2*p^2 = %pi*R_tt^2
    // p = (%pi/2)^0.5*R_tt
    p = (%pi/2)^0.5*R_tt;
    q = 2*p; //Mass of rectangular torque transfer bar TT_mass1
    PH_lnk_mass = ml2 - 4*TT_mass1;
    PL_Vol = (ml2 - 4*TT_mass1)/steel_density;
    // The proximal link is made up of 2 identical cross pieces and 2
    // horizontal pieces.
    // b1 = 0.5*w1
    // b2 = 2*a1
    // c1 = 2*a1
    // a2 = a1
    // 0.5*PL_Vol = a1*b2*l1 + b1*c1*a1 + (w1-a1)*c1*a2 - b2*c1*a1
    //     = 2*a1^2*l1 + 0.5*w1*2*a1*a1 + (w1-a1)*2*a1*a2 - 2*a1*2*a1*a1
    //     = 2*a1^2*l1 + w1*a1^2 + 2*w1*a1^2 - 2*a1^3 - 4*a1^3
    //     = -6*a1^3 +(2*l1 + 3*w1)*a1^2
    // 0 = -6*a1^3 +(2*l1 + 3*w1)*a1^2 + 0*a1 - 0.5*PL_Vol

    AA = -6;
    BB = 2*L1_Meters + 3*w1;
    CC = 0
    DD = -0.5*PL_Vol;

    Cube_Rts = Solve_Cubic(AA, BB, CC, DD);
    a1 = Cube_Rts(3);
    b1 = 0.5*w1;
    b2 = 2*a1;
    c1 = 2*a1;
    a2 = a1;


    Vol_Vbar1 = a1*b2*L1_Meters;
    Vol_Hbar1 = c1*b1*a1;
    Vol_Hbar2 = c1*a2*(w1-a1);
    Vbar1_mass = Vol_Vbar1*steel_density;
    Hbar1_mass = Vol_Hbar1*steel_density;
    Hbar2_mass = Vol_Hbar2*steel_density;
    Common1_mass = b2*c1*a1*steel_density;
    // PH_lnk_mass = 2*(Vbar1_mass + Hbar1_mass + Hbar2_mass - Common_mass)
    // 2*(Vbar1_mass + Hbar1_mass + Hbar2_mass - Common_mass) + 4*TT_mass1

    tab_space = ascii(9);// + ascii(9); //ascii(9) is the horizontal tab!
    if ~exists('disp_dyn_vars') then
        disp_dyn_vars = 0;
    end //...if ~exists('disp_dyn_vars') then

    if (disp_dyn_vars == 1) then //Prox link vars
        disp('DESIGN 1 - Dynamic Params...')
        disp('Cube roots = [' + strcat(string(Cube_Rts), ',') + ' ]');
        disp('a1 = Cube roots(3) = ' + string(a1) + tab_space + ...
                    'a2 = a1 = ' + string(a2));
        //disp('a2 = a1 = ' + string(a2));
        disp('w1 = ' + string(w1));
        disp('b1 = 0.5*w1 = ' + string(b1) + tab_space + ...
                    'b2 = 2*a1 = ' + string(b2));
        //disp('b1 = 0.5*w1 = ' + string(b1));
        //disp('b2 = 2*a1 = ' + string(b2));
        disp('c1 = 2*a1 = ' + string(c1));
```

```
    disp('p = (%pi/2)^0.5*R_tt = ' + string(p));
    disp('q = 2*p = ' + string(q));

    disp('PH_lnk_mass = 2*(Vbar1_mass + Hbar1_mass +' + ...
         ' Hbar2_mass - Common1_mass) = ' + ...
          string(2*(Vbar1_mass + Hbar1_mass + Hbar2_mass - ...
          Common1_mass)));

    disp('PH_lnk_mass = ml2 - 4*TT_mass1 = ' + string(PH_lnk_mass));
end //... if (disp_dyn_vars == 1) then //Prox link vars



// ****************************************************************
// LINK 2
//disp('TT_mass2 = ' + string(TT_mass2));
TT_mass2 = p*q*L2_Meters*steel_density;
//disp('TT_mass2 = ' + string(TT_mass2));
DH_lnk_mass = ml3 - 3*TT_mass2;
DL_Vol = DH_lnk_mass/steel_density;

// w2+a3 = w1-a1
// w2-a3 = w1-a1 - 2*a3
// w2 = w1-a1 - a3
// b4 = 0.5*w2 = 0.5*(w1-a1 - a3) = 0.5*(w1-a1) - 0.5*a3
// b3 = 2*a3
// c2 = 2*a3
// a4 = a3
// Vol_Vbar2 = a3*b3*l2 = 2*a3^2*l2
// Vol_Hbar3 = b4*c2*a3 =  2*a3^2*(0.5*(w1-a1) - 0.5*a3) = ...
//                         (w1-a1)*a3^2 - a3^3
// Vol_Hbar4 = (w2-a3)*c2*a4 = 2*a3^2*(w1-a1 - 2*a3) = ...
//                         2*(w1-a1)*a3^2 - 4*a3^3
// Common_vol = c2*b3*a3 = 4*a3^3
// 0.5*DL_Vol = Vol_Vbar2 + Vol_Hbar3 + Vol_Hbar4 - Common_vol
//            = 2*a3^2*l2 + (w1-a1)*a3^2 - a3^3 + 2*(w1-a1)*a3^2 ...
//              - 4*a3^3 - 4*a3^3
//            = -9*a3^3 +(2*l2 + 3*(w1-a1))*a^2 + 0*a1
// 0 = -9*a3^3 +(2*l2 + 3*(w1-a1))*a^2 + 0*a1 - 0.5*DL_Vol

AA = -9;
BB = (2*L2_Meters + 3*(w1-a1));
CC = 0
DD = -0.5*DL_Vol;
Cube_Rts = Solve_Cubic(AA, BB, CC, DD);
a3 = Cube_Rts(3);
b3 = 2*a3;
c2 = 2*a3;
w2 = w1-a1-a3;
b4 = 0.5*w2;
a4 = a3

Vol_Vbar2 = a3*b3*L2_Meters;
Vol_Hbar3 = b4*c2*a3;
Vol_Hbar4 = (w2-a3)*c2*a4;
Vbar2_mass = Vol_Vbar2*steel_density;
Hbar3_mass = Vol_Hbar3*steel_density;
Hbar4_mass = Vol_Hbar4*steel_density;
Common2_mass = c2*b3*a3*steel_density;
// DH_lnk_mass = 2*(Vbar2_mass + Hbar3_mass + Hbar4_mass - ...
//               Common_mass)

if (disp_dyn_vars == 1) then //Dist link vars
    disp(ascii(10)); //ascii(10) is the newline character
    disp('Cube roots = [' + strcat(string(Cube_Rts), ',') + ' ]');
    disp('a3 = Cube roots(3) = ' + string(a3) + tab_space + ...
                    'a4 = a3 = ' + string(a4));
    //disp('a3 = Cube roots(3) = ' + string(a3));
    //disp('a4 = a3 = ' + string(a4));
    disp('w2 = w1-a1-a3 = ' + string(w2));
    disp('b3 = 2*a3 = ' + string(b3) + tab_space + ...
                    'b4 = 0.5*w2 = ' + string(b4));
    //disp('b3 = 2*a3 = ' + string(b3));
    //disp('b4 = 0.5*w2 = ' + string(b4));
    disp('c2 = 2*a3 = ' + string(c2));
    disp('DH_lnk_mass = 2*(Vbar2_mass + Hbar3_mass + '+ ...
                    'Hbar4_mass - Common2_mass) = ' + ...
```

```
        string(2*(Vbar2_mass + Hbar3_mass + Hbar4_mass - ...
                  Common2_mass)));
    disp('DH_lnk_mass = ml3 - 3*TT_mass2 = ' + string(DH_lnk_mass));
end // ... if (disp_dyn_vars == 1) then //Dist link vars



// ************************************************************
// START I_L1Complete
I_tensor = list(0);
PCom_Vec = list(0);
Mass = [];
Mass(1) = ml1;
PCom_Vec(1) = [0;0;0];

Ixx = 3*rl1^2 + hl1^2;
Iyy = Ixx;
Izz = 6*rl1^2;
Ibg1 = ml1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

I_L1Complete = Ibg1;
I_tensor(1) = I_L1Complete;
// END I_L1Complete
// ************************************************************



// ************************************************************
// START I_phl_at_COM - PHL inertia tensor initialization!
Mass(2) = ml2;
Half_L1 = 0.5*L1_Meters;
PCom_Vec(2) = [Half_L1; 0; 0]; //Machine is symmetric

Ixx = (a1^2 + b2^2);
Iyy = (a1^2 + L1_Meters^2);
Izz = (b2^2 + L1_Meters^2);
I_vbar1 = Vbar1_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Ixx = (a1^2 + b1^2);
Iyy = (a1^2 + c1^2);
Izz = (b1^2 + c1^2);
I_hbar1 = Hbar1_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Ixx = (a1^2 + b2^2);
Iyy = (a1^2 + c1^2);
Izz = (b2^2 + c1^2);
I_common1 = Common1_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Pplus_mass = Vbar1_mass + Hbar1_mass - Common1_mass
I_plus1 = I_vbar1 + I_hbar1 - I_common1;
drr_vec = [Half_L1; 0; w1/2] - PCom_Vec(2);
I_plus1a_at_COM = RotTrans_IT(I_plus1, Pplus_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L1; 0; -w1/2] - PCom_Vec(2);
I_plus1b_at_COM = RotTrans_IT(I_plus1, Pplus_mass, drr_vec, Eye_3x3);

hbar2_dy = (b1 - a2)/2;
Ixx = (a2^2 + (w1-a1)^2);
Iyy = (c1^2 + (w1-a1)^2);
Izz = (c1^2 + a2^2);
I_hbar2 = Hbar2_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [Half_L1; hbar2_dy; 0] - PCom_Vec(2);
I_hbar2a_at_COM = RotTrans_IT(I_hbar2, Hbar2_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L1; -hbar2_dy; 0] - PCom_Vec(2);
I_hbar2b_at_COM = RotTrans_IT(I_hbar2, Hbar2_mass, drr_vec, Eye_3x3);



I_phl_at_COM = I_plus1a_at_COM + I_plus1b_at_COM + ...
               I_hbar2a_at_COM + I_hbar2b_at_COM;


Ixx = p^2 + q^2;
Iyy = p^2 + L1_Meters^2;
Izz = q^2 + L1_Meters^2;
I_ttb1 = TT_mass1/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
```

```
// END I_phl_at_COM - PHL inertia tensor initialization!
// ************************************************************




// ************************************************************
// START I_dhl_at_COM - DHL inertia tensor initialization!
Mass(3) = ml3;
Half_L2 = 0.5*L2_Meters;
PCom_Vec(3) = [Half_L2; 0; 0]; //Machine is symmetric

Ixx = (a3^2 + b3^2);
Iyy = (a3^2 + L2_Meters^2);
Izz = (b3^2 + L2_Meters^2);
I_vbar2 = Vbar2_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Ixx = (a3^2 + b4^2);
Iyy = (a3^2 + c2^2);
Izz = (b4^2 + c2^2);
I_hbar3 = Hbar3_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Ixx = (a3^2 + b3^2);
Iyy = (a3^2 + c2^2);
Izz = (b3^2 + c2^2);
I_common2 = Common2_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

Dplus_mass = Vbar2_mass + Hbar3_mass - Common2_mass;
I_plus2 = I_vbar2 + I_hbar3 - I_common2;
drr_vec = [Half_L2; 0; w2/2] - PCom_Vec(3);
I_plus2a_at_COM = RotTrans_IT(I_plus2, Dplus_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L2; 0; -w2/2] - PCom_Vec(3);
I_plus2b_at_COM = RotTrans_IT(I_plus2, Dplus_mass, drr_vec, Eye_3x3);


hbar4_dy = (b4 - a4)/2;
Ixx = (a4^2 + (w2-a3)^2);
Iyy = (c2^2 + (w2-a3)^2);
Izz = (c2^2 + a2^2);
I_hbar4 = Hbar4_mass/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
drr_vec = [Half_L2; hbar4_dy; 0] - PCom_Vec(3);
I_hbar4a_at_COM = RotTrans_IT(I_hbar4, Hbar4_mass, drr_vec, Eye_3x3);
drr_vec = [Half_L2; -hbar4_dy; 0] - PCom_Vec(3);
I_hbar4b_at_COM = RotTrans_IT(I_hbar4, Hbar4_mass, drr_vec, Eye_3x3);


I_dhl_at_COM = I_plus2a_at_COM + I_plus2b_at_COM + ...
               I_hbar4a_at_COM + I_hbar4b_at_COM;

Ixx = p^2 + q^2;
Iyy = p^2 + L2_Meters^2;
Izz = q^2 + L2_Meters^2;
I_ttb2 = TT_mass2/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];

// END I_dhl_at_COM - DHL inertia tensor initialization!
// ************************************************************




// ************************************************************
// START I_L4,5 Complete
Mass(4) = 0;
Mass(5) = Mass(4);
PCom_Vec(4) = [0;0;0];
PCom_Vec(5) = PCom_Vec(4);
I_L4Complete = [0 0 0; 0 0 0; 0 0 0];
I_L5Complete = I_L4Complete;
I_tensor(4) = I_L4Complete;
I_tensor(5) = I_L5Complete;
// END I_L4,5 Complete
// ************************************************************
```

```
    // ************************************************************
    // START I_WRIST
    // Wrist Params

    if ~exists('L3_Meters') then
        L3_len =  181.2938; //181.2937611;
        L3_Meters = L3_len/1000;
    end
    rw = 0.25*L3_Meters;

    m_wrist = %pi*(rw^2)*L3_Meters*steel_density; //10
    Mass(6) = m_wrist


    // In the kinematic solution we coded the 6th axis coordinate frame
    // lies at the tool point. Hence to get its centre of mass we must
    // travel 0.5*L3_Meters in the negative Z6 direction!!!
    PCom_Vec(6) = [0; 0; -0.5*L3_Meters];
    Ixx = 3*rw^2 + L3_Meters^2;
    Iyy = Ixx
    Izz = 6*rw^2;
    I_L6Complete = Mass(6)/12*[Ixx 0 0; 0 Iyy 0; 0 0 Izz];
    I_tensor(6) = I_L6Complete;
    // END I_WRIST
    // ************************************************************


    // Store Dynamic model parameters
    Dyn_Design1 = tlist(["Dyn_Model", "I_Tensor", "Mass", "P_COM", ...
                         "L2_components", "L3_components"], [], [], ...
                         [], [], []);
    L2_parts = tlist(["Desn2_L2_Parts", "pq", "TT_Mass", ...
                      "ttI_Tensor"], [], [], []);
    L3_parts = tlist(["Desn2_L3_Parts", "pq", "TT_Mass", ...
                      "ttI_Tensor"], [], [], []);

    // These don't change from kinematic configuration to kinematic
    // configuration
    L2_parts.pq = [p,q];
    L2_parts.TT_Mass = TT_mass1;
    L2_parts.ttI_Tensor = I_ttb1;

    L3_parts.pq = [p,q];
    L3_parts.TT_Mass = TT_mass2;
    L3_parts.ttI_Tensor = I_ttb2;

    Dyn_Design1.L2_components = L2_parts;
    Dyn_Design1.L3_components = L3_parts;
    Dyn_Design1.Mass = Mass;

    // NEED TO SAVE TORQUE TRANSFER LINKS PCOMS TO CALCULATE KINETIC
    // ENERGY LATER...
    TT_params = list(0);
    TT_configs = tlist(["TT_configs", "ttc3", "ttc4", "ttc5", ...
                        "ttc6"], [], [], [], []);
    TT_config = tlist(["TT_config", "ttla_com", "theta1", ...
                        "ttlb_com", "theta2"], [], [], [], []);

    //Remove
    disp('Completed initialization!')  // Remove

end //... if ~exists('Dyn_Design1') then



//*************************************************************
// CHANGED KINEMATIC CONFIGURATION - INERTIA TENSORS FOR LINKS
// 2 AND 3 MUST CHANGE AS WELL!!!!
r_k = 0.04;

d_ttz = (w2 - a3)/5;
d1z =  1.5*d_ttz;
d2z =  0.5*d_ttz;
d3z = -0.5*d_ttz;
d4z = -1.5*d_ttz;
```

```
TT_PCOM_Script = Dyn_Scr_Path + "TT_Lnks_I_PCOM.sce";
tt_i = 3;
tt_angZ = -Animation_Angs(k, tt_i);//+ Animation_Angs(k, 2); //the gear for distal link
// rotates in opposite direction!
dz_tt_pos = d1z;
exec(TT_PCOM_Script, -1);
// Got Prx_TT_COM, Dst_TT_COM, Theta1 and Theta2.
Prx_TT_COM_3 = Prx_TT_COM + dz_tt_com; //[0; 0; d1z];
Dst_TT_COM_3 = Dst_TT_COM + dz_tt_com; //[0; 0; d1z];;
Theta1_3 = Theta1;
Theta2_3 = Theta2;
TT_config.ttla_com = Prx_TT_COM_3;
TT_config.theta1 = Theta1_3;
TT_config.ttlb_com = Dst_TT_COM_3;
TT_config.theta2 = Theta2_3;
TT_configs.ttc3 = TT_config;

AAngs_2p3 = Animation_Angs(k, 2) + Animation_Angs(k, 3);
tt_i = 4;
tt_angZ = Animation_Angs(k, tt_i);// + AAngs_2p3;
dz_tt_pos = d2z;
exec(TT_PCOM_Script, -1);
// Got Prx_TT_COM, Dst_TT_COM, Theta1 and Theta2.
Prx_TT_COM_4 = Prx_TT_COM + dz_tt_com; //[0; 0; d2z];
Dst_TT_COM_4 = Dst_TT_COM + dz_tt_com; //[0; 0; d2z];
Theta1_4 = Theta1;
Theta2_4 = Theta2;
TT_config.ttla_com = Prx_TT_COM_4;
TT_config.theta1 = Theta1_4;
TT_config.ttlb_com = Dst_TT_COM_4;
TT_config.theta2 = Theta2_4;
TT_configs.ttc4 = TT_config;


tt_i = 5;
tt_angZ = Animation_Angs(k, tt_i);// + AAngs_2p3;
dz_tt_pos = d3z;
exec(TT_PCOM_Script, -1);
// Got Prx_TT_COM, Dst_TT_COM, Theta1 and Theta2.
Prx_TT_COM_5 = Prx_TT_COM + dz_tt_com; //[0; 0; d3z];
Dst_TT_COM_5 = Dst_TT_COM + dz_tt_com; //[0; 0; d3z];
Theta1_5 = Theta1;
Theta2_5 = Theta2;
TT_config.ttla_com = Prx_TT_COM_5;
TT_config.theta1 = Theta1_5;
TT_config.ttlb_com = Dst_TT_COM_5;
TT_config.theta2 = Theta2_5;
TT_configs.ttc5 = TT_config;


tt_i = 6;
tt_angZ = Animation_Angs(k, tt_i);// + AAngs_2p3;
dz_tt_pos = d4z;
exec(TT_PCOM_Script, -1);
// Got Prx_TT_COM, Dst_TT_COM, Theta1 and Theta2.
Prx_TT_COM_6 = Prx_TT_COM + dz_tt_com; //[0; 0; d4z];
Dst_TT_COM_6 = Dst_TT_COM + dz_tt_com; //[0; 0; d4z];
Theta1_6 = Theta1;
Theta2_6 = Theta2;
TT_config.ttla_com = Prx_TT_COM_6;
TT_config.theta1 = Theta1_6;
TT_config.ttlb_com = Dst_TT_COM_6;
TT_config.theta2 = Theta2_6;
TT_configs.ttc6 = TT_config;

//************************************************************
TT_params($+1) = TT_configs; //Saves torque transfer link
// cofigurations per pose.

TTm1x4_ova_ml2 = 4*TT_mass1/(ml2);
HLP_COM_dX = TTm1x4_ova_ml2*(Prx_TT_COM_3(1) + Prx_TT_COM_4(1) + ...
             Prx_TT_COM_5(1) + Prx_TT_COM_6(1));

HLP_COM_dY = TTm1x4_ova_ml2*(Prx_TT_COM_3(2) + Prx_TT_COM_4(2) + ...
             Prx_TT_COM_5(2) + Prx_TT_COM_6(2));
```

```
HLP_COM_dZ = TTm1x4_ova_ml2*(Prx_TT_COM_3(3) + Prx_TT_COM_4(3) + ...
             Prx_TT_COM_5(3) + Prx_TT_COM_6(3));

Delta_HLP_COM = [HLP_COM_dX; HLP_COM_dY; HLP_COM_dZ];

PCom_Vec(2) = [0.5*L1_Meters; 0; 0] + Delta_HLP_COM;

TTm2x3_ova_ml3 = 3*TT_mass2/(ml3);
HLD_COM_dX = TTm2x3_ova_ml3*(Dst_TT_COM_4(1) + Dst_TT_COM_5(1) + ...
             Dst_TT_COM_6(1));

HLD_COM_dY = TTm2x3_ova_ml3*(Dst_TT_COM_4(2) + Dst_TT_COM_5(2) + ...
             Dst_TT_COM_6(2));

HLD_COM_dZ = TTm2x3_ova_ml3*(Dst_TT_COM_4(3) + Dst_TT_COM_5(3) + ...
             Dst_TT_COM_6(3));

Delta_HLD_COM = [HLD_COM_dX; HLD_COM_dY; HLD_COM_dZ];

PCom_Vec(3) = [0.5*L2_Meters; 0; 0] + Delta_HLD_COM;


//J = I + m(r^2*Eye3x3 - r_vec*r_vec')
drr_vec = Delta_HLP_COM;
I_phl_at_COM1 = RotTrans_IT(I_phl_at_COM, PH_lnk_mass, drr_vec, Eye_3x3);

drr_vec = Prx_TT_COM_3 - Delta_HLP_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta1_3);
I_ttb1_1COM = RotTrans_IT(I_ttb1, TT_mass1, drr_vec, RZ_Theta);

drr_vec = Prx_TT_COM_4 - Delta_HLP_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta1_4);
I_ttb1_2COM = RotTrans_IT(I_ttb1, TT_mass1, drr_vec, RZ_Theta);

drr_vec = Prx_TT_COM_5 - Delta_HLP_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta1_5);
I_ttb1_3COM = RotTrans_IT(I_ttb1, TT_mass1, drr_vec, RZ_Theta);

drr_vec = Prx_TT_COM_6 - Delta_HLP_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta1_6);
I_ttb1_4COM = RotTrans_IT(I_ttb1, TT_mass1, drr_vec, RZ_Theta);


I_L2Complete = I_phl_at_COM1 + I_ttb1_1COM + I_ttb1_2COM + ...
               I_ttb1_3COM + I_ttb1_4COM;
I_tensor(2) = I_L2Complete;




drr_vec = Delta_HLD_COM;
I_dhl_at_COM1 = RotTrans_IT(I_dhl_at_COM, DH_lnk_mass, drr_vec, Eye_3x3);

drr_vec = Dst_TT_COM_4 - Delta_HLD_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta2_4);
I_ttb2_2COM = RotTrans_IT(I_ttb2, TT_mass2, drr_vec, RZ_Theta);

drr_vec = Dst_TT_COM_5 - Delta_HLD_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta2_5);
I_ttb2_3COM = RotTrans_IT(I_ttb2, TT_mass2, drr_vec, RZ_Theta);

drr_vec = Dst_TT_COM_6 - Delta_HLD_COM;
RZ_Theta = Rotate_XYZ(0, 0, Theta2_6);
I_ttb2_4COM = RotTrans_IT(I_ttb2, TT_mass2, drr_vec, RZ_Theta);

I_L3Complete = I_dhl_at_COM1 + I_ttb2_2COM + I_ttb2_3COM + I_ttb2_4COM;
I_tensor(3) = I_L3Complete;

//*********************************************************************



//// Store Dynamic model parameters
Dyn_Design1.I_Tensor = I_tensor;
Dyn_Design1.P_COM = PCom_Vec;
Dyn_Design1.L2_components = L2_parts;
Dyn_Design1.L3_components = L3_parts;
```

4.12 **File name:** Solve_Cubic.sci

Purpose: Solves for the roots of a cubic equation.

```
// Cubic Equation Solver
// disp("-Debug line 173: Solve_Cubic")
function [Rts, Result, Msg] = Solve_Cubic(A, B, C, D)

    This_Fun_Name = "Solve_Cubic:";
    Err_Msg = "";
    Result = 0;
    Rts = [];


    if (A == 0) then
        // Cannot solve, division by 0!
        Err_Msg = Err_Msg + "This is not a cubic equation, A is 0!"
        disp(Err_Msg);
        Result = 0;
    else //...if (A == 0) then
        a2 = B/A;
        a1 = C/A;
        a0 = D/A;
        a3 = A/A;
        // z^3 + a2.z^2 + a1.z + a0 = 0
        cube_root = 1/3;
        one_third = cube_root;
        //Solution from WOLFRAM
        Q = (3*a1 - a2^2)/9;
        R = (9*a2*a1 - 27*a0 - 2*a2^3)/54;
        D = Q^3 + R^2;
        S = (R + (D)^0.5)^cube_root
        T = (R - (D)^0.5)^cube_root

        NEG_one_third_a2 = -one_third*a2;
        S_PLUS_T = S+T;
        Half_S_PLUS_T = 0.5*S_PLUS_T;
        Half_i_S_MINUS_T = 0.5*%i*(3^0.5)*(S-T);

        z1 = NEG_one_third_a2 + S_PLUS_T;
        z2 = NEG_one_third_a2 - Half_S_PLUS_T + Half_i_S_MINUS_T;
        z3 = NEG_one_third_a2 - Half_S_PLUS_T - Half_i_S_MINUS_T;

        Rts = [z1, z2, z3];

    end //...if (A == 0) then

    Msg = This_Fun_Name + Err_Msg;
endfunction
```

4.13 **File name:** RotTrans_IT.sci

Purpose: Rotates and translates the inertia tensor. Rotation must be done through a similarity transform.

```
//Rotate and translate inertia tensor....
// disp("-Debug line 173: RotTrans_IT")
function [Result, Msg] = RotTrans_IT(I, mass, Drr_vec, Rot)
    This_Fun_Name = "RotTrans_IT:";
    Err_Msg = "";

    Drr_sqd = Drr_vec'*Drr_vec;
    Drr_sqd_3x3 = [Drr_sqd 0 0; 0 Drr_sqd 0; 0 0 Drr_sqd]
```

```
    I_at_COM = Rot*I*(Rot') + mass*[Drr_sqd_3x3 - Drr_vec*(Drr_vec')];

    Result = I_at_COM;
    Msg = This_Fun_Name + Err_Msg;
endfunction
```

4.14 **File name:** TT_Lnks_I_PCOM.sce

Purpose: Finds the centre of mass of the torque transfer links for the first design and then the orientation of the inertia matrix for those links.

```
// Finds the COM of the torque transfer links for the first design, and
// the orientation of its inertia matrix. These are at the mid-points of
// the torque transfer links.

// A simplification: The wrist angles are the same as the gears driving
// the wrist and those angles are with respect to the distal link. The
// driving gears at the base of the proximal arm make the same angle
// with the proximal arm. And they rotate in the same direction. The
// distal arm angle is the same as the angle of the driving gear at the
// bottom of the proximal arm but in the opposite direction.

// Base transfer gears have radii r_k.
// Need to add z location

//Rrel_0(2)*Rotate_XYZ(0, 0, tt_angZ)*[0; 0; dz_tt_pos];
dz_tt_com = [0; 0; dz_tt_pos];

if (tt_i == 3) then
    Prx_tt_beg = Rrel_0(2)*Rotate_XYZ(0, 0, tt_angZ)*[r_k; 0; 0];
    Prx_tt_Vec = 0.5*Knee_Coord - Prx_tt_beg;
    Prx_tt_Vec = Prx_tt_Vec/((Prx_tt_Vec'*Prx_tt_Vec)^0.5);
    Prx_tt_end = Prx_tt_beg + L1_Meters*Prx_tt_Vec;

    Mid_Dst_Lnk = 0.5*(Wrist_Org_Coord + Knee_Coord);
    Dst_tt_beg = Prx_tt_end;
    Dst_tt_Vec = Mid_Dst_Lnk - Dst_tt_beg;
    Dst_tt_Vec = Dst_tt_Vec/((Dst_tt_Vec'*Dst_tt_Vec)^0.5);
    Dst_tt_end = Dst_tt_beg + L2_Meters*Dst_tt_Vec;

else //...if (tti == 3) then
    // Now we're setting the driver at the wrist origin, but we need to
    // maintain the same variable names!!!
    Dst_tt_end = Rrel_0(3)*Rotate_XYZ(0, 0, tt_angZ+180)*[r_k; 0; 0] + Wrist_Org_Coord;
    Mid_Dst_Lnk = 0.5*(Wrist_Org_Coord + Knee_Coord);
    Dst_tt_Vec = Mid_Dst_Lnk - Dst_tt_end;
    Dst_tt_Vec = Dst_tt_Vec/((Dst_tt_Vec'*Dst_tt_Vec)^0.5);
    Dst_tt_beg = Dst_tt_end + L2_Meters*Dst_tt_Vec;

    Prx_tt_end = Dst_tt_beg;
    Prx_tt_Vec = 0.5*Knee_Coord - Prx_tt_end;
    Prx_tt_Vec = Prx_tt_Vec/((Prx_tt_Vec'*Prx_tt_Vec)^0.5);
    Prx_tt_beg = Prx_tt_end + L1_Meters*Prx_tt_Vec;
end //...if (tti == 3) then



// This gives the center of masses of the proximal and distal torque
// transfer linkages. However these must be written relative to the COMs
// of the proximal arm and the distal arm. Prx_COM = 0.5*Knee_Coord;
// Dst_COM = Mid_Dst_Lnk. Also the answers must lie on the xy plane,
// this is in keeping with the kinematic convention, which is used in
// the dynamics.

angX = Initial_Joint_Angles(1,1); //Static model params
angY = Initial_Joint_Angles(1,2); //Static model params
angZ = Animation_Angs(k, 1);
Rot_on2_X1 = Rotate_ZYX(-angX, -angY, -angZ);
angX = Initial_Joint_Angles(2,1); //Static model params
angY = Initial_Joint_Angles(2,2); //Static model params
angZ = Animation_Angs(k, 2);
Rot_on2_X2 = Rotate_ZYX(-angX, -angY, -angZ)*Rot_on2_X1;
```

```
angX = Initial_Joint_Angles(3,1); //Static model params
angY = Initial_Joint_Angles(3,2); //Static model params
angZ = Animation_Angs(k, 3);
Rot_on2_X3 = Rotate_ZYX(-angX, -angY, -angZ)*Rot_on2_X2;

Prx_TT_COM = Rot_on2_X2*(0.5*(Prx_tt_beg + Prx_tt_end) - 0.5*Knee_Coord);
Dst_TT_COM = Rot_on2_X3*(0.5*(Dst_tt_beg + Dst_tt_end)- Mid_Dst_Lnk);

// Also the answers must lie on the xy plane, this is in keeping with
// the kinematic convention, which is used in the dynamics.


// *****************************************************************
// NOW WE NEED TO FIND THE ORIENTATION OF THE TORQUE TRANSFER LINKS!!!!
// Prox - First translate Prx_tt_beg to the origin.
Prx_tt_end2 = Prx_tt_end - Prx_tt_beg;

// Rotate onto xz-plane
Rot_on2_XZ = Rotate_ZYX(0, 0, -Animation_Angs(k, 1));
Prx_tt_end2 = Rot_on2_XZ*Prx_tt_end2;
Knee_Coord2 = Rot_on2_XZ*Knee_Coord;
Wrist_Org_Coord2 = Rot_on2_XZ*Wrist_Org_Coord;

KC_x = Knee_Coord2(1);
KC_y = Knee_Coord2(2);
KC_z = Knee_Coord2(3);
PTE_dx = Prx_tt_end2(1);
PTE_dy = Prx_tt_end2(2)
PTE_dz = Prx_tt_end2(3)
// [c  0 -s; 0  1  0; s  0   c]*Knee_Coord = Prx_tt_end

if (KC_x == 0)&(KC_z == 0) then //infinite number of solutions!!!!
    disp('Problem: Infinite number of solutions for proximal TT' + ...
         ' bar orientation!')
    abort();
elseif (KC_x == 0) then
    sin_theta1 =  -PTE_dx/KC_z;
    cos_theta1 =  PTE_dz/KC_z;
else
    sin_theta1 = (KC_x*PTE_dz - KC_z*PTE_dx)/(KC_x^2 + KC_z^2)
    cos_theta1 = (PTE_dx + sin_theta1*KC_z)/KC_x;
end

//CHECK ...
//Prx_tt_end2b = [cos_theta1  0 -sin_theta1; 0 1 0; ...
//                            sin_theta1 0 cos_theta1]*Knee_Coord2;
Theta1 = A_sin_cos_d(sin_theta1, cos_theta1);
Theta1 = -Theta1; //You were looking at it from the wrong side,
// hence the negative!
// Rotate_ZYX(0, Theta1, 0)*Knee_Coord2




Dst_tt_end2 = Dst_tt_end - Dst_tt_beg;

//Rotate onto xz-plane
Dst_tt_end2 = Rotate_ZYX(0, 0, -Animation_Angs(k, 1))*Dst_tt_end2;
Wrist_Org_End2 = Wrist_Org_Coord2 - Knee_Coord2;

WO_x = Wrist_Org_End2(1);
WO_y = Wrist_Org_End2(2);
WO_z = Wrist_Org_End2(3);
DTE_dx = Dst_tt_end2(1);
DTE_dy = Dst_tt_end2(2)
DTE_dz = Dst_tt_end2(3)
// [c  0 -s; 0  1  0; s  0  c]*Distal_link = Dst_tt_end

if (WO_x == 0)&(WO_z == 0) then //infinite number of solutions!!!!
    disp('Problem: Infinite number of solutions for distal TT ' + ...
         'bar orientation!')
    abort();
elseif (WO_x == 0) then
    sin_theta2 =  -DTE_dx/WO_z;
    cos_theta2 =  DTE_dz/WO_z;
else
```

```
    sin_theta2 = (WO_x*DTE_dz - WO_z*DTE_dx)/(WO_x^2 + WO_z^2)
    cos_theta2 = (DTE_dx + sin_theta2*WO_z)/WO_x;
end

// CHECK ...
// Dst_tt_end2b = [cos_theta2  0 -sin_theta2; 0 1 0; ...
//                              sin_theta2 0 cos_theta2]*Wrist_Org_End2;
Theta2 = A_sin_cos_d(sin_theta2, cos_theta2);
Theta2 = -Theta2; //You were looking at it from the wrong side,
// hence the negative!
// Rotate_XYZ(0, Theta2, 0)*Wrist_Org_End2
```

4.15 **File name:** Calc_Energy_Used.sce

Purpose: Calculates the energy used for each of the 4 dynamic models by calculating the angular work done, which is given by torque multiplied by angular displacement.

```
//// Calc_Energy_Used.

// Dynmod_Torques = tlist(["Dynmod _Torques", "Pozzi", ...
//                         "Serial", "Design2", "Design1"], [], [], [], []);
Dynmod_Energy = tlist(["Dynmod_Energy", "Pozzi", "Serial", ...
                       "Design2", "Design1"], [], [], [], []);
Model_Energy = tlist(["Model_Energy", "Joints", "Total"], [], []);

// Energy used equals torque times angular displacement in radians
// We need to find the energy used for each of the 6 joints, this is done
// by taking the average torque between two consecutive points and
// multiplying it by the change in angle, and integrating that value.
// We also sum all the motor energies to get total energy used at each
// point.

Deg_2_Rad = %pi/180;
SZ_Animation_Angs = size(Animation_Angs); //Animation_Angs has 6 columns
end_pose_k = SZ_Animation_Angs(1);
end_Ji = 6;
end_model = 5;

for model = 2 : end_model //... Represents the current model, index
                          //... for tree starts at 2!

    tmp_energy_mat = []; //Reset

    for Ji = 1 : end_Ji // Represents joints in current model

        tmp_energy_jnt = []; //Reset
        tmp_energy_jnt(1) = 0;

        for pose_k = 2 : end_pose_k // Represents torque and angle values
                                    // at specific times

            delta_ang_Ji = Deg_2_Rad*(Animation_Angs(pose_k, Ji) -...
                           Animation_Angs(pose_k-1, Ji));
                           //Find change in radians.
            ave_torq_Ji = 0.5*(Dynmod_Torques(model)(pose_k,Ji) + ...
                           Dynmod_Torques(model)(pose_k-1,Ji));
            tmp_energy_jnt(pose_k) = tmp_energy_jnt(pose_k-1) + ...
                           abs(delta_ang_Ji*ave_torq_Ji);
                           // Energy use is always positive!

        end //...for ang_k = 2 : end_ang_k // Represents torque and angle
                                           //values at specific times

        tmp_energy_mat = [tmp_energy_mat, tmp_energy_jnt]; //Store values

    end //...for Ji = 1 : end_Ji  // Represents joints in current model
```

```
    Model_Energy.Joints = tmp_energy_mat; //Save matrix to correct model
    Model_Energy.Total = sum(tmp_energy_mat, 'c');
    Dynmod_Energy(model) = Model_Energy;

end //...for model = 2 : end_model // Represents the current model,
                                   // index for tree starts at 2!




// ****************************************
// START: Save energy data to text files....
if ~exists('Save_Data') then
    global Save_Data;
    Save_Data = 1;
end //...if ~exists('Save_Data') then

if (Save_Data == 1) & ~exists('Multi_Path') then //Saving data

    end_jnt_cnt = 3; //6;
    Labels = '';
    Headings = '';
    Matrix_Data = [];
    End_Char = ["\t", "\t", "\n"];
    for jnt_cnt = 1 : end_jnt_cnt //...saving data

        //File_Name = 'J' + string(jnt_cnt) + '_Energy.txt';
        PJ_i = Dynmod_Energy.Pozzi.Joints(:,jnt_cnt);
        SJ_i = Dynmod_Energy.Serial.Joints(:,jnt_cnt);
        D1J_i = Dynmod_Energy.Design1.Joints(:,jnt_cnt);
        D2J_i = Dynmod_Energy.Design2.Joints(:,jnt_cnt);
        Matrix_Data = [Matrix_Data, Time_Vec, PJ_i, SJ_i, D1J_i, D2J_i];
        Labels = Labels + "Time\t" + "Pozzi_JE" + string(jnt_cnt) +"\t" + ...
                     + "Serial_JE" + string(jnt_cnt) +"\t" + ...
                     + "Design1_JE" + string(jnt_cnt) +"\t" + ...
                     + "Design2_JE" + string(jnt_cnt) + "\t"; //"\n";

            Headings = Headings + lbl_heading + " joint" + string(jnt_cnt) + ...
                       " energy (" + lbl_Full_MM (i_MM) + ...
                       string(Total_Play_Time) + 's)\t\t\t\t\t';

//        // Save2_TXT(Location, File_Name, Matrix_Data, Labels)
//        Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);

    end // ...for jnt_cnt = 1 : end_jnt_cnt //...saving data


    //File_Name = 'Total_Energy.txt';
    PJ_i = Dynmod_Energy.Pozzi.Total;
    SJ_i = Dynmod_Energy.Serial.Total;
    D1J_i = Dynmod_Energy.Design1.Total;
    D2J_i = Dynmod_Energy.Design2.Total;
    Matrix_Data = [Matrix_Data, Time_Vec, PJ_i, SJ_i, D1J_i, D2J_i];
    Labels = Labels + "Time\tPozzi_TE\tSerial_TE\tDesign1_TE\tDesign2_TE\n";
    Headings = Headings + lbl_heading + " total energy (" + ...
                    lbl_Full_MM (i_MM) + ...
                    string(Total_Play_Time) + 's)\t\t\t\t\n';


    File_Location = Dyn_Scr_Path + 'Dynamics Data\'+ Current_Path + ...
                   '\Constant_Orientation_' + string(lbl_Motor_Mass(i_MM)) +...
                    '_' + string(Total_Play_Time) + 's\';

    Labels = Headings + Labels;
    File_Name = 'All_Energy_Data.txt';
    // Save2_TXT(Location, File_Name, Matrix_Data, Labels)
    Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);
//
//    File_Name_and_Location = File_Location + 'DynMod_Energy.dat';
//    save (File_Name_and_Location, Time_Vec, Dynmod_Energy);
end //...if (Save_Data == 1) then  //Saving data
// END: Save energy data to text files....
// ****************************************
```

4.16 **File name:** Multi_Path_ECalc.sce

Purpose: Calculates the energy used for multiple horizontal, vertical and radial paths, for all models so that they can be compared. The number of paths is set using the 'num_paths' variable. For a low number of paths comment out the 'global Dont_show;' and 'Dont_show = 1;' lines, to see the animation as it moves through the number of paths set.

```
// THIS CODE CREATES THE DATA FOR MULTIPLE HORIZONTAL, VERTICAL OR RADIAL
// PATHS AND CALCULATES THE ENERGEY USED AT THE END OF THE PATH. THE
// DATA FOR EACH MODEL IS THEN PLOTTED AGAINST EACHOTHER.....


if ~exists('delete_and_clear') then
    global delete_and_clear;
    delete_and_clear = 1;
end //...if ~exists('delete_and_clear') then

if delete_and_clear then
    clc;
    xdel(winsid());
    //Destroy all existing figures, this will be reinitialized in the
    //forward kinematics routine!
    clearglobal();
    clear;

    global delete_and_clear;
    delete_and_clear = 0;
end //...if delete_and_clear then



if ~exists('LinkageTester') then // Execute Inverse_Kinematics

    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\"
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";
    exec(IK_Script_Path, -1);
    disp("Executing Path Following!");
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\"
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";

else //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics

    //UPDATES THE REFERENCE ANGLE GRAPHIC
    Script_Path = Kin_Scr_Path + "Angle_Reference_Display.sce";
    exec(Script_Path, -1);

end //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics



// ********************************************************************
// DONT SAVE ANY DATA - THIS IS FOR ALL THE SCRIPT FILES THAT ARE CALLED
// FROM HERE. THE OPTION TO SAVE DATA FOR THIS PARTICULAR SCRIPT IS AT
// THE END.
global Save_Data;
Save_Data = 0;  // 1 -> Save
// ********************************************************************


// ******************************************************
// THIS OPTION STOPS THE DISPLAY OF THE FORWARD KINEMATICS
// SET IT TO 1 FOR LARGE DATASETS TO SPEED UP CALCULATION!
// COMMENT IT OUT FOR SMALL DATASETS TO SEE ANIMATION!
global Dont_show;
Dont_show = 1;
// ******************************************************


// ********************************************************************
```

```
// SCRIPT PATHS
Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";
Scr_Lin_Fol = Dyn_Scr_Path + "Dyn_Path_Fol_Lin.sce";
Scr_Calc_VA = Dyn_Scr_Path + "Calc_Vel_Acc.sce";
Scr_Inv_Dyn = Dyn_Scr_Path + "Inverse_Dynamics.sce";
Scr_Calc_EU = Dyn_Scr_Path + "Calc_Energy_Used.sce";
// ********************************************************************


// First we need to create center points and linear displacements
// SET THE CURRENT PATH.......
if ~exists('Current_Path') then
    All_paths = ["H_line1";...
                 "H_line2";...
                 "V_line1";...
                 "V_line2";...
                 "R_line1";...
                 "R_line2"];

    global Current_Path;
    Current_Path = All_paths(5);

    Saved_Path = Dyn_Scr_Path + "Dynamics Data\Current_Path.dat";
    save(Saved_Path, Current_Path);
end // if ~exists('Current_Path')


// Change the path here! Valid paths for this code are 'H_line1',
// 'V_line1' and 'R_line1', so 1, 3 and 5.
Current_Path = All_paths(5);



if ~strcmp(Current_Path, "H_line1") then
    lbl_heading = "Multi-horizontal line paths";

elseif ~strcmp(Current_Path, "V_line1") then
    lbl_heading = "Multi-vertical line paths";

elseif ~strcmp(Current_Path, "R_line1") then
    lbl_heading = "Multi-radial line paths";
end



// Now we vary centre positions before calling the Dyn_Path_Fol_Lin
// script which will solve the inverse kinematics along the line.

// For horizontal lines we vary along the Y-axis
// For vertical lines we vary along the Y-axis as well
// For radial lines we vary along the z axis

Disp_Len = L1b_len;
// This will set the half line displacement
if ~strcmp(Current_Path, "H_line1") then
    // Adds half line displacement along X, horizontal lines
    Half_Line_Disp = [0.5*Disp_Len; 0; 0];
    Start_CenPos = [0; 0.3*L1b_len; 0.7*L1b_len];
    End_CenPos = [0; 1.6*L1b_len; 0.7*L1b_len];

elseif ~strcmp(Current_Path, "V_line1") then
    // Half line displacement along Z, vertical lines
    Half_Line_Disp = [0; 0; 0.5*Disp_Len];
    Start_CenPos = [0; 0.3*L1b_len; 0.7*L1b_len];
    End_CenPos = [0; 1.6*L1b_len; 0.7*L1b_len];

elseif ~strcmp(Current_Path, "R_line1") then
    Half_Line_Disp = [0; 0.5*Disp_Len; 0];
    Start_CenPos = [0; 0.7*L1b_len; 0.5*L1b_len];
    End_CenPos = [0; 0.7*L1b_len; 1.5*L1b_len];

else
    disp("Pick a correct path!")
    abort();
end
```

```
path_dx = End_CenPos(1)-Start_CenPos(1);
path_dy = End_CenPos(2)-Start_CenPos(2);
path_dz = End_CenPos(3)-Start_CenPos(3);
tmp_Path_Dist = (path_dx^2 + path_dy^2 + path_dz^2)^0.5;

num_paths = 28; // just set it to the number of paths you want


path_dx_inc = path_dx/num_paths;
path_dy_inc = path_dy/num_paths;
path_dz_inc = path_dz/num_paths;



str_num_paths = string(num_paths);


PJE = [];
SJE = [];
D1JE = [];
D2JE = [];
All_CPos = [];
get_EDat = 1;


Save_Data = 0;
global Multi_Path;
Multi_Path = 1;

Motor_Mass = [0, 1];
lbl_Motor_Mass = ["LM", "HM"];
lbl_Full_MM = ["light motors, ", "heavy motors, "]
Tot_Time_P = [5, 5/20];

for i_MM = 1 : 2
    for i_TTP = 1 : 2

        disp("i_MM: " + string(i_MM) + ' of 2;' + "i_TTP: " + string(i_TTP) + ' of 2;');
        Iteration = string(2^(i_MM) + i_TTP - 2);
        global Heavy_motors;
        Heavy_motors = Motor_Mass(i_MM); //0 - light; 1 - heavy
        global Total_Play_Time;
        Total_Play_Time = Tot_Time_P(i_TTP); //5/20; // /20


        for path_inc = 0 : num_paths
            Centre_Pos = Start_CenPos + path_inc*[path_dx_inc; path_dy_inc; path_dz_inc];
            // Call Dyn_Path_Fol_Lin
            exec(Scr_Lin_Fol, -1);

            if (get_EDat == 1) then
                //GET ENERGY DATA
                exec(Scr_Calc_VA, -1);
                exec(Scr_Inv_Dyn, -1);
                exec(Scr_Calc_EU, -1);

                // Get the required energy data
                All_CPos(:, path_inc+1) = Centre_Pos;
                PJE(path_inc+1) = Dynmod_Energy.Pozzi.Total($);
                SJE(path_inc+1) = Dynmod_Energy.Serial.Total($);
                D1JE(path_inc+1) = Dynmod_Energy.Design1.Total($);
                D2JE(path_inc+1) = Dynmod_Energy.Design2.Total($);
            end //...if (get_EDat == 1) then

            disp(Iteration + ':- ' + string(path_inc) + ' of ' + str_num_paths);
        end //...for path_inc = 0 : num_paths


        Headings = '';

        Save_Data = 1;
        // *********************************************************************
        // START: Save energy data to text files....
        if (Save_Data == 1) then //Saving data

            File_Location = Dyn_Scr_Path + 'Dynamics Data\Multi_Line_Paths\'+ ...
            Current_Path + '\';
```

```
            File_Name = 'MPTE_' + Current_Path + '_' + string(lbl_Motor_Mass(i_MM)) +...
            '_' + string(Total_Play_Time) + 's.txt';
            Matrix_Data = [All_CPos', PJE, SJE, D1JE, D2JE];
            Labels = "X\tY\tZ\tPozzi_TE\tSerial_TE\tDesign1_TE\tDesign2_TE\n";

            Headings = lbl_heading + ", " + lbl_Full_MM(i_MM) + ...
                       string(Total_Play_Time) + 's\n';

            Labels = Headings + Labels;
            // Save2_TXT(Location, File_Name, Matrix_Data, Labels)
            Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);

            //    File_Name_and_Location = File_Location + 'Multi_Path_Energy_' + ...
            //                             Current_Path + '.dat';
            //    save (File_Name_and_Location, All_CPos, PJE, SJE, D1JE, D2JE);
        end //...if (Save_Data == 1) then //Saving data
        // END: Save energy data to text files....
        // ********************************************************************

    end//...for i_TTP = 1 : 2
end//...for i_MM = 1 : 2

disp('Done!');
```

4.17 **File name:** Multi_CirPath_ECalc.sce

Purpose: Calculates the energy used for each of the 3 main joints responsible for end-effector spatial positioning. It singles out each joint (other joint angles were kept constant) and calculates the joint energy for best and worse case arm mass distribution. The joints are also rotated through their full range of motion with large displacements.

```
// THIS CODE CREATES THE DATA FOR MULTIPLE HORIZONTAL, VERTICAL OR RADIAL
// PATHS AND CALCULATES THE ENERGEY USED AT THE END OF THE PATH. THE
// DATA FOR EACH MODEL IS THEN PLOTTED AGAINST EACHOTHER.....


if ~exists('delete_and_clear') then
    global delete_and_clear;
    delete_and_clear = 1;
end //...if ~exists('delete_and_clear') then

if delete_and_clear then
    clc;
    xdel(winsid());
    //Destroy all existing figures, this will be reinitialized in the
    //forward kinematics routine!
    clearglobal();
    clear;

    global delete_and_clear;
    delete_and_clear = 0;
end //...if delete_and_clear then



if ~exists('LinkageTester') then // Execute Inverse_Kinematics

    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";
    exec(IK_Script_Path, -1);
    disp("Executing Path Following!");
    Kin_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Kinematics\";
    IK_Script_Path = Kin_Scr_Path + "Inverse_Kinematics.sce";

else //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics
```

```
    //UPDATES THE REFERENCE ANGLE GRAPHIC
    Script_Path = Kin_Scr_Path + "Angle_Reference_Display.sce";
    exec(Script_Path, -1);
    //show_window(); //Raise the current graphics window

end //...if ~exists('LinkageTester') then // Execute Inverse_Kinematics



// ************************************************************************
// DONT SAVE ANY DATA - THIS IS FOR ALL THE SCRIPT FILES THAT ARE CALLED
// FROM HERE. THE OPTION TO SAVE DATA FOR THIS PARTICULAR SCRIPT IS AT
// THE END.
global Save_Data;
Save_Data = 0;  // 1 -> Save
// ************************************************************************


// ************************************************************************
// THIS OPTION STOPS THE DISPLAY OF THE FORWARD KINEMATICS
// SET IT TO 1 FOR LARGE DATASETS TO SPEED UP CALCULATION!
// COMMENT IT OUT FOR SMALL DATASETS TO SEE ANIMATION!
global Dont_show;
Dont_show = 1;
// ************************************************************************


// ************************************************************************
global Heavy_motors;
Heavy_motors = 0;
Motor_Mass = [0, 1];
lbl_Motor_Mass = ["LM", "HM"];
lbl_Full_MM = ["light motors, ", "heavy motors, "]
global lbl_config_ang;

global Total_Play_Time;
Tot_Time_P = [40, 2]; // differs by a factor of 20
//************************************************************************


// ************************************************************************
// SCRIPT PATHS
Dyn_Scr_Path = "D:\Program Files\scilab-5.3.3\SCI_Work\HKM Dynamics\";
Scr_Cir_Fol = Dyn_Scr_Path + "Dyn_Path_Fol_Cir.sce";
Scr_Calc_VA = Dyn_Scr_Path + "Calc_Vel_Acc.sce";
Scr_Inv_Dyn = Dyn_Scr_Path + "Inverse_Dynamics.sce";
Scr_Calc_EU = Dyn_Scr_Path + "Calc_Energy_Used.sce";
// ************************************************************************



global plot_data;
plot_data = 0;

global Multi_Path;
Multi_Path = 1;

All_Circ_Paths = ["C1_Ang1";...
                  "C2_Ang1";...
                  "C3_Ang2";...
                  "C4_Ang2";...
                  "C5_Ang3";...
                  "C6_Ang3"];

num_paths = 28;//28
path_end_cnt = 5;
data_set_cnt = 0;
data_set_totcnt = string(4*path_end_cnt);

// RUNS THROUGH ALL PATHS AUTOMATICALLY
for path_cnt = 4 : path_end_cnt

    // Can rund a for loop here to go through all paths
    Current_Path = All_Circ_Paths(path_cnt);

    if (~strcmp(Current_Path, "C1_Ang1")) | (~strcmp(Current_Path, "C2_Ang1")) then
```

```
    Ang_Range = [0, 180];
    lbl_heading = "Varying angle 1";
elseif (~strcmp(Current_Path, "C3_Ang2")) | (~strcmp(Current_Path, "C4_Ang2")) then
    Ang_Range = [0, 90];
    lbl_heading = "Varying angle 2";
elseif (~strcmp(Current_Path, "C5_Ang3")) | (~strcmp(Current_Path, "C6_Ang3")) then
    Ang_Range = [-90, 90];
    lbl_heading = "Varying angle 3";
else
    //???
end


Ang_Range_inc = (Ang_Range(2)-Ang_Range(1))/num_paths;


PJE = [];
SJE = [];
D1JE = [];
D2JE = [];
Ang_Range_Vec = [];


global All_Total_Time;
All_Total_Time = [];



//These 2 for loops run through the motor mass and speed!!!!
for i_MM = 1 : 2
    for i_TTP = 1 : 2

        data_set_cnt = data_set_cnt + 1;
        disp('Data set ' + string(data_set_cnt) + ' of ' + data_set_totcnt);

        global Heavy_motors;
        Heavy_motors = Motor_Mass(i_MM); //0 - light; 1 - heavy
        global Total_Play_Time;
        Total_Play_Time = Tot_Time_P(i_TTP); //5/20; // /20

        for path_inc = 1 : num_paths

            disp("path_inc = " + string(path_inc));
            // Call Dyn_Path_Fol_Cir
            Ang_Range_Vec(path_inc) = Ang_Range(1) + path_inc*Ang_Range_inc;
            exec(Scr_Cir_Fol, -1);

            exec(Scr_Calc_VA, -1);
            //plot_data = 0; //Only want to see it for 1 path
            All_Total_Time(path_inc) = Total_Play_Time;

            exec(Scr_Inv_Dyn, -1);
            exec(Scr_Calc_EU, -1);

            // Get the required energy data
            PJE(path_inc) = Dynmod_Energy.Pozzi.Total($);
            SJE(path_inc) = Dynmod_Energy.Serial.Total($);
            D1JE(path_inc) = Dynmod_Energy.Design1.Total($);
            D2JE(path_inc) = Dynmod_Energy.Design2.Total($);

        end //...for path_inc = 0 : num_paths



        Save_Data = 1;
        // *********************************************************************
        // START: Save energy data to text files....
        if (Save_Data == 1) then //Saving data


            File_Location = Dyn_Scr_Path + 'Dynamics Data\Multi_Circ_Paths\'+ ...
            Current_Path + '\';

            File_Name = 'MPTE_' + Current_Path + '_' + string(lbl_Motor_Mass(i_MM)) +...
            '_' + string(Total_Play_Time) + 's.txt';
            Matrix_Data = [Ang_Range_Vec, PJE, SJE, D1JE, D2JE];
            Labels = "Angle\tPozzi_TE\tSerial_TE\tDesign1_TE\tDesign2_TE\n";
```

```
                Headings = lbl_heading + ', ' + lbl_config_ang + ...
                lbl_Full_MM(i_MM) + string(Total_Play_Time) + 's\n';

                Labels = Headings + Labels;

                // Save2_TXT(Location, File_Name, Matrix_Data, Labels)
                Save2_TXT(File_Location, File_Name, Matrix_Data, Labels);

                //File_Name_and_Location = File_Location + 'Multi_Path_Energy_' + ...
                //Current_Path + '.dat';
                //save (File_Name_and_Location, Ang_Range_Vec, PJE, SJE, D1JE, D2JE);
            end //...if (Save_Data == 1) then //Saving data
            // END: Save energy data to text files....
            // ************************************************************************

        end//...for i_TTP = 1 : 2
    end//...for i_MM = 1 : 2

end //...for path_cnt = 1 : 2

disp('Done!');
```

4.18 **File name:** Dyn_Path_Fol_Cir.sce

Purpose: Creates the joint angle vectors for circular motion, varying one joint only while keeping others constant.

```
//All_Circ_Paths = ["C1_Ang1";...
//                  "C2_Ang1";...
//                  "C3_Ang2";...
//                  "C4_Ang2";...
//                  "C5_Ang3";...
//                  "C6_Ang3"];

//Current_Path = All_Circ_Paths(1);
global lbl_config_ang;

if (~strcmp(Current_Path, "C1_Ang1")) then
    cir_idx = 1;
    Robot_Angs = [0, 0, 0, 0, 0, 0,];
    lbl_config_ang = "angle(2) = 0, ";
elseif (~strcmp(Current_Path, "C2_Ang1")) then
    cir_idx = 2;
    Robot_Angs = [0, 90, 0, 0, 0, 0,];
    lbl_config_ang = "angle(2) = 90, ";
elseif (~strcmp(Current_Path, "C3_Ang2")) then
    cir_idx = 3;
    Robot_Angs = [0, 0, 0, 0, 0, 0,];
    lbl_config_ang = "angle(3) = 0, ";
elseif (~strcmp(Current_Path, "C4_Ang2")) then
    cir_idx = 4;
    Robot_Angs = [0, 0, 170, 0, 0, 0,];
    lbl_config_ang = "angle(3) = 170, ";
elseif (~strcmp(Current_Path, "C5_Ang3")) then
    cir_idx = 5;
    Robot_Angs = [0, 0, 0, 0, 0, 0,];
    lbl_config_ang = "angle(2) = 0, ";
elseif (~strcmp(Current_Path, "C6_Ang3")) then
    cir_idx = 6;
    Robot_Angs = [0, 0, 0, 0, 0, 0,];
    lbl_config_ang = "angle(2) = 0, ";
else
    // Cannot choose cir_idx = 1;
    disp('Cannot choose cir_idx, at Dyn_Path_Fol_Cir!');
    abort();
    //???
end

Ang_res = (Ang_Range_Vec(path_inc) - Ang_Range(1))/28; //must be divided
// by an even number, this prevents wiggle on second half of velocity
// curve!
```

```
Joint_Index.Current = ceil(cir_idx/2);
J_Index = Joint_Index.Current;


// just create CP_Data; CP_Data has 3 rows, then we can call
// Calc_Vel_Acc - this is much quicker than having to rewrite that code
// for these paths!
// CP_Data - 3xn
// Animation_Angs - nx6 matrix
// Time_Vec - nx1
Animation_Angs = [];
CP_Data = [];
for AR_ang = Ang_Range(1) : Ang_res : Ang_Range_Vec(path_inc)

    Robot_Angs(Joint_Index.Current) = AR_ang;
    Animation_Angs = [Animation_Angs; Robot_Angs];
    for k = 1 : 6
        // if it exists do this to select the correct angle set!!!!
        Kin_Model.Joints(k).Joint_Angle.angZd = Robot_Angs(k);
    end //...for k = 1 : 6

    if ~exists('Dont_show') then
        Update_Graphics_Model(); //show graphics
    else
        //Just the Forward kinematics multiplication
        Num_of_Joints = size(Kin_Model.Joints);
        for jnt_i = J_Index : Num_of_Joints
            angX = Kin_Model.Joints(jnt_i).Joint_Angle.angXd;
            angY = Kin_Model.Joints(jnt_i).Joint_Angle.angYd;
            angZ = Kin_Model.Joints(jnt_i).Joint_Angle.angZd;

            Link_Length = Kin_Model.Joints(jnt_i).Link_Length;
            X_vec = Link_Length*[1; 0; 0];

            if (jnt_i == 1)
                RotMatAll(jnt_i) = Rotate_XYZ(angX, angY, angZ);
                TransVecAll(jnt_i) = RotMatAll(jnt_i)*X_vec;
                Trans_Vec = [0 0; 0 0; 0 0]';
                // Can Update the link vector here
                Kin_Model.Joints(jnt_i).Triad_Data.Link_Handle.data = Trans_Vec;

            else // from joints 2 to 6
                //Rx*Ry*Rz;
                RotMatAll(jnt_i) = RotMatAll(jnt_i-1)*Rotate_XYZ(angX, angY, angZ);
                TransVecAll(jnt_i) = TransVecAll(jnt_i-1) + RotMatAll(jnt_i-1)*X_vec;
            end //...if (jnt_i == 1)
        end //...for jnt_i = J_Index : Num_of_Joints
    end //...if ~exists('Dont_show') then

    CP_Data = [CP_Data, TransVecAll(6)];
end //...for AR_ang = Ang_Range(1) : Ang_res : Ang_Range_Vec(path_inc)

CP_Data(:, $+1) = CP_Data(:, $);//Repeated last point!
Animation_Angs($+1, :) = Animation_Angs($, :);//Repeated last angle set!


SZ_CP_Data = size(CP_Data);
```

APPENDIX B: Micro Controller Code

File name: servo.c

Purpose: Generates PWM signals (using the controller's timers) to control servo motors, and communicates with PC through RS232 ports. This code was taken from my master's project, 'Parallel robot design incorporating a direct end effector sensing system', submitted to UKZN in November 2007.

```
/*****************************************
This program was produced by the
CodeWizardAVR V1.23.7a Standard
Automatic Program Generator
© Copyright 1998-2002 HP InfoTech s.r.l.
http://www.hpinfotech.ro
e-mail:office@hpinfotech.ro

Project :
Version :
Date    :
Author  :
Company :
Comments:


Chip type           : ATmega128
Program type        : Application
Clock frequency     : 16.000000 MHz
Memory model        : Small
Internal SRAM size  : 4096
External SRAM size  : 0
Data Stack size     : 1024
*****************************************/

#include <mega128.h>
#include <math.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

//*****************************************
//********* My Global Variables **************
#define NUETRAL_POS 41535
#define TIMER_INC 25
unsigned int New_Motor_Values[6] = {NUETRAL_POS, NUETRAL_POS, NUETRAL_POS, NUETRAL_POS,
NUETRAL_POS, NUETRAL_POS};
unsigned int Current_Motor_Values[6] = {NUETRAL_POS, NUETRAL_POS, NUETRAL_POS, NUETRAL_POS,
NUETRAL_POS, NUETRAL_POS};
int Temp;
unsigned char Motor_ID;
unsigned int Received_Number = 0;
unsigned char Toggle_Comp_Int_Values = 0;
unsigned char Start = 0;
//*****************************************

//***********  Function Prototypes  ***********
void Comms_Received_Data (char Data);
void Send_Data (unsigned char M_ID, unsigned int M_Value);
```

```
void putchar(char c);
unsigned int Inc_Timr_Vals (unsigned int CMV, unsigned int NMV);

//*****************************************

// USART0 Receiver buffer
#define RX_BUFFER_SIZE0 8
char rx_buffer0[RX_BUFFER_SIZE0];
unsigned char rx_wr_index0,rx_rd_index0,rx_counter0;
// This flag is set on USART0 Receiver buffer overflow
bit rx_buffer_overflow0;

// USART0 Receiver interrupt service routine
#pragma savereg-
interrupt [USART0_RXC] void uart0_rx_isr(void)
{
char status,data;
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
status=UCSR0A;
data=UDR0;


if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer0[rx_wr_index0]=data;
   if (++rx_wr_index0 == RX_BUFFER_SIZE0) rx_wr_index0=0;
   if (++rx_counter0 == RX_BUFFER_SIZE0)
      {
      rx_counter0=0;
      rx_buffer_overflow0=1;
      };
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm

       // **** check data received ****
       Comms_Received_Data (data);
}



#pragma savereg+

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART0 Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter0==0);
data=rx_buffer0[rx_rd_index0];
if (++rx_rd_index0 == RX_BUFFER_SIZE0) rx_rd_index0=0;
#asm("cli")
--rx_counter0;
#asm("sei")
return data;
}
#pragma used-
#endif

// USART0 Transmitter buffer
#define TX_BUFFER_SIZE0 8
```

```
char tx_buffer0[TX_BUFFER_SIZE0];
unsigned char tx_wr_index0,tx_rd_index0,tx_counter0;

// USART0 Transmitter interrupt service routine
#pragma savereg-
interrupt [USART0_TXC] void uart0_tx_isr(void)
{
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
if (tx_counter0)
   {
   --tx_counter0;
   UDR0=tx_buffer0[tx_rd_index0];
   if (++tx_rd_index0 == TX_BUFFER_SIZE0) tx_rd_index0=0;
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART0 Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter0 == TX_BUFFER_SIZE0);
#asm("cli")
if (tx_counter0 || ((UCSR0A & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer0[tx_wr_index0]=c;
   if (++tx_wr_index0 == TX_BUFFER_SIZE0) tx_wr_index0=0;
   ++tx_counter0;
   }
else UDR0=c;
#asm("sei")
}
#pragma used-
#endif

// USART1 Receiver buffer
#define RX_BUFFER_SIZE1 8
char rx_buffer1[RX_BUFFER_SIZE1];
unsigned char rx_wr_index1,rx_rd_index1,rx_counter1;
// This flag is set on USART1 Receiver buffer overflow
bit rx_buffer_overflow1;

// USART1 Receiver interrupt service routine
#pragma savereg-
interrupt [USART1_RXC] void uart1_rx_isr(void)
{
char status,data;
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
status=UCSR1A;
data=UDR1;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer1[rx_wr_index1]=data;
```

```
   if (++rx_wr_index1 == RX_BUFFER_SIZE1) rx_wr_index1=0;
   if (++rx_counter1 == RX_BUFFER_SIZE1)
      {
      rx_counter1=0;
      rx_buffer_overflow1=1;
      };
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

// Get a character from the USART1 Receiver buffer
#pragma used+
char getchar1(void)
{
char data;
while (rx_counter1==0);
data=rx_buffer1[rx_rd_index1];
if (++rx_rd_index1 == RX_BUFFER_SIZE1) rx_rd_index1=0;
#asm("cli")
--rx_counter1;
#asm("sei")
return data;
}
#pragma used-
// USART1 Transmitter buffer
#define TX_BUFFER_SIZE1 8
char tx_buffer1[TX_BUFFER_SIZE1];
unsigned char tx_wr_index1,tx_rd_index1,tx_counter1;

// USART1 Transmitter interrupt service routine
#pragma savereg-
interrupt [USART1_TXC] void uart1_tx_isr(void)
{
#asm
    push r26
    push r27
    push r30
    push r31
    in   r26,sreg
    push r26
#endasm
if (tx_counter1)
   {
   --tx_counter1;
   UDR1=tx_buffer1[tx_rd_index1];
   if (++tx_rd_index1 == TX_BUFFER_SIZE1) tx_rd_index1=0;
   };
#asm
    pop  r26
    out  sreg,r26
    pop  r31
    pop  r30
    pop  r27
    pop  r26
#endasm
}
#pragma savereg+

// Write a character to the USART1 Transmitter buffer
#pragma used+
void putchar1(char c)
{
while (tx_counter1 == TX_BUFFER_SIZE1);
#asm("cli")
if (tx_counter1 || ((UCSR1A & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer1[tx_wr_index1]=c;
   if (++tx_wr_index1 == TX_BUFFER_SIZE1) tx_wr_index1=0;
   ++tx_counter1;
```

```
   }
else UDR1=c;
#asm("sei")
}
#pragma used-

// Standard Input/Output functions
#include <stdio.h>

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here


}

// Timer 0 output compare interrupt service routine
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
// Place your code here

}

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
        unsigned char inc_Index;
        // Place your code here
        PORTA = 0x00;
        PORTF = 0x00;

        if (Toggle_Comp_Int_Values >= 6)
   {
        Toggle_Comp_Int_Values = 0;
   }


        Current_Motor_Values[Toggle_Comp_Int_Values]              =              Inc_Timr_Vals
(Current_Motor_Values[Toggle_Comp_Int_Values], New_Motor_Values[Toggle_Comp_Int_Values]);
        OCR1A = Current_Motor_Values[Toggle_Comp_Int_Values];  // Motor 1 and 2

        inc_Index = Toggle_Comp_Int_Values + 1;
        Current_Motor_Values[inc_Index]   =   Inc_Timr_Vals   (Current_Motor_Values[inc_Index],
New_Motor_Values[inc_Index]);
        OCR1B = Current_Motor_Values[inc_Index];

   Toggle_Comp_Int_Values = Toggle_Comp_Int_Values + 2;

   if (Start == 0)
   {
        TCCR1B=0x00; //Stop the timer
   }
}

// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
        // Place your code here
        if (Toggle_Comp_Int_Values == 2)
        {
                PORTA.7 = 1;
                PORTF = (0b10000000 | PORTF);
        }

        else if (Toggle_Comp_Int_Values == 4)
        {
                PORTA.5 = 1;
                PORTF = (0b00100000 | PORTF);
        }
        else if (Toggle_Comp_Int_Values == 6)
        {
                PORTF = (0b00001000 | PORTF);
        }
}

// Timer 1 output compare B interrupt service routine
```

```
interrupt [TIM1_COMPB] void timer1_compb_isr(void)
{

        // Place your code here
   if (Toggle_Comp_Int_Values == 2)
        {
               PORTA.6 = 1;
               PORTF = (0b01000000 | PORTF);
        }
        else if (Toggle_Comp_Int_Values == 4)
        {
               PORTA.4 = 1;
               PORTF = (0b00010000 | PORTF);
        }
        else if (Toggle_Comp_Int_Values == 6)
        {
               PORTF = (0b00000100 | PORTF);
        }
}

// Timer 1 output compare C interrupt service routine
interrupt [TIM1_COMPC] void timer1_compc_isr(void)
{
// Place your code here

}

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization - Data Direction 1 - output, 0 - intput
PORTA=0x00;
DDRA=0xff;

// Port B initialization
PORTB=0x00;
DDRB=0x00;

// Port C initialization
PORTC=0x00;
DDRC=0x00;

// Port D initialization
PORTD=0x00;
DDRD=0x00;

// Port E initialization
PORTE=0x00;
DDRE=0x00;

// Port F initialization
PORTF=0x00;
DDRF=0xff;

// Port G initialization
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
```

```
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x00;  //This enables the clock and sets the prescalling
       //TCCR1B=0x01;      //Enable Clock with no Pre-Scaling
       //Start = 1;

TCNT1H=0x00;
TCNT1L=0x00;

OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x1C; // Enable output compare interrupts and timer overflow for timer 1
ETIMSK=0x01;

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud rate: 9600
UCSR0A=0x00;
UCSR0B=0xD8;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x67;

// USART1 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
```

```
// USART1 Receiver: On
// USART1 Transmitter: On
// USART1 Mode: Asynchronous
// USART1 Baud rate: 9600
UCSR1A=0x00;
UCSR1B=0xD8;
UCSR1C=0x06;
UBRR1H=0x00;
UBRR1L=0x67;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
// Analog Comparator Output: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")


// *** Testing Functions ***

Comms_Received_Data ('r');


// ************************

while (1)
      {
      // Place your code here

      };
}

void Comms_Received_Data (char Data)
{
       // Local Variables
       unsigned int temp = 0;
       char i;

   if (Data == ' ')
   {
        //do nothing
   }
        else
        {
               switch(Data)
               {
                      // Motor ID's
                      case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':
                             Motor_ID = Data - 97; // Ascii number for character a is 97
                             Received_Number = 0;  //Reset motor value
                      break;


                      // Data digits
                      case '0': case '1': case '2': case '3': case '4': case '5': case '6':
case '7': case '8': case '9':
                             temp = Data - 48; // Ascii number for character 0 is 48
                      break;


                      // End Data character
                      case 'x':
                             New_Motor_Values[Motor_ID] = Received_Number;
                             temp = 0;  // Reset
                             Send_Data(Motor_ID + 97, New_Motor_Values[Motor_ID]); // the +97
shifts 0-5 to a-f
                             Received_Number = 0;  // Reset
                      break;


                      // Control Sgnals
                      case 'g':      // Start/Go
                             TCCR1B=0x01;      //Enable Clock with no Pre-Scaling
                             Start = 1;
```

```
                        putchar ('g');
                        putchar ('X');
                break;

                case 's':  // End, Stop
                        Start = 0; // So that we stop the timer in the timer overflow
routine
                        putchar ('s');
                        putchar ('X');
                break;

                case 'r':  // Send all data
                        for (i = 0; i<6; i++)
                        {
                                //i + 97, Letters a-f
                                Send_Data (i+97, New_Motor_Values[i]);
                        }

                break;

                default:
                break;
        }
        Received_Number = 10*Received_Number + temp;
    }
}


void Send_Data (unsigned char M_ID, unsigned int M_Value)
{
    // Local Variables
    char temp = 0;

    putchar (M_ID);
    if (M_Value >= 50000) {M_Value = M_Value - 50000; temp = temp + 5;}
    if (M_Value >= 30000) {M_Value = M_Value - 30000; temp = temp + 3;}
    if (M_Value >= 20000) {M_Value = M_Value - 20000; temp = temp + 2;}
    if (M_Value >= 10000) {M_Value = M_Value - 10000; temp = temp + 1;}
    putchar (temp + 48);

    temp = 0;
    if (M_Value >= 5000) {M_Value = M_Value - 5000; temp = temp + 5;}
    if (M_Value >= 3000) {M_Value = M_Value - 3000; temp = temp + 3;}
    if (M_Value >= 2000) {M_Value = M_Value - 2000; temp = temp + 2;}
    if (M_Value >= 1000) {M_Value = M_Value - 1000; temp = temp + 1;}
    putchar (temp + 48);

    temp = 0;
    if (M_Value >= 500) {M_Value = M_Value - 500; temp = temp + 5;}
    if (M_Value >= 300) {M_Value = M_Value - 300; temp = temp + 3;}
    if (M_Value >= 200) {M_Value = M_Value - 200; temp = temp + 2;}
    if (M_Value >= 100) {M_Value = M_Value - 100; temp = temp + 1;}
    putchar (temp + 48);

    temp = 0;
    if (M_Value >= 50) {M_Value = M_Value - 50; temp = temp + 5;}
    if (M_Value >= 30) {M_Value = M_Value - 30; temp = temp + 3;}
    if (M_Value >= 20) {M_Value = M_Value - 20; temp = temp + 2;}
    if (M_Value >= 10) {M_Value = M_Value - 10; temp = temp + 1;}
    putchar (temp + 48);

    temp = 0;
    if (M_Value >= 5) {M_Value = M_Value - 5; temp = temp + 5;}
    if (M_Value >= 3) {M_Value = M_Value - 3; temp = temp + 3;}
    if (M_Value >= 2) {M_Value = M_Value - 2; temp = temp + 2;}
    if (M_Value >= 1) {M_Value = M_Value - 1; temp = temp + 1;}
    putchar (temp + 48);

    putchar ('X');

}

unsigned int Inc_Timr_Vals (unsigned int CMV, unsigned int NMV)
{
    int Temp;
    Temp = CMV - NMV;
```

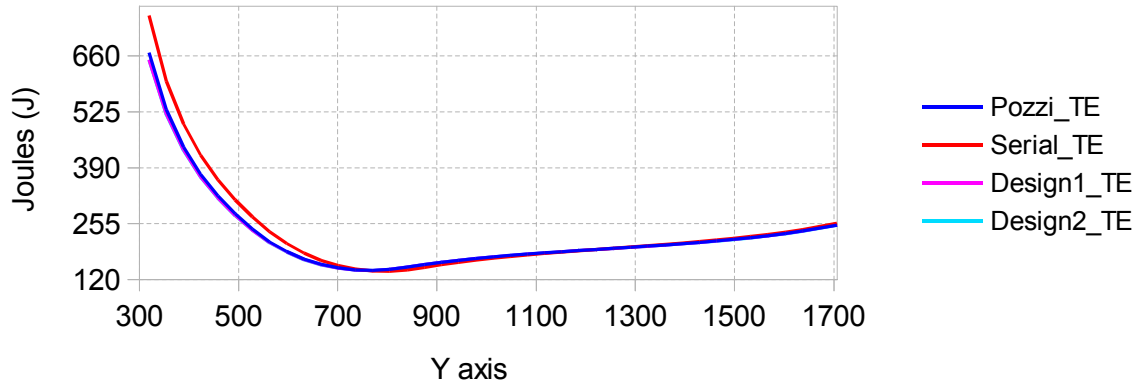
```
        if (Temp == 0)
        {
                // Do nothing
        }
        else
        {
                if (abs(Temp) > TIMER_INC)
                {
                        if (Temp > TIMER_INC) {CMV = CMV - TIMER_INC;}
                        else {CMV = CMV + TIMER_INC;}
                }
                else {CMV = NMV;}
        }
        return CMV;
}
```

APPENDIX C: Simulation data tables and graphs

Multi-horizontal line paths, light motors, 5s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	667.66	757.75	650.45	651.08
354.54	529.95	598.89	519.86	518.54
389.2	439.96	495.64	432.55	431.5
423.85	373.98	420.36	368.29	367.42
458.51	321.93	361.27	317.43	316.7
493.16	278.9	312.61	275.24	274.63
527.82	242.07	271.13	239.03	238.53
562.47	211.39	235.84	209.51	208.99
597.12	188.15	207.2	186.82	186.26
631.78	169.6	183.65	168.45	167.91
666.43	156.99	166.41	156.18	155.7
701.09	148.18	153.69	147.68	147.26
735.74	143.26	145.13	142.97	142.64
770.4	142.02	140.69	141.89	141.65
805.05	144.24	140.13	144.25	144.17
839.71	149.58	143.26	149.65	149.74
874.36	155.89	149.46	155.93	156.02
909.02	161.47	155.95	161.47	161.55
943.67	166.4	161.71	166.37	166.45
978.32	170.79	166.87	170.74	170.81
1012.98	174.72	171.5	174.64	174.7
1047.63	178.28	175.7	178.17	178.23
1082.29	181.53	179.54	181.4	181.45
1116.94	184.53	183.09	184.39	184.43
1151.6	187.36	186.42	187.21	187.23
1186.25	190.07	189.59	189.91	189.91
1220.91	192.71	192.66	192.53	192.52
1255.56	195.32	195.67	195.14	195.11
1290.22	197.95	198.68	197.77	197.72
1324.87	200.66	201.74	200.48	200.41
1359.52	203.48	204.9	203.31	203.21
1394.18	206.46	208.2	206.3	206.17
1428.83	209.66	211.71	209.51	209.34
1463.49	213.12	215.47	212.99	212.78
1498.14	216.91	219.55	216.8	216.55

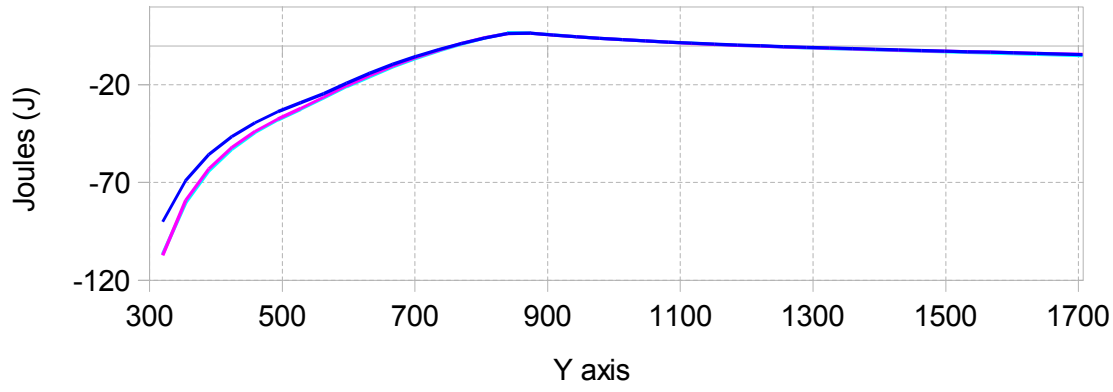
a. Total energy consumed per model



Multi-horizontal line paths, light motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-90.09	-107.3	-106.67
354.54	-68.94	-79.04	-80.35
389.2	-55.68	-63.09	-64.14
423.85	-46.38	-52.07	-52.94
458.51	-39.34	-43.84	-44.57
493.16	-33.71	-37.37	-37.98
527.82	-29.06	-32.1	-32.6
562.47	-24.44	-26.33	-26.85
597.12	-19.05	-20.39	-20.94
631.78	-14.05	-15.21	-15.74
666.43	-9.42	-10.23	-10.71
701.09	-5.52	-6.01	-6.43
735.74	-1.87	-2.15	-2.48
770.4	1.33	1.21	0.96
805.05	4.11	4.13	4.04
839.71	6.32	6.39	6.48
874.36	6.43	6.47	6.55
909.02	5.52	5.53	5.61
943.67	4.69	4.66	4.74
978.32	3.92	3.87	3.94
1012.98	3.22	3.14	3.2
1047.63	2.58	2.48	2.53
1082.29	1.99	1.87	1.91
1116.94	1.45	1.31	1.34
1151.6	0.94	0.79	0.81
1186.25	0.48	0.31	0.32
1220.91	0.05	-0.13	-0.13
1255.56	-0.35	-0.53	-0.56
1290.22	-0.73	-0.91	-0.96
1324.87	-1.08	-1.26	-1.34
1359.52	-1.42	-1.59	-1.69
1394.18	-1.74	-1.9	-2.03
1428.83	-2.05	-2.2	-2.36
1463.49	-2.35	-2.48	-2.68
1498.14	-2.64	-2.75	-3

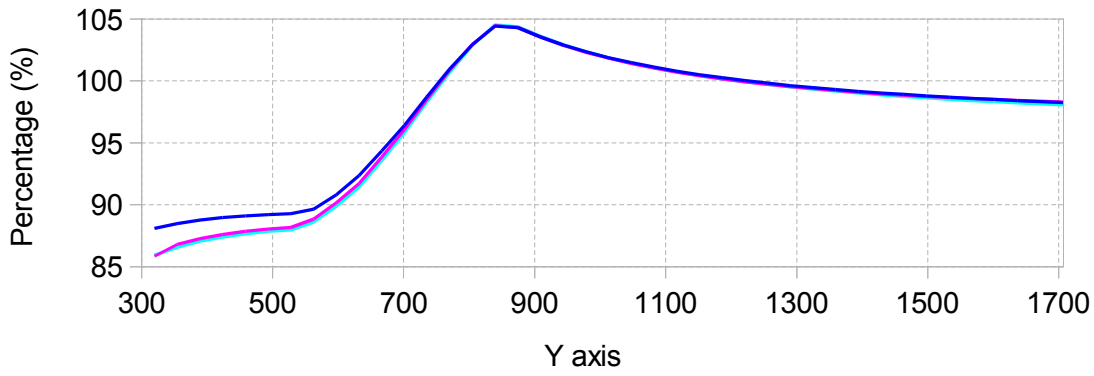
b. Difference from serial model



Multi-horizontal line paths, light motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	88.11	85.84	85.92
354.54	88.49	86.8	86.58
389.2	88.77	87.27	87.06
423.85	88.97	87.61	87.41
458.51	89.11	87.86	87.66
493.16	89.22	88.05	87.85
527.82	89.28	88.16	87.98
562.47	89.64	88.84	88.62
597.12	90.81	90.16	89.89
631.78	92.35	91.72	91.43
666.43	94.34	93.85	93.57
701.09	96.41	96.09	95.82
735.74	98.71	98.52	98.29
770.4	100.95	100.86	100.69
805.05	102.93	102.94	102.88
839.71	104.41	104.46	104.52
874.36	104.3	104.33	104.39
909.02	103.54	103.54	103.6
943.67	102.9	102.88	102.93
978.32	102.35	102.32	102.36
1012.98	101.88	101.83	101.87
1047.63	101.47	101.41	101.44
1082.29	101.11	101.04	101.06
1116.94	100.79	100.71	100.73
1151.6	100.51	100.42	100.43
1186.25	100.25	100.17	100.17
1220.91	100.03	99.94	99.93
1255.56	99.82	99.73	99.71
1290.22	99.63	99.54	99.52
1324.87	99.46	99.37	99.34
1359.52	99.31	99.22	99.17
1394.18	99.16	99.09	99.02
1428.83	99.03	98.96	98.88
1463.49	98.91	98.85	98.75
1498.14	98.8	98.75	98.64

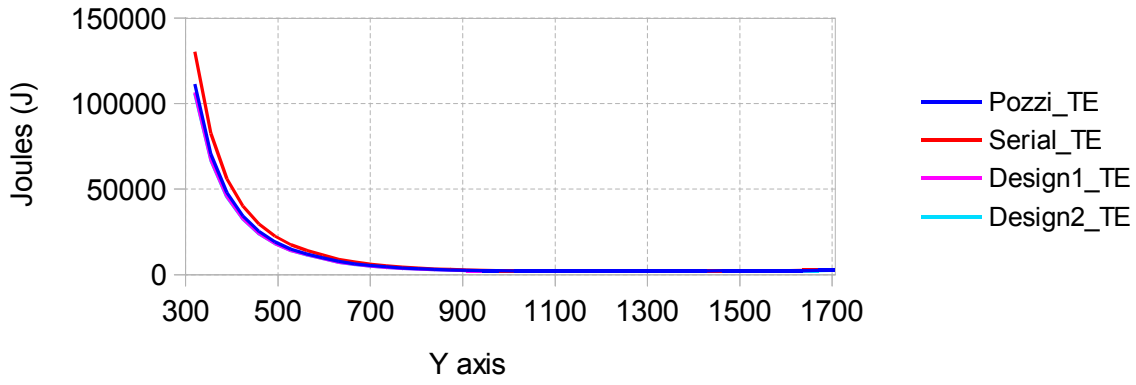
c. As a percentage of serial model



Multi-horizontal line paths, light motors, 0.25s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	111491.53	130452.68	106540.77	105808.7
354.54	70505.8	82569.93	67317.74	66843.45
389.2	47937.29	56160.15	45731.96	45401.64
423.85	34239.46	40118.36	32615.01	32383.86
458.51	25321.7	29663.52	24106.46	23937.01
493.16	19226.75	22520.25	18294.62	18171.36
527.82	14882.7	17416.19	14140.67	14051.11
562.47	12040.5	14172.32	11637.36	11570.46
597.12	9854.42	11557.32	9536.34	9486.11
631.78	7658.77	8884.61	7255.34	7222.12
666.43	6322.16	7297.16	5988.09	5964.02
701.09	5293.75	6054.09	4995.75	4978.78
735.74	4508.65	5098.67	4240.19	4227.88
770.4	3899.98	4361.64	3662.57	3654.38
805.05	3426.66	3789.86	3218.68	3213.32
839.71	3059.36	3346.49	2877.74	2874.51
874.36	2776.93	3003.84	2616.78	2615.07
909.02	2560.49	2739.44	2418.57	2418.03
943.67	2397.07	2536.72	2270.18	2270.06
978.32	2274.73	2383.1	2161.97	2162.9
1012.98	2185.8	2269.92	2085.23	2086.5
1047.63	2123.26	2188.65	2032.54	2034.19
1082.29	2081.64	2132.66	1999.82	2001.61
1116.94	2056.8	2096.29	1982.78	1985.05
1151.6	2045.68	2076.45	1978.22	1980.5
1186.25	2044.71	2069.35	1982.98	1985.07
1220.91	2052.28	2072.6	1995.68	1997.71
1255.56	2067.05	2084.33	2014.84	2016.52
1290.22	2086.78	2102.13	2038.44	2039.62
1324.87	2110.65	2125.13	2065.85	2066.59
1359.52	2138.11	2152.63	2096.1	2096.52
1394.18	2168.05	2183.16	2128.54	2128.47
1428.83	2199.82	2216.2	2162.85	2162.19
1463.49	2233.54	2251.64	2198.51	2197.39
1498.14	2268.54	2288.63	2235.28	2233.57

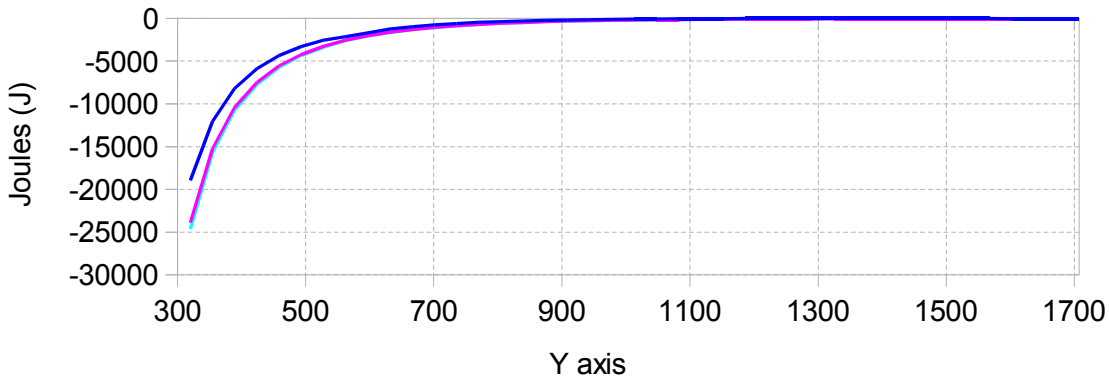
a. Total energy consumed per model



Multi-horizontal line paths, light motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-18961.15	-23911.91	-24643.98
354.54	-12064.13	-15252.19	-15726.48
389.2	-8222.86	-10428.19	-10758.51
423.85	-5878.91	-7503.35	-7734.51
458.51	-4341.82	-5557.06	-5726.51
493.16	-3293.51	-4225.63	-4348.89
527.82	-2533.49	-3275.52	-3365.08
562.47	-2131.82	-2534.96	-2601.85
597.12	-1702.89	-2020.97	-2071.2
631.78	-1225.85	-1629.27	-1662.49
666.43	-975	-1309.07	-1333.14
701.09	-760.34	-1058.34	-1075.31
735.74	-590.02	-858.48	-870.79
770.4	-461.67	-699.08	-707.26
805.05	-363.2	-571.18	-576.54
839.71	-287.13	-468.75	-471.99
874.36	-226.9	-387.06	-388.77
909.02	-178.95	-320.87	-321.4
943.67	-139.65	-266.54	-266.67
978.32	-108.37	-221.12	-220.19
1012.98	-84.12	-184.7	-183.43
1047.63	-65.4	-156.12	-154.46
1082.29	-51.02	-132.84	-131.05
1116.94	-39.49	-113.51	-111.25
1151.6	-30.77	-98.24	-95.95
1186.25	-24.64	-86.37	-84.28
1220.91	-20.32	-76.92	-74.89
1255.56	-17.28	-69.49	-67.81
1290.22	-15.34	-63.69	-62.51
1324.87	-14.48	-59.28	-58.54
1359.52	-14.52	-56.53	-56.12
1394.18	-15.12	-54.62	-54.69
1428.83	-16.39	-53.35	-54.01
1463.49	-18.1	-53.13	-54.26
1498.14	-20.1	-53.35	-55.06

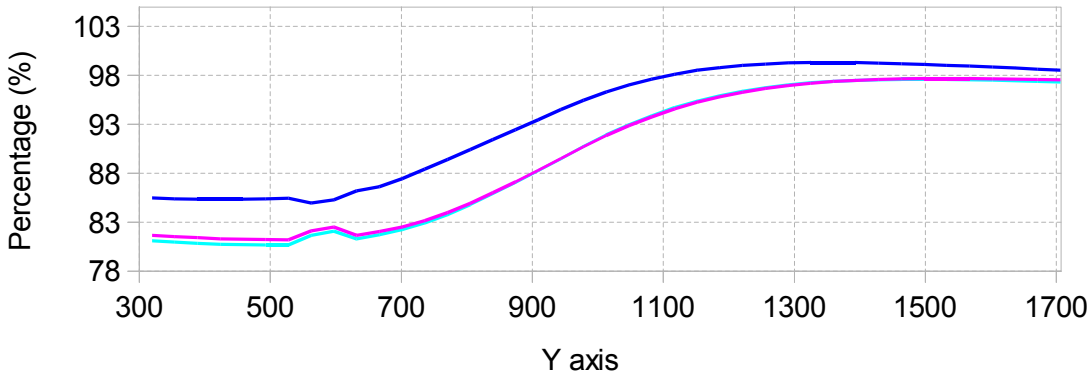
b. Difference from serial model



Multi-horizontal line paths, light motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	85.47	81.67	81.11
354.54	85.39	81.53	80.95
389.2	85.36	81.43	80.84
423.85	85.35	81.3	80.72
458.51	85.36	81.27	80.7
493.16	85.38	81.24	80.69
527.82	85.45	81.19	80.68
562.47	84.96	82.11	81.64
597.12	85.27	82.51	82.08
631.78	86.2	81.66	81.29
666.43	86.64	82.06	81.73
701.09	87.44	82.52	82.24
735.74	88.43	83.16	82.92
770.4	89.42	83.97	83.78
805.05	90.42	84.93	84.79
839.71	91.42	85.99	85.9
874.36	92.45	87.11	87.06
909.02	93.47	88.29	88.27
943.67	94.49	89.49	89.49
978.32	95.45	90.72	90.76
1012.98	96.29	91.86	91.92
1047.63	97.01	92.87	92.94
1082.29	97.61	93.77	93.86
1116.94	98.12	94.59	94.69
1151.6	98.52	95.27	95.38
1186.25	98.81	95.83	95.93
1220.91	99.02	96.29	96.39
1255.56	99.17	96.67	96.75
1290.22	99.27	96.97	97.03
1324.87	99.32	97.21	97.25
1359.52	99.33	97.37	97.39
1394.18	99.31	97.5	97.49
1428.83	99.26	97.59	97.56
1463.49	99.2	97.64	97.59
1498.14	99.12	97.67	97.59

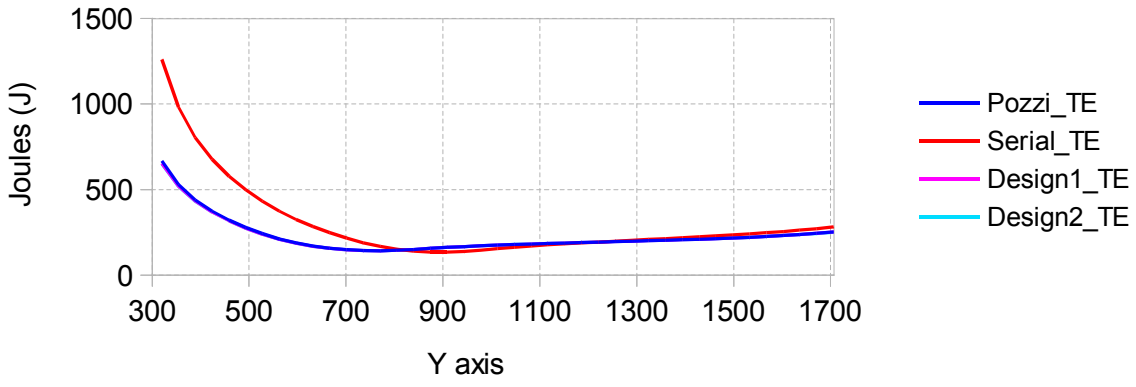
c. As a percentage of serial model



Multi-horizontal line paths, heavy motors, 5s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	667.66	1260.81	652.85	651.08
354.54	529.95	982.32	519.86	518.54
389.2	439.96	804.58	432.55	431.5
423.85	373.98	677.2	368.29	367.42
458.51	321.93	578.76	317.43	316.7
493.16	278.9	498.78	275.24	274.63
527.82	242.07	431.5	239.03	238.53
562.47	211.39	373	209.51	208.99
597.12	188.15	325.46	186.82	186.26
631.78	169.6	284.72	168.45	167.91
666.43	156.99	249.17	156.18	155.7
701.09	148.18	217.43	147.68	147.26
735.74	143.26	189.18	142.97	142.64
770.4	142.02	166.73	141.89	141.65
805.05	144.24	150.21	144.25	144.17
839.71	149.58	139.36	149.65	149.74
874.36	155.89	133.89	155.93	156.02
909.02	161.47	133.58	161.47	161.55
943.67	166.4	138.03	166.37	166.45
978.32	170.79	146.32	170.74	170.81
1012.98	174.72	154.77	174.64	174.7
1047.63	178.28	162.46	178.17	178.23
1082.29	181.53	169.51	181.4	181.45
1116.94	184.53	176.02	184.39	184.43
1151.6	187.36	182.09	187.21	187.23
1186.25	190.07	187.78	189.91	189.91
1220.91	192.71	193.2	192.53	192.52
1255.56	195.32	198.4	195.14	195.11
1290.22	197.95	203.46	197.77	197.72
1324.87	200.66	208.46	200.48	200.41
1359.52	203.48	213.45	203.31	203.21
1394.18	206.46	218.5	206.3	206.17
1428.83	209.66	223.69	209.51	209.34
1463.49	213.12	229.09	212.99	212.78

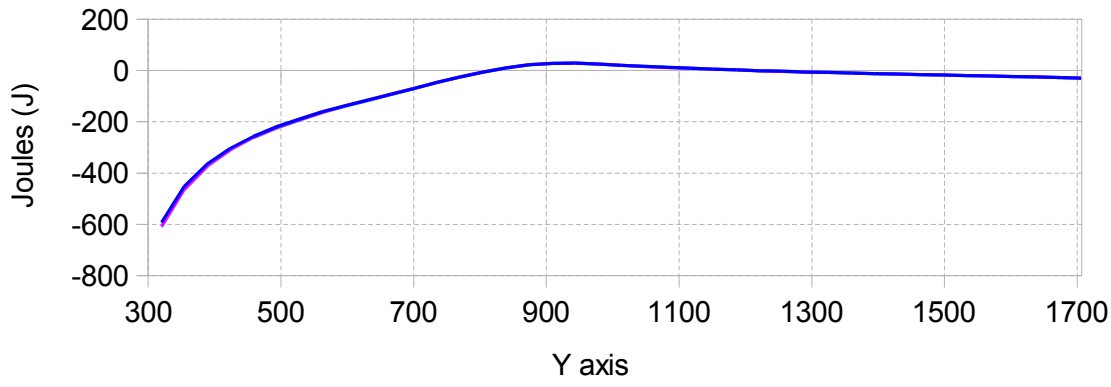
a. Total energy consumed per model



Multi-horizontal line paths, heavy motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-593.15	-607.96	-609.73
354.54	-452.37	-462.46	-463.78
389.2	-364.61	-372.02	-373.08
423.85	-303.22	-308.91	-309.78
458.51	-256.83	-261.33	-262.06
493.16	-219.88	-223.54	-224.15
527.82	-189.44	-192.47	-192.97
562.47	-161.6	-163.49	-164.01
597.12	-137.3	-138.64	-139.2
631.78	-115.12	-116.27	-116.81
666.43	-92.18	-92.99	-93.46
701.09	-69.26	-69.75	-70.17
735.74	-45.92	-46.2	-46.54
770.4	-24.71	-24.84	-25.08
805.05	-5.97	-5.95	-6.04
839.71	10.23	10.29	10.38
874.36	22.01	22.04	22.13
909.02	27.89	27.9	27.98
943.67	28.37	28.35	28.42
978.32	24.47	24.41	24.48
1012.98	19.96	19.88	19.94
1047.63	15.82	15.71	15.77
1082.29	12.01	11.89	11.93
1116.94	8.51	8.37	8.4
1151.6	5.28	5.12	5.15
1186.25	2.29	2.12	2.13
1220.91	-0.49	-0.66	-0.67
1255.56	-3.08	-3.26	-3.29
1290.22	-5.51	-5.69	-5.74
1324.87	-7.8	-7.98	-8.05
1359.52	-9.97	-10.14	-10.24
1394.18	-12.04	-12.2	-12.33
1428.83	-14.03	-14.18	-14.35
1463.49	-15.97	-16.1	-16.3

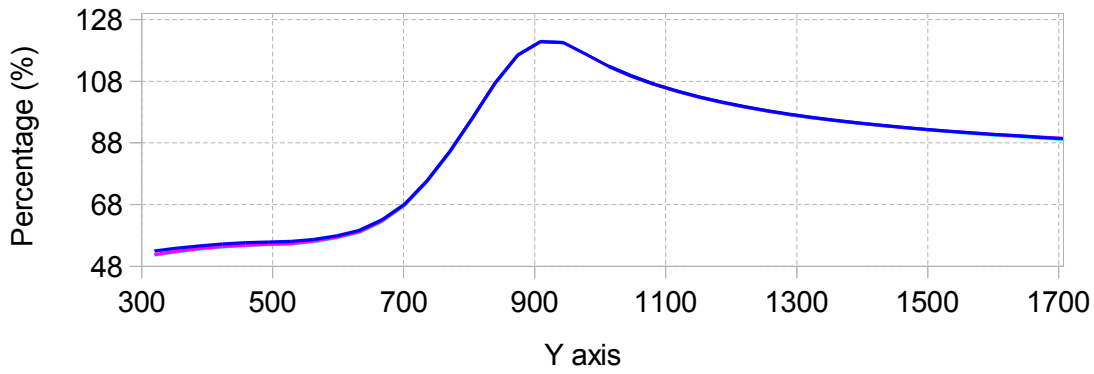
b. Difference from serial model



Multi-horizontal line paths, heavy motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	52.96	51.78	51.64
354.54	53.95	52.92	52.79
389.2	54.68	53.76	53.63
423.85	55.22	54.38	54.26
458.51	55.62	54.85	54.72
493.16	55.92	55.18	55.06
527.82	56.1	55.39	55.28
562.47	56.67	56.17	56.03
597.12	57.81	57.4	57.23
631.78	59.57	59.16	58.97
666.43	63.01	62.68	62.49
701.09	68.15	67.92	67.73
735.74	75.73	75.58	75.4
770.4	85.18	85.1	84.96
805.05	96.03	96.04	95.98
839.71	107.34	107.39	107.45
874.36	116.44	116.46	116.53
909.02	120.88	120.88	120.95
943.67	120.56	120.54	120.59
978.32	116.72	116.68	116.73
1012.98	112.89	112.84	112.88
1047.63	109.73	109.67	109.7
1082.29	107.09	107.01	107.04
1116.94	104.83	104.75	104.77
1151.6	102.9	102.81	102.83
1186.25	101.22	101.13	101.13
1220.91	99.75	99.66	99.65
1255.56	98.45	98.36	98.34
1290.22	97.29	97.2	97.18
1324.87	96.26	96.17	96.14
1359.52	95.33	95.25	95.2
1394.18	94.49	94.42	94.36
1428.83	93.73	93.66	93.59
1463.49	93.03	92.97	92.88

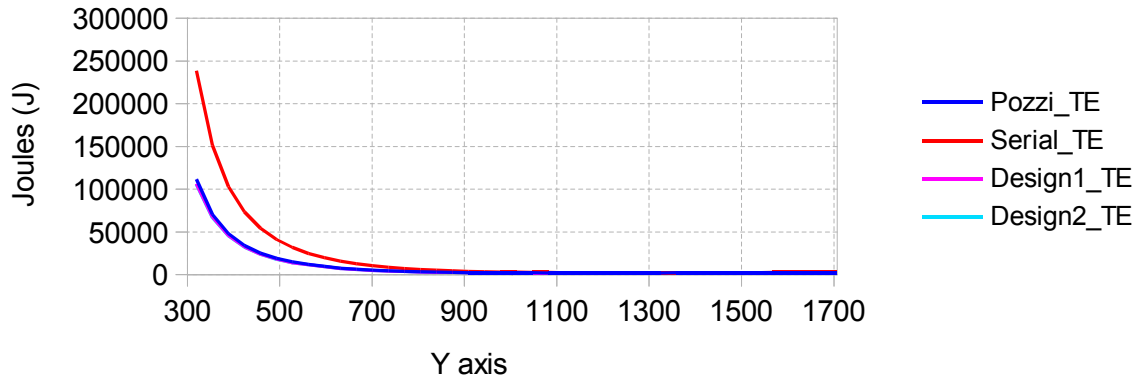
c. As a percentage of serial model



Multi-horizontal line paths, heavy motors, 0.25s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	111491.53	238720.33	106540.77	105808.7
354.54	70505.8	151300.48	67317.74	66843.45
389.2	47937.29	103009.69	45731.96	45401.64
423.85	34239.46	73585.92	32615.01	32383.86
458.51	25321.7	54366.94	24106.46	23937.01
493.16	19226.75	41169.57	18294.62	18171.36
527.82	14882.7	31753.18	14140.67	14051.11
562.47	12040.5	24735.32	11637.36	11570.46
597.12	9854.42	19870.55	9536.34	9486.11
631.78	7658.77	15834.47	7255.34	7222.12
666.43	6322.16	12834.42	5988.09	5964.02
701.09	5293.75	10485.51	4995.75	4978.78
735.74	4508.65	8658.72	4240.19	4227.88
770.4	3899.98	7230.74	3662.57	3654.38
805.05	3426.66	6108.08	3218.68	3213.32
839.71	3059.36	5223.23	2877.74	2874.51
874.36	2776.93	4525.4	2616.78	2615.07
909.02	2560.49	3975.54	2418.57	2418.03
943.67	2397.07	3543.24	2270.18	2270.06
978.32	2274.73	3206.92	2161.97	2162.9
1012.98	2185.8	2946.28	2085.23	2086.5
1047.63	2123.26	2746.2	2032.54	2034.19
1082.29	2081.64	2596.38	1999.82	2001.61
1116.94	2056.8	2485.91	1982.78	1985.05
1151.6	2045.68	2408.39	1978.22	1980.5
1186.25	2044.71	2356.48	1982.98	1985.07
1220.91	2052.28	2325.58	1995.68	1997.71
1255.56	2067.05	2312.29	2014.84	2016.52
1290.22	2086.78	2313.24	2038.44	2039.62
1324.87	2110.65	2324.9	2065.85	2066.59
1359.52	2138.11	2345.29	2096.1	2096.52
1394.18	2168.05	2373.84	2128.54	2128.47
1428.83	2199.82	2408.45	2162.85	2162.19
1463.49	2233.54	2447.62	2198.51	2197.39
1498.14	2268.54	2490.78	2235.28	2233.57

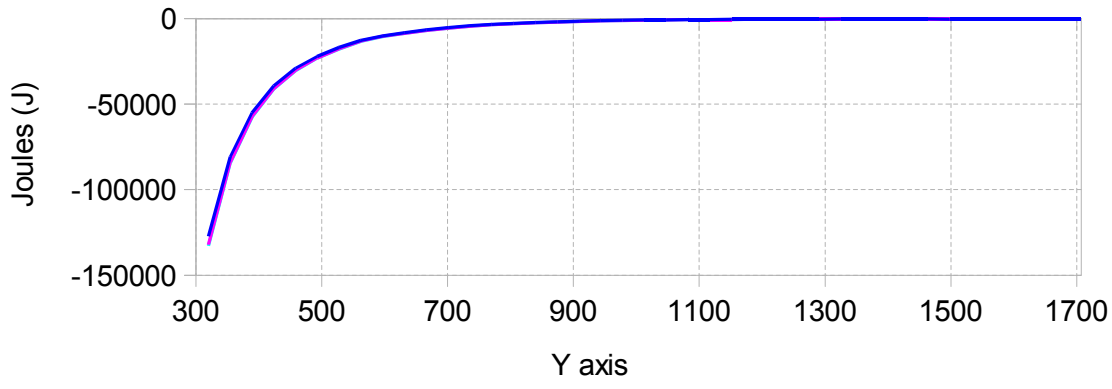
a. Total energy consumed per model



Multi-horizontal line paths, heavy motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-127228.8	-132179.56	-132911.63
354.54	-80794.68	-83982.74	-84457.03
389.2	-55072.4	-57277.73	-57608.05
423.85	-39346.46	-40970.9	-41202.06
458.51	-29045.23	-30260.47	-30429.92
493.16	-21942.83	-22874.95	-22998.21
527.82	-16870.48	-17612.5	-17702.07
562.47	-12694.83	-13097.97	-13164.86
597.12	-10016.13	-10334.21	-10384.44
631.78	-8175.7	-8579.13	-8612.34
666.43	-6512.27	-6846.34	-6870.41
701.09	-5191.76	-5489.77	-5506.73
735.74	-4150.07	-4418.53	-4430.84
770.4	-3330.76	-3568.17	-3576.36
805.05	-2681.41	-2889.4	-2894.75
839.71	-2163.86	-2345.49	-2348.72
874.36	-1748.47	-1908.62	-1910.34
909.02	-1415.05	-1556.97	-1557.5
943.67	-1146.17	-1273.06	-1273.18
978.32	-932.19	-1044.94	-1044.01
1012.98	-760.48	-861.05	-859.79
1047.63	-622.95	-713.66	-712.01
1082.29	-514.74	-596.56	-594.77
1116.94	-429.11	-503.13	-500.87
1151.6	-362.71	-430.17	-427.89
1186.25	-311.77	-373.5	-371.41
1220.91	-273.3	-329.89	-327.87
1255.56	-245.24	-297.45	-295.77
1290.22	-226.46	-274.8	-273.62
1324.87	-214.26	-259.05	-258.31
1359.52	-207.18	-249.19	-248.77
1394.18	-205.79	-245.29	-245.36
1428.83	-208.63	-245.6	-246.26
1463.49	-214.08	-249.11	-250.23
1498.14	-222.24	-255.49	-257.21

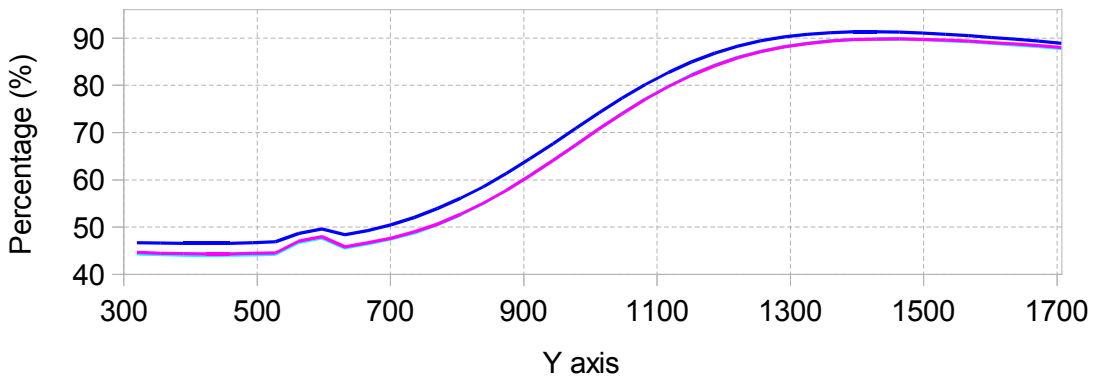
b. Difference from serial model



Multi-horizontal line paths, heavy motors, 0.25s

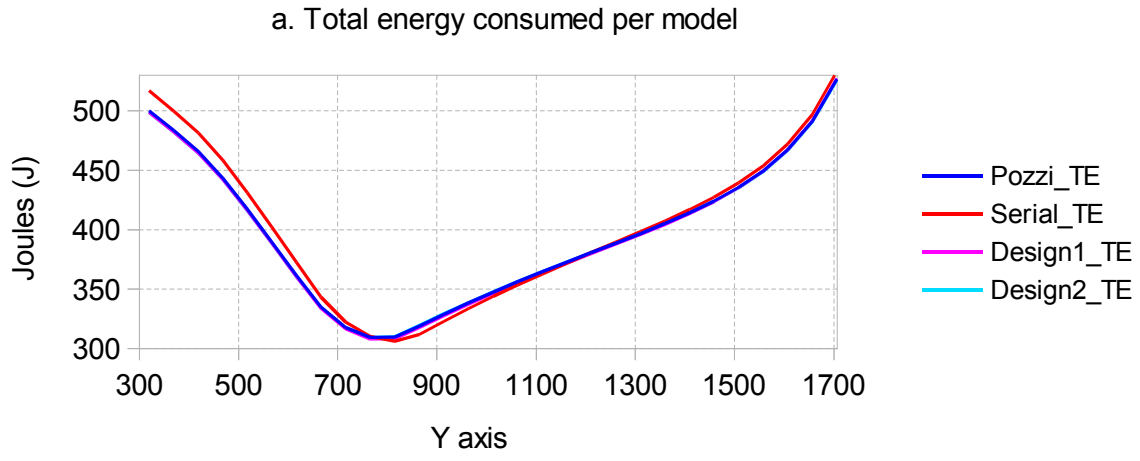
Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	46.7	44.63	44.32
354.54	46.6	44.49	44.18
389.2	46.54	44.4	44.08
423.85	46.53	44.32	44.01
458.51	46.58	44.34	44.03
493.16	46.7	44.44	44.14
527.82	46.87	44.53	44.25
562.47	48.68	47.05	46.78
597.12	49.59	47.99	47.74
631.78	48.37	45.82	45.61
666.43	49.26	46.66	46.47
701.09	50.49	47.64	47.48
735.74	52.07	48.97	48.83
770.4	53.94	50.65	50.54
805.05	56.1	52.7	52.61
839.71	58.57	55.1	55.03
874.36	61.36	57.82	57.79
909.02	64.41	60.84	60.82
943.67	67.65	64.07	64.07
978.32	70.93	67.42	67.44
1012.98	74.19	70.77	70.82
1047.63	77.32	74.01	74.07
1082.29	80.17	77.02	77.09
1116.94	82.74	79.76	79.85
1151.6	84.94	82.14	82.23
1186.25	86.77	84.15	84.24
1220.91	88.25	85.81	85.9
1255.56	89.39	87.14	87.21
1290.22	90.21	88.12	88.17
1324.87	90.78	88.86	88.89
1359.52	91.17	89.37	89.39
1394.18	91.33	89.67	89.66
1428.83	91.34	89.8	89.78
1463.49	91.25	89.82	89.78
1498.14	91.08	89.74	89.67

c. As a percentage of serial model



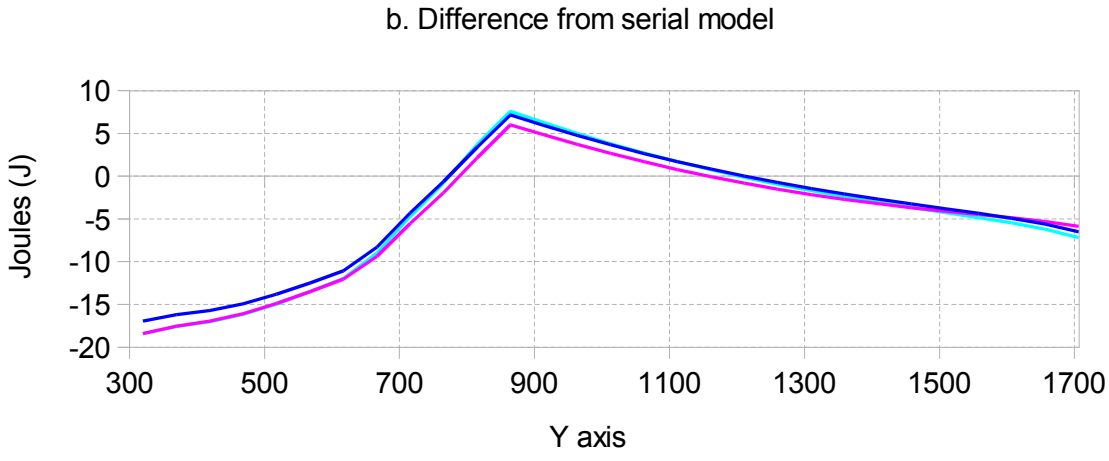
Multi-vertical line paths, light motors, 5s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	500.06	517.02	498.63	498.61
369.39	483.67	499.88	482.34	482.3
418.9	465.77	481.48	464.53	464.45
468.41	443.3	458.21	442.14	442.06
517.91	417.29	431.09	416.19	416.15
567.42	389.5	401.98	388.46	388.46
616.93	361.23	372.3	360.24	360.31
666.43	335.03	343.32	333.92	334.35
715.94	317.85	322.16	316.63	317.43
765.45	309.44	310.06	308.12	309.3
814.95	309.55	306.15	308.3	309.83
864.46	318.83	311.68	317.69	319.25
913.97	328.84	322.91	327.75	329.16
963.47	338.36	333.6	337.32	338.59
1012.98	347.45	343.78	346.44	347.59
1062.49	356.16	353.51	355.19	356.22
1111.99	364.56	362.85	363.63	364.55
1161.5	372.69	371.87	371.81	372.62
1211.01	380.68	380.67	379.85	380.54
1260.51	388.64	389.38	387.86	388.44
1310.02	396.74	398.17	396.03	396.48
1359.52	405.16	407.24	404.53	404.85
1409.03	414.16	416.84	413.62	413.8
1458.54	424.07	427.31	423.64	423.67
1508.04	435.63	439.43	435.34	435.18
1557.55	449.48	453.82	449.35	448.99
1607.06	467.22	472.15	467.29	466.68
1656.56	491.25	496.85	491.56	490.65
1706.07	526.59	533.09	527.21	525.91



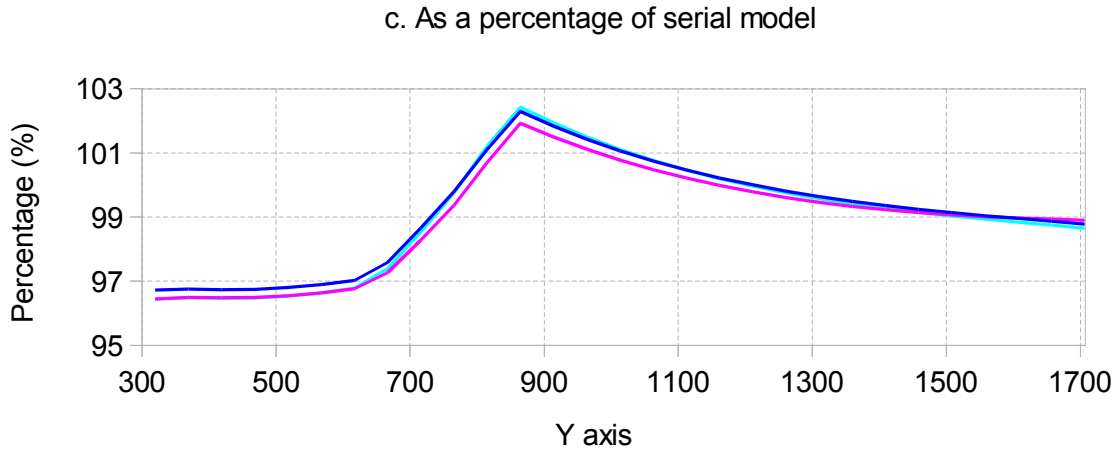
Multi-vertical line paths, light motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-16.96	-18.39	-18.41
369.39	-16.21	-17.54	-17.59
418.9	-15.71	-16.94	-17.03
468.41	-14.91	-16.07	-16.16
517.91	-13.79	-14.89	-14.94
567.42	-12.48	-13.52	-13.51
616.93	-11.07	-12.05	-11.98
666.43	-8.3	-9.4	-8.97
715.94	-4.31	-5.53	-4.73
765.45	-0.62	-1.94	-0.76
814.95	3.4	2.15	3.68
864.46	7.14	6	7.57
913.97	5.93	4.84	6.25
963.47	4.76	3.71	4.99
1012.98	3.67	2.66	3.81
1062.49	2.65	1.69	2.72
1111.99	1.7	0.78	1.69
1161.5	0.83	-0.05	0.75
1211.01	0.01	-0.82	-0.13
1260.51	-0.74	-1.52	-0.94
1310.02	-1.43	-2.15	-1.69
1359.52	-2.08	-2.71	-2.39
1409.03	-2.68	-3.22	-3.04
1458.54	-3.25	-3.68	-3.65
1508.04	-3.8	-4.09	-4.24
1557.55	-4.34	-4.47	-4.83
1607.06	-4.92	-4.86	-5.46
1656.56	-5.6	-5.29	-6.2
1706.07	-6.51	-5.88	-7.18



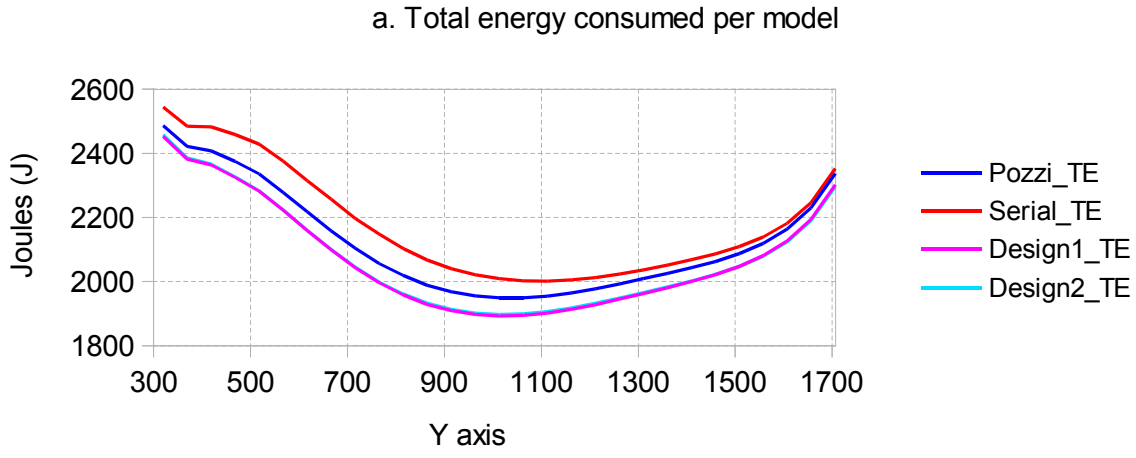
Multi-vertical line paths, light motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	96.72	96.44	96.44
369.39	96.76	96.49	96.48
418.9	96.74	96.48	96.46
468.41	96.75	96.49	96.47
517.91	96.8	96.55	96.53
567.42	96.9	96.64	96.64
616.93	97.03	96.76	96.78
666.43	97.58	97.26	97.39
715.94	98.66	98.28	98.53
765.45	99.8	99.37	99.76
814.95	101.11	100.7	101.2
864.46	102.29	101.93	102.43
913.97	101.84	101.5	101.94
963.47	101.43	101.11	101.5
1012.98	101.07	100.77	101.11
1062.49	100.75	100.48	100.77
1111.99	100.47	100.22	100.47
1161.5	100.22	99.99	100.2
1211.01	100	99.78	99.97
1260.51	99.81	99.61	99.76
1310.02	99.64	99.46	99.58
1359.52	99.49	99.33	99.41
1409.03	99.36	99.23	99.27
1458.54	99.24	99.14	99.15
1508.04	99.14	99.07	99.03
1557.55	99.04	99.01	98.94
1607.06	98.96	98.97	98.84
1656.56	98.87	98.93	98.75
1706.07	98.78	98.9	98.65



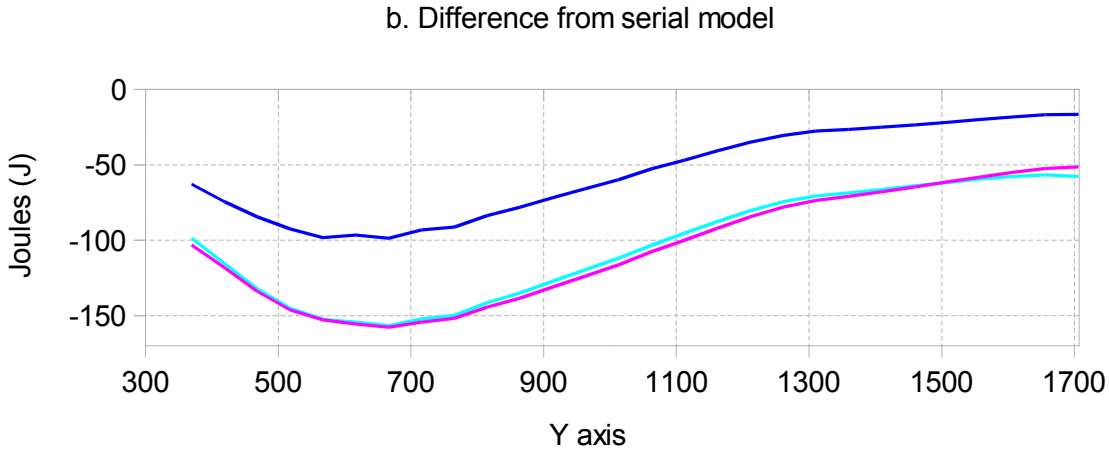
Multi-vertical line paths, light motors, 0.25s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	2486.33	2543.87	2452.11	2457.22
369.39	2421.42	2484.2	2381.05	2385.58
418.9	2407.43	2481.89	2363.61	2366.38
468.41	2374.16	2458.75	2324.83	2326.57
517.91	2335.27	2427.83	2281.65	2282.78
567.42	2277.14	2375.34	2222.44	2222.97
616.93	2217.67	2314.27	2158.74	2159.99
666.43	2157.93	2256.69	2099.02	2100.01
715.94	2103.76	2197	2042.7	2044.79
765.45	2056.32	2147.61	1995.84	1997.8
814.95	2018.72	2102.38	1958	1960.95
864.46	1988.67	2066.83	1928.51	1931.85
913.97	1968.48	2040.13	1909.3	1912.94
963.47	1954.89	2020.71	1896.93	1901.21
1012.98	1949	2008.8	1892.47	1896.85
1062.49	1949.42	2002.14	1894.45	1898.83
1111.99	1954.53	2001.36	1901.45	1906.03
1161.5	1964.09	2004.85	1912.76	1917.14
1211.01	1977.1	2012.18	1927.58	1931.65
1260.51	1992.49	2022.96	1944.82	1948.48
1310.02	2008.55	2036.18	1962.47	1965.5
1359.52	2024.94	2051.4	1980.47	1982.8
1409.03	2042.47	2067.55	1999.64	2001.2
1458.54	2062.2	2085.67	2021	2021.7
1508.04	2087.25	2109.02	2047.61	2047.33
1557.55	2119	2138.97	2080.88	2079.48
1607.06	2163.59	2181.82	2126.84	2124.12
1656.56	2229.36	2246.15	2193.76	2189.46
1706.07	2336.42	2352.92	2301.46	2295.17



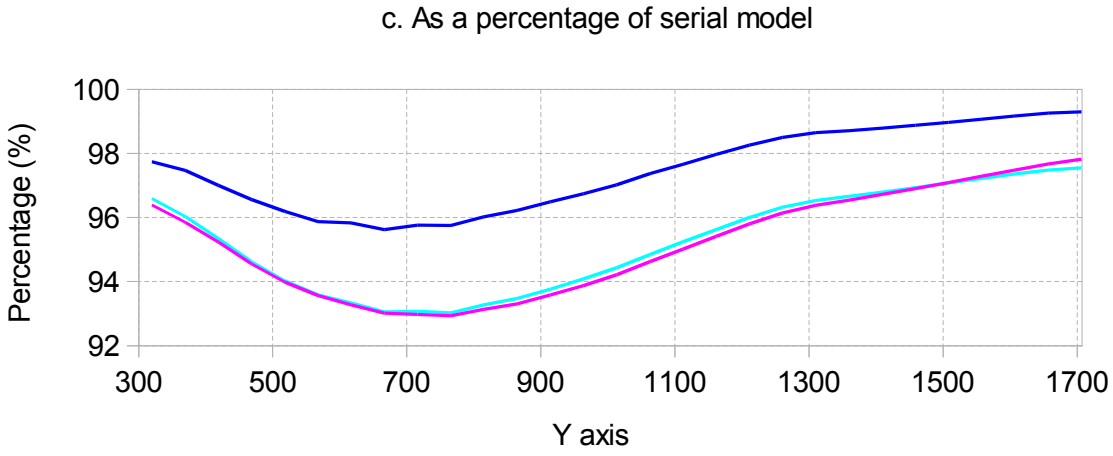
Multi-vertical line paths, light motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-57.54	-91.76	-86.65
369.39	-62.78	-103.15	-98.62
418.9	-74.46	-118.28	-115.51
468.41	-84.59	-133.92	-132.18
517.91	-92.56	-146.18	-145.06
567.42	-98.19	-152.9	-152.37
616.93	-96.61	-155.53	-154.28
666.43	-98.75	-157.67	-156.67
715.94	-93.23	-154.3	-152.21
765.45	-91.29	-151.77	-149.8
814.95	-83.66	-144.37	-141.43
864.46	-78.16	-138.32	-134.97
913.97	-71.65	-130.83	-127.19
963.47	-65.82	-123.78	-119.5
1012.98	-59.81	-116.33	-111.95
1062.49	-52.73	-107.7	-103.31
1111.99	-46.83	-99.91	-95.33
1161.5	-40.77	-92.09	-87.72
1211.01	-35.08	-84.6	-80.53
1260.51	-30.48	-78.15	-74.48
1310.02	-27.62	-73.71	-70.68
1359.52	-26.47	-70.94	-68.6
1409.03	-25.07	-67.91	-66.35
1458.54	-23.47	-64.67	-63.97
1508.04	-21.78	-61.41	-61.69
1557.55	-19.97	-58.08	-59.49
1607.06	-18.23	-54.97	-57.69
1656.56	-16.79	-52.39	-56.69
1706.07	-16.5	-51.46	-57.75



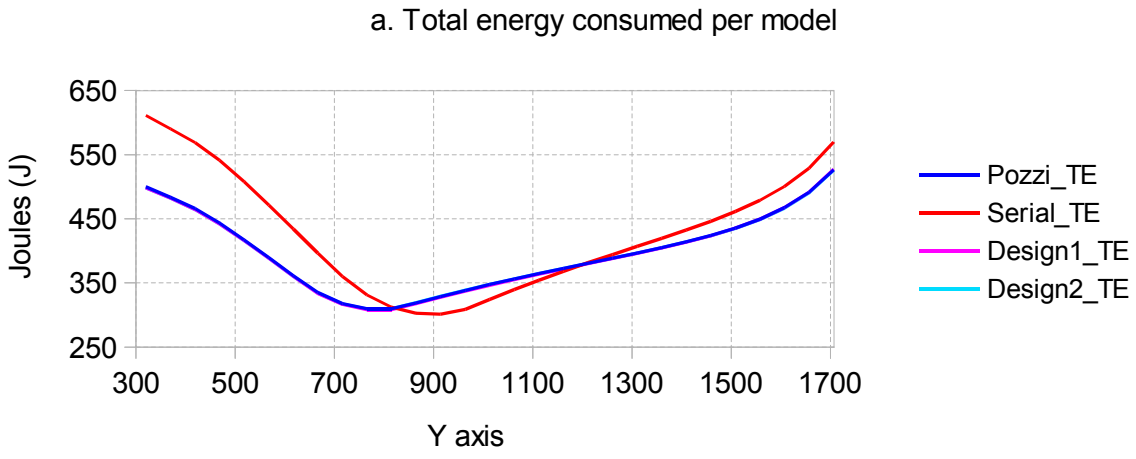
Multi-vertical line paths, light motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	97.74	96.39	96.59
369.39	97.47	95.85	96.03
418.9	97	95.23	95.35
468.41	96.56	94.55	94.62
517.91	96.19	93.98	94.03
567.42	95.87	93.56	93.59
616.93	95.83	93.28	93.33
666.43	95.62	93.01	93.06
715.94	95.76	92.98	93.07
765.45	95.75	92.93	93.02
814.95	96.02	93.13	93.27
864.46	96.22	93.31	93.47
913.97	96.49	93.59	93.77
963.47	96.74	93.87	94.09
1012.98	97.02	94.21	94.43
1062.49	97.37	94.62	94.84
1111.99	97.66	95.01	95.24
1161.5	97.97	95.41	95.62
1211.01	98.26	95.8	96
1260.51	98.49	96.14	96.32
1310.02	98.64	96.38	96.53
1359.52	98.71	96.54	96.66
1409.03	98.79	96.72	96.79
1458.54	98.87	96.9	96.93
1508.04	98.97	97.09	97.07
1557.55	99.07	97.28	97.22
1607.06	99.16	97.48	97.36
1656.56	99.25	97.67	97.48
1706.07	99.3	97.81	97.55



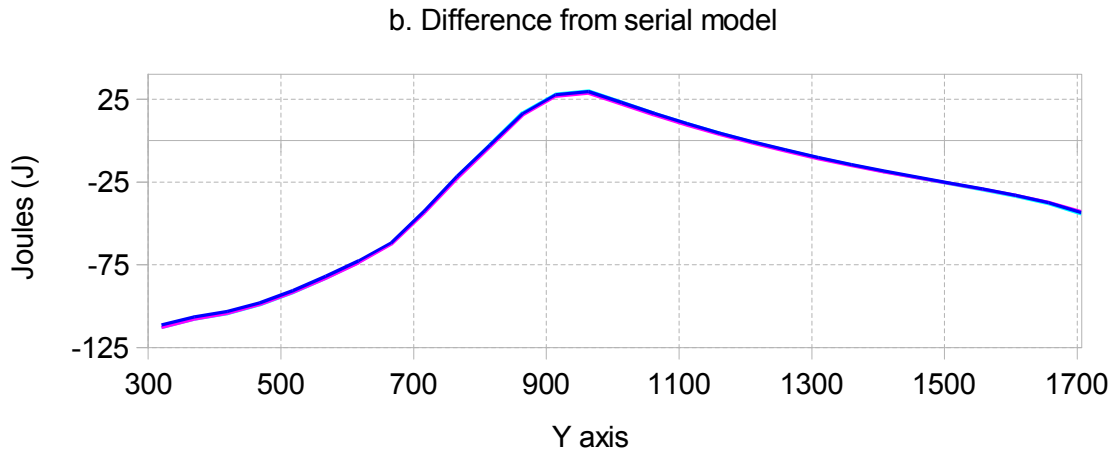
Multi-vertical line paths, heavy motors, 5s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	500.06	611.37	498.62	498.61
369.39	483.67	590.07	482.34	482.3
418.9	465.77	568.86	464.53	464.45
468.41	443.3	541.17	442.14	442.06
517.91	417.29	507.85	416.19	416.15
567.42	389.5	471.47	388.46	388.46
616.93	361.23	433.96	360.24	360.31
666.43	335.03	396.6	333.92	334.35
715.94	317.85	360.32	316.63	317.43
765.45	309.44	331.12	308.12	309.3
814.95	309.55	312.24	308.3	309.83
864.46	318.83	302.49	317.69	319.25
913.97	328.84	301.12	327.75	329.16
963.47	338.36	308.72	337.32	338.59
1012.98	347.45	324.38	346.44	347.59
1062.49	356.16	339.72	355.19	356.22
1111.99	364.56	354.28	363.63	364.55
1161.5	372.69	368.15	371.81	372.62
1211.01	380.68	381.43	379.85	380.54
1260.51	388.64	394.3	387.86	388.44
1310.02	396.74	406.94	396.03	396.48
1359.52	405.16	419.57	404.53	404.85
1409.03	414.16	432.5	413.62	413.8
1458.54	424.07	446.11	423.64	423.67
1508.04	435.63	461.26	435.34	435.18
1557.55	449.48	478.68	449.35	448.99
1607.06	467.22	500.25	467.29	466.68
1656.56	491.25	528.75	491.56	490.65
1706.07	526.59	570.04	527.21	525.91



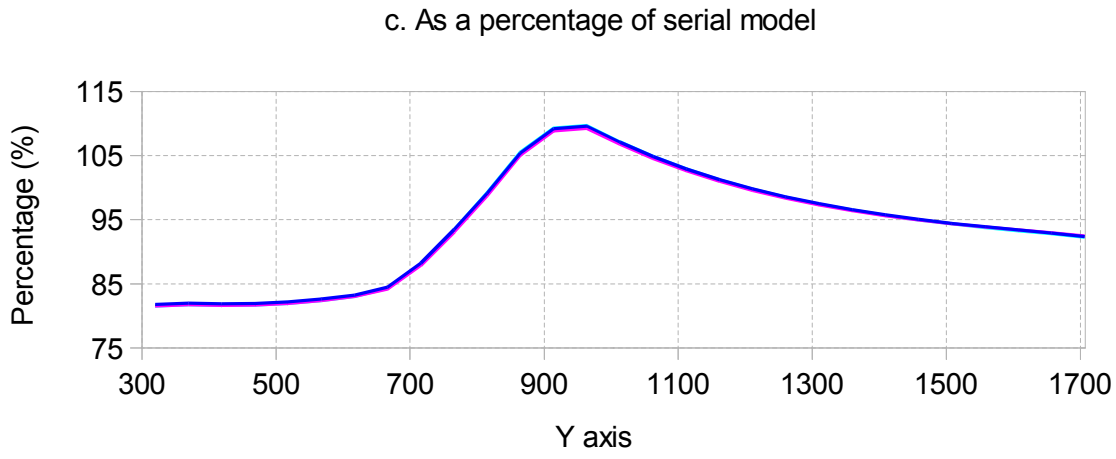
Multi-vertical line paths, heavy motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-111.31	-112.75	-112.76
369.39	-106.39	-107.73	-107.77
418.9	-103.09	-104.33	-104.42
468.41	-97.86	-99.02	-99.11
517.91	-90.56	-91.66	-91.71
567.42	-81.97	-83.01	-83
616.93	-72.73	-73.72	-73.65
666.43	-61.58	-62.68	-62.25
715.94	-42.48	-43.69	-42.89
765.45	-21.68	-23.01	-21.82
814.95	-2.69	-3.94	-2.41
864.46	16.34	15.2	16.76
913.97	27.72	26.63	28.04
963.47	29.64	28.59	29.87
1012.98	23.07	22.07	23.22
1062.49	16.44	15.47	16.5
1111.99	10.27	9.35	10.26
1161.5	4.55	3.67	4.47
1211.01	-0.75	-1.59	-0.9
1260.51	-5.66	-6.44	-5.86
1310.02	-10.2	-10.91	-10.46
1359.52	-14.41	-15.04	-14.72
1409.03	-18.34	-18.88	-18.7
1458.54	-22.04	-22.47	-22.44
1508.04	-25.63	-25.92	-26.07
1557.55	-29.2	-29.34	-29.7
1607.06	-33.03	-32.97	-33.57
1656.56	-37.5	-37.19	-38.1
1706.07	-43.46	-42.83	-44.13



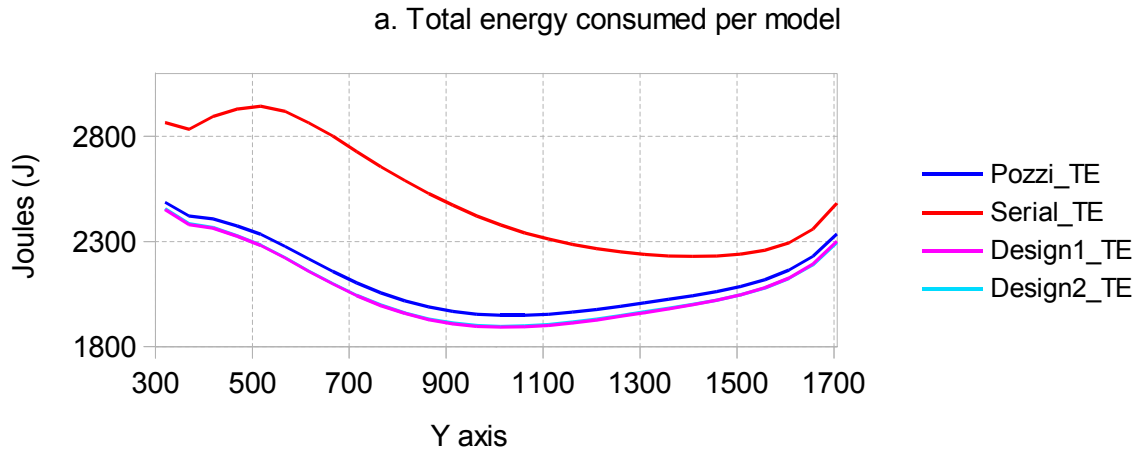
Multi-vertical line paths, heavy motors, 5s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	81.79	81.56	81.56
369.39	81.97	81.74	81.74
418.9	81.88	81.66	81.64
468.41	81.92	81.7	81.69
517.91	82.17	81.95	81.94
567.42	82.61	82.39	82.4
616.93	83.24	83.01	83.03
666.43	84.47	84.2	84.3
715.94	88.21	87.87	88.1
765.45	93.45	93.05	93.41
814.95	99.14	98.74	99.23
864.46	105.4	105.03	105.54
913.97	109.21	108.84	109.31
963.47	109.6	109.26	109.68
1012.98	107.11	106.8	107.16
1062.49	104.84	104.56	104.86
1111.99	102.9	102.64	102.9
1161.5	101.24	101	101.21
1211.01	99.8	99.58	99.77
1260.51	98.56	98.37	98.51
1310.02	97.49	97.32	97.43
1359.52	96.57	96.41	96.49
1409.03	95.76	95.63	95.68
1458.54	95.06	94.96	94.97
1508.04	94.44	94.38	94.35
1557.55	93.9	93.87	93.8
1607.06	93.4	93.41	93.29
1656.56	92.91	92.97	92.79
1706.07	92.38	92.49	92.26



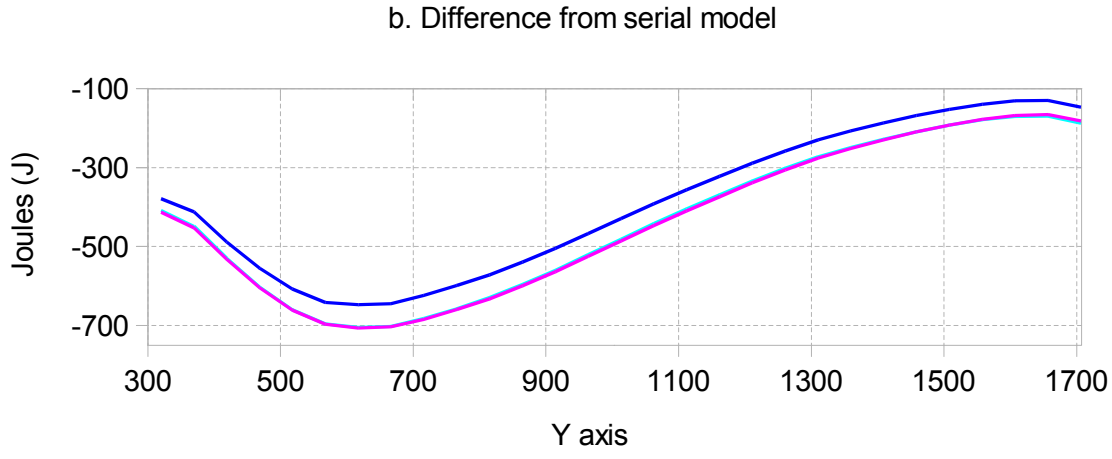
Multi-vertical line paths, heavy motors, 0.25s

Y	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
319.89	2486.33	2865.4	2452.11	2457.22
369.39	2421.42	2833.97	2381.05	2385.58
418.9	2407.43	2895.68	2363.61	2366.38
468.41	2374.16	2929.25	2324.83	2326.57
517.91	2335.27	2942.9	2281.65	2282.78
567.42	2277.14	2918.88	2222.44	2222.97
616.93	2217.67	2865.04	2158.74	2159.99
666.43	2157.93	2802.76	2099.02	2100.01
715.94	2103.76	2727.51	2042.7	2044.79
765.45	2056.32	2655.27	1995.84	1997.8
814.95	2018.72	2590.69	1958	1960.95
864.46	1988.67	2528.22	1928.51	1931.85
913.97	1968.48	2473.27	1909.3	1912.94
963.47	1954.89	2421.97	1896.93	1901.21
1012.98	1949	2378.57	1892.47	1896.85
1062.49	1949.42	2341.11	1894.45	1898.83
1111.99	1954.53	2311.01	1901.45	1906.03
1161.5	1964.09	2286.53	1912.76	1917.14
1211.01	1977.1	2265.9	1927.58	1931.65
1260.51	1992.49	2250.33	1944.82	1948.48
1310.02	2008.55	2238.34	1962.47	1965.5
1359.52	2024.94	2231.74	1980.47	1982.8
1409.03	2042.47	2229.36	1999.64	2001.2
1458.54	2062.2	2230.57	2021	2021.7
1508.04	2087.25	2240.05	2047.61	2047.33
1557.55	2119	2258.36	2080.88	2079.48
1607.06	2163.59	2294.31	2126.84	2124.12
1656.56	2229.36	2359.06	2193.76	2189.46
1706.07	2336.42	2482.93	2301.46	2295.17



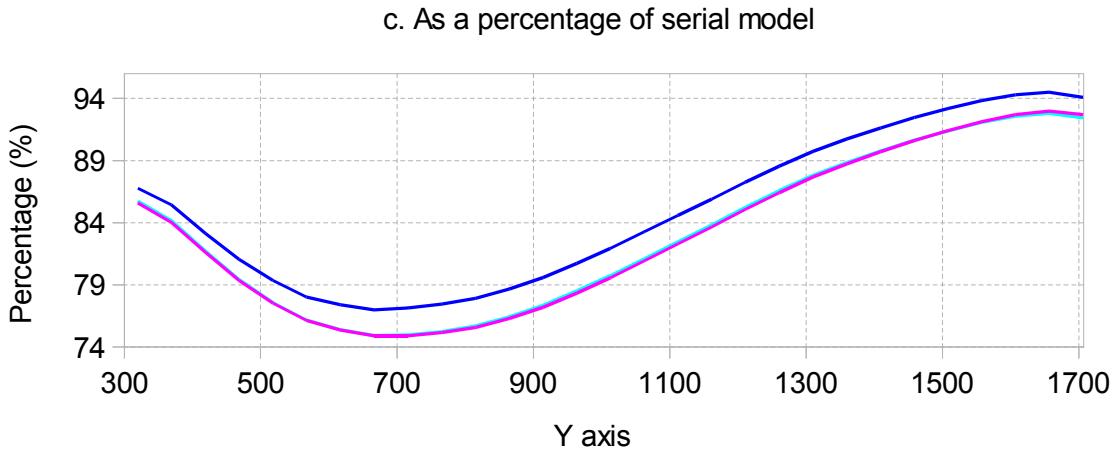
Multi-vertical line paths, heavy motors, 0.25s

Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	-379.07	-413.29	-408.18
369.39	-412.54	-452.91	-448.38
418.9	-488.25	-532.07	-529.3
468.41	-555.08	-604.41	-602.67
517.91	-607.63	-661.25	-660.12
567.42	-641.73	-696.44	-695.91
616.93	-647.37	-706.3	-705.04
666.43	-644.83	-703.74	-702.75
715.94	-623.75	-684.81	-682.72
765.45	-598.95	-659.44	-657.47
814.95	-571.97	-632.69	-629.74
864.46	-539.55	-599.71	-596.37
913.97	-504.79	-563.97	-560.33
963.47	-467.07	-525.04	-520.76
1012.98	-429.58	-486.1	-481.72
1062.49	-391.69	-446.66	-442.27
1111.99	-356.48	-409.56	-404.98
1161.5	-322.44	-373.77	-369.39
1211.01	-288.8	-338.32	-334.25
1260.51	-257.84	-305.51	-301.85
1310.02	-229.79	-275.87	-272.85
1359.52	-206.8	-251.27	-248.94
1409.03	-186.88	-229.72	-228.16
1458.54	-168.36	-209.57	-208.87
1508.04	-152.8	-192.43	-192.72
1557.55	-139.36	-177.48	-178.89
1607.06	-130.72	-167.46	-170.18
1656.56	-129.7	-165.31	-169.61
1706.07	-146.5	-181.47	-187.76



Multi-vertical line paths, heavy motors, 0.25s

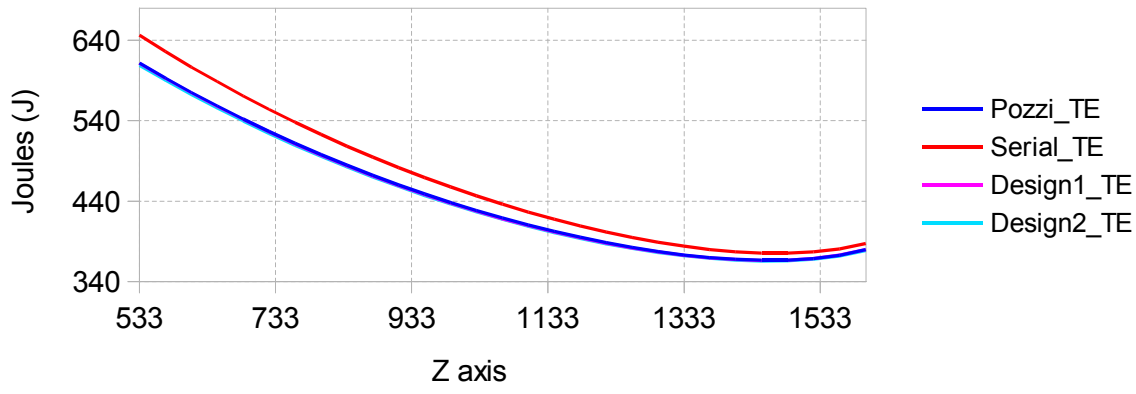
Y	Pozzi_TE	Design1_TE	Design2_TE
319.89	86.77	85.58	85.75
369.39	85.44	84.02	84.18
418.9	83.14	81.63	81.72
468.41	81.05	79.37	79.43
517.91	79.35	77.53	77.57
567.42	78.01	76.14	76.16
616.93	77.4	75.35	75.39
666.43	76.99	74.89	74.93
715.94	77.13	74.89	74.97
765.45	77.44	75.17	75.24
814.95	77.92	75.58	75.69
864.46	78.66	76.28	76.41
913.97	79.59	77.2	77.34
963.47	80.72	78.32	78.5
1012.98	81.94	79.56	79.75
1062.49	83.27	80.92	81.11
1111.99	84.57	82.28	82.48
1161.5	85.9	83.65	83.84
1211.01	87.25	85.07	85.25
1260.51	88.54	86.42	86.59
1310.02	89.73	87.68	87.81
1359.52	90.73	88.74	88.85
1409.03	91.62	89.7	89.77
1458.54	92.45	90.6	90.64
1508.04	93.18	91.41	91.4
1557.55	93.83	92.14	92.08
1607.06	94.3	92.7	92.58
1656.56	94.5	92.99	92.81
1706.07	94.1	92.69	92.44



Multi-radial line paths, light motors, 5s

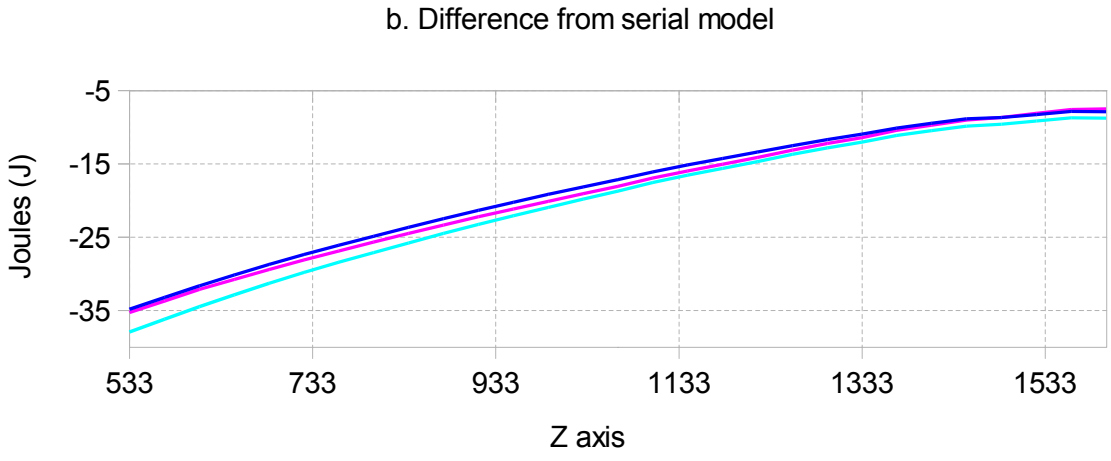
Z	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
533.15	611.77	646.58	611.34	608.64
571.23	593.15	626.34	592.67	590.18
609.31	575.28	606.9	574.78	572.42
647.39	558.43	588.58	557.83	555.72
685.47	542.26	570.98	541.58	539.69
723.56	526.78	554.12	526.04	524.35
761.64	512.02	538.13	511.26	509.71
799.72	498.12	522.96	497.3	495.86
837.8	484.87	508.49	484.01	482.72
875.88	472.28	494.73	471.39	470.24
913.97	460.35	481.66	459.44	458.41
952.05	449.07	469.28	448.15	447.21
990.13	438.43	457.58	437.5	436.66
1028.21	428.41	446.54	427.49	426.73
1066.29	419.03	436.17	418.12	417.46
1104.38	410.38	426.45	409.53	408.91
1142.46	402.4	417.52	401.58	400.99
1180.54	395.07	409.31	394.29	393.72
1218.62	388.39	401.79	387.67	387.09
1256.7	382.46	394.97	381.85	381.27
1294.79	377.42	389.08	376.88	376.27
1332.87	373.14	384.07	372.67	372.03
1370.95	369.82	379.92	369.51	368.8
1409.03	367.61	377.08	367.39	366.62
1447.11	366.48	375.33	366.34	365.51
1485.19	366.61	375.25	366.57	365.66
1523.28	368.57	376.84	368.73	367.69
1561.36	372.69	380.52	372.95	371.81
1599.44	379.61	387.46	379.97	378.72

a. Total energy consumed per model



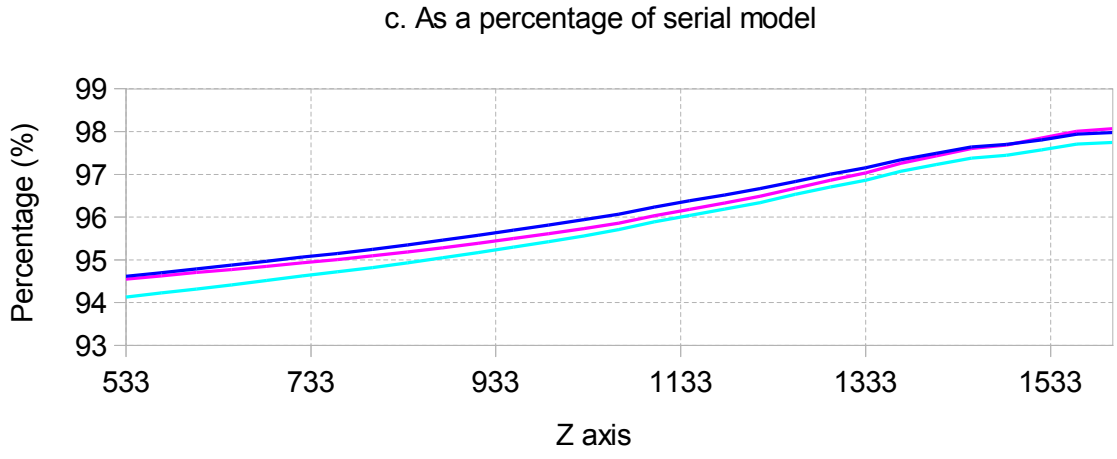
Multi-radial line paths, light motors, 5s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	-34.81	-35.24	-37.94
571.23	-33.2	-33.68	-36.16
609.31	-31.62	-32.13	-34.49
647.39	-30.15	-30.75	-32.86
685.47	-28.73	-29.4	-31.29
723.56	-27.34	-28.09	-29.78
761.64	-26.11	-26.87	-28.42
799.72	-24.84	-25.66	-27.1
837.8	-23.62	-24.48	-25.77
875.88	-22.45	-23.33	-24.49
913.97	-21.31	-22.22	-23.25
952.05	-20.21	-21.13	-22.06
990.13	-19.15	-20.08	-20.92
1028.21	-18.13	-19.05	-19.82
1066.29	-17.14	-18.05	-18.71
1104.38	-16.07	-16.93	-17.55
1142.46	-15.11	-15.93	-16.53
1180.54	-14.24	-15.01	-15.59
1218.62	-13.39	-14.12	-14.69
1256.7	-12.51	-13.12	-13.7
1294.79	-11.66	-12.21	-12.81
1332.87	-10.93	-11.4	-12.04
1370.95	-10.1	-10.42	-11.12
1409.03	-9.46	-9.69	-10.45
1447.11	-8.86	-8.99	-9.83
1485.19	-8.64	-8.68	-9.59
1523.28	-8.27	-8.11	-9.15
1561.36	-7.83	-7.57	-8.71
1599.44	-7.85	-7.49	-8.74



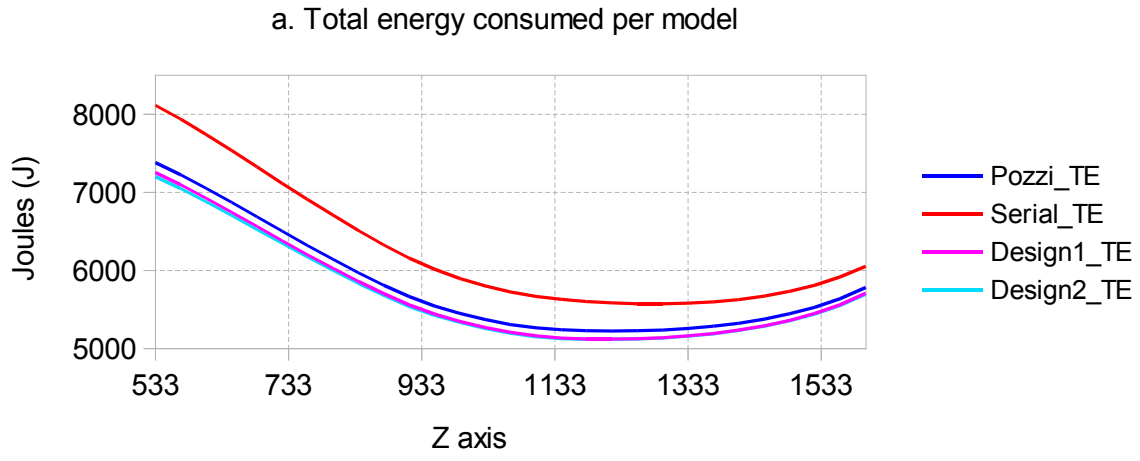
Multi-radial line paths, light motors, 5s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	94.62	94.55	94.13
571.23	94.7	94.62	94.23
609.31	94.79	94.71	94.32
647.39	94.88	94.78	94.42
685.47	94.97	94.85	94.52
723.56	95.07	94.93	94.63
761.64	95.15	95.01	94.72
799.72	95.25	95.09	94.82
837.8	95.35	95.19	94.93
875.88	95.46	95.28	95.05
913.97	95.58	95.39	95.17
952.05	95.69	95.5	95.3
990.13	95.81	95.61	95.43
1028.21	95.94	95.73	95.56
1066.29	96.07	95.86	95.71
1104.38	96.23	96.03	95.89
1142.46	96.38	96.18	96.04
1180.54	96.52	96.33	96.19
1218.62	96.67	96.49	96.34
1256.7	96.83	96.68	96.53
1294.79	97	96.86	96.71
1332.87	97.15	97.03	96.86
1370.95	97.34	97.26	97.07
1409.03	97.49	97.43	97.23
1447.11	97.64	97.6	97.38
1485.19	97.7	97.69	97.44
1523.28	97.81	97.85	97.57
1561.36	97.94	98.01	97.71
1599.44	97.97	98.07	97.74



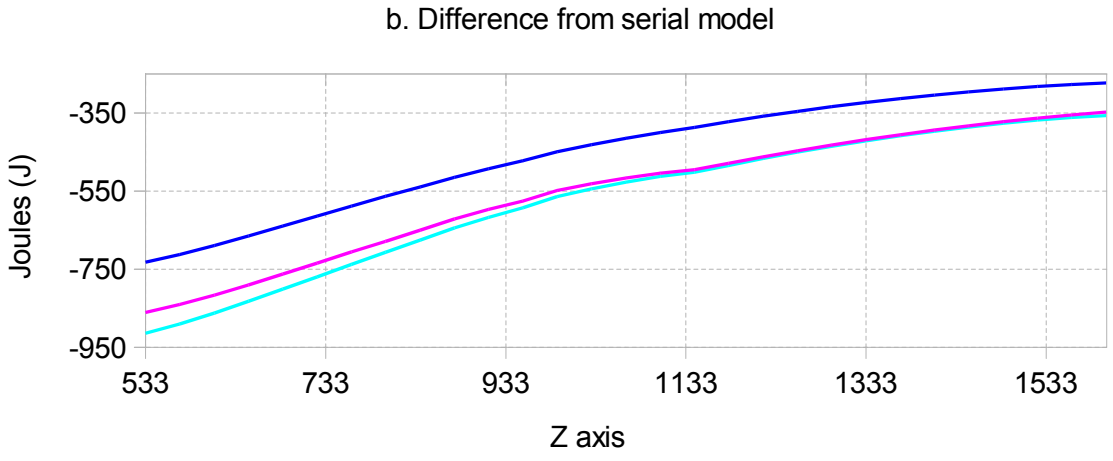
Multi-radial line paths, light motors, 0.25s

Z	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
533.15	7381.93	8114.46	7253.1	7200.03
571.23	7225.1	7937.52	7096.83	7047.21
609.31	7053.04	7742.7	6926.06	6880.04
647.39	6872.12	7537.34	6746.99	6704.58
685.47	6687.31	7327.18	6564.4	6525.59
723.56	6502.4	7116.62	6381.98	6346.69
761.64	6320.21	6908.95	6202.46	6170.58
799.72	6142.81	6706.56	6027.84	5999.21
837.8	5971.71	6511.13	5861.09	5835.35
875.88	5811.88	6327.29	5705.44	5682.48
913.97	5666.69	6159.78	5562.94	5542.59
952.05	5545.74	6018.21	5442.73	5425.13
990.13	5449.13	5898.2	5349.26	5334.13
1028.21	5370.84	5802.62	5270.74	5257.9
1066.29	5309.32	5724.77	5208.05	5197.34
1104.38	5265.54	5666.28	5162.08	5153.31
1142.46	5242.06	5629.25	5134.11	5127.14
1180.54	5230.56	5603.1	5124.23	5118.73
1218.62	5225.77	5584.6	5122.13	5117.77
1256.7	5228.17	5574.24	5127.24	5123.76
1294.79	5238.39	5572.62	5140.2	5137.33
1332.87	5257.19	5580.5	5161.79	5159.24
1370.95	5285.61	5598.89	5193.02	5190.52
1409.03	5324.97	5629.11	5235.23	5232.5
1447.11	5377.06	5672.99	5290.22	5286.95
1485.19	5444.33	5733	5360.44	5356.32
1523.28	5530.28	5812.68	5449.37	5444.08
1561.36	5640.02	5917.23	5562.14	5555.3
1599.44	5781.57	6054.86	5706.76	5697.97



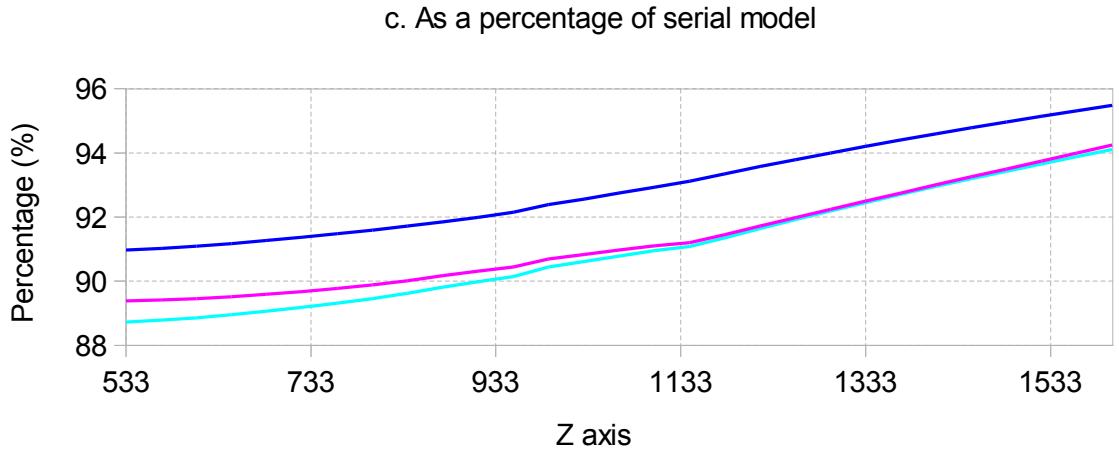
Multi-radial line paths, light motors, 0.25s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	-732.53	-861.36	-914.44
571.23	-712.42	-840.7	-890.31
609.31	-689.65	-816.63	-862.66
647.39	-665.21	-790.35	-832.76
685.47	-639.87	-762.78	-801.59
723.56	-614.23	-734.64	-769.93
761.64	-588.74	-706.49	-738.37
799.72	-563.74	-678.72	-707.34
837.8	-539.42	-650.04	-675.78
875.88	-515.41	-621.85	-644.81
913.97	-493.09	-596.84	-617.19
952.05	-472.47	-575.48	-593.08
990.13	-449.07	-548.94	-564.07
1028.21	-431.78	-531.89	-544.72
1066.29	-415.45	-516.72	-527.43
1104.38	-400.74	-504.2	-512.97
1142.46	-387.19	-495.14	-502.11
1180.54	-372.53	-478.87	-484.37
1218.62	-358.84	-462.48	-466.83
1256.7	-346.07	-447	-450.48
1294.79	-334.24	-432.42	-435.29
1332.87	-323.3	-418.71	-421.26
1370.95	-313.27	-405.87	-408.36
1409.03	-304.15	-393.88	-396.62
1447.11	-295.93	-382.77	-386.04
1485.19	-288.67	-372.56	-376.68
1523.28	-282.4	-363.31	-368.6
1561.36	-277.22	-355.1	-361.93
1599.44	-273.28	-348.09	-356.89



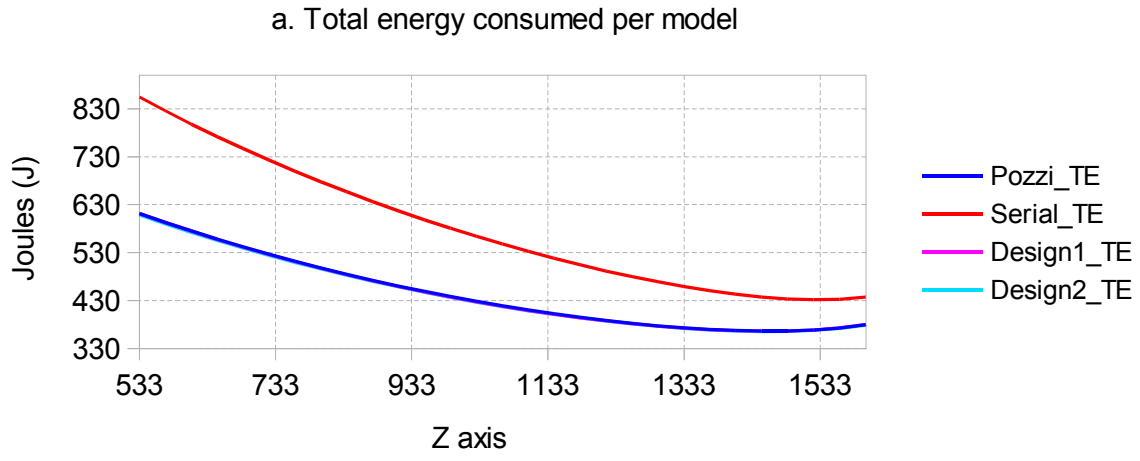
Multi-radial line paths, light motors, 0.25s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	90.97	89.38	88.73
571.23	91.02	89.41	88.78
609.31	91.09	89.45	88.86
647.39	91.17	89.51	88.95
685.47	91.27	89.59	89.06
723.56	91.37	89.68	89.18
761.64	91.48	89.77	89.31
799.72	91.59	89.88	89.45
837.8	91.72	90.02	89.62
875.88	91.85	90.17	89.81
913.97	92	90.31	89.98
952.05	92.15	90.44	90.15
990.13	92.39	90.69	90.44
1028.21	92.56	90.83	90.61
1066.29	92.74	90.97	90.79
1104.38	92.93	91.1	90.95
1142.46	93.12	91.2	91.08
1180.54	93.35	91.45	91.36
1218.62	93.57	91.72	91.64
1256.7	93.79	91.98	91.92
1294.79	94	92.24	92.19
1332.87	94.21	92.5	92.45
1370.95	94.4	92.75	92.71
1409.03	94.6	93	92.95
1447.11	94.78	93.25	93.2
1485.19	94.96	93.5	93.43
1523.28	95.14	93.75	93.66
1561.36	95.32	94	93.88
1599.44	95.49	94.25	94.11



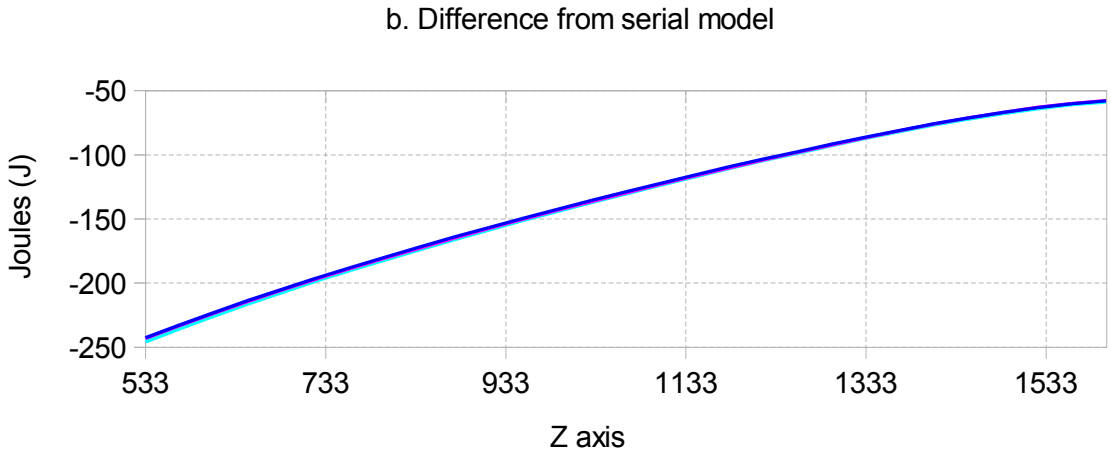
Multi-radial line paths, heavy motors, 5s

Z	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
533.15	611.77	854.43	611.41	608.64
571.23	593.15	825.73	592.67	590.18
609.31	575.28	798.11	574.78	572.42
647.39	558.43	771.94	557.83	555.72
685.47	542.26	746.84	541.58	539.69
723.56	526.78	722.65	526.04	524.35
761.64	512.02	699.77	511.26	509.71
799.72	498.12	677.79	497.3	495.86
837.8	484.87	656.67	484.01	482.72
875.88	472.28	636.43	471.39	470.24
913.97	460.35	617.05	459.44	458.41
952.05	449.07	598.52	448.15	447.21
990.13	438.43	580.83	437.5	436.66
1028.21	428.41	563.97	427.49	426.73
1066.29	419.03	547.92	418.12	417.46
1104.38	410.38	532.67	409.53	408.91
1142.46	402.4	518.21	401.58	400.99
1180.54	395.07	504.54	394.29	393.72
1218.62	388.39	491.73	387.67	387.09
1256.7	382.46	480	381.85	381.27
1294.79	377.42	469.12	376.88	376.27
1332.87	373.14	459.25	372.67	372.03
1370.95	369.82	450.71	369.51	368.8
1409.03	367.61	443.26	367.39	366.62
1447.11	366.48	437.58	366.34	365.51
1485.19	366.61	433.51	366.57	365.66
1523.28	368.57	431.73	368.73	367.69
1561.36	372.69	432.78	372.95	371.81
1599.44	379.61	437.54	379.97	378.72



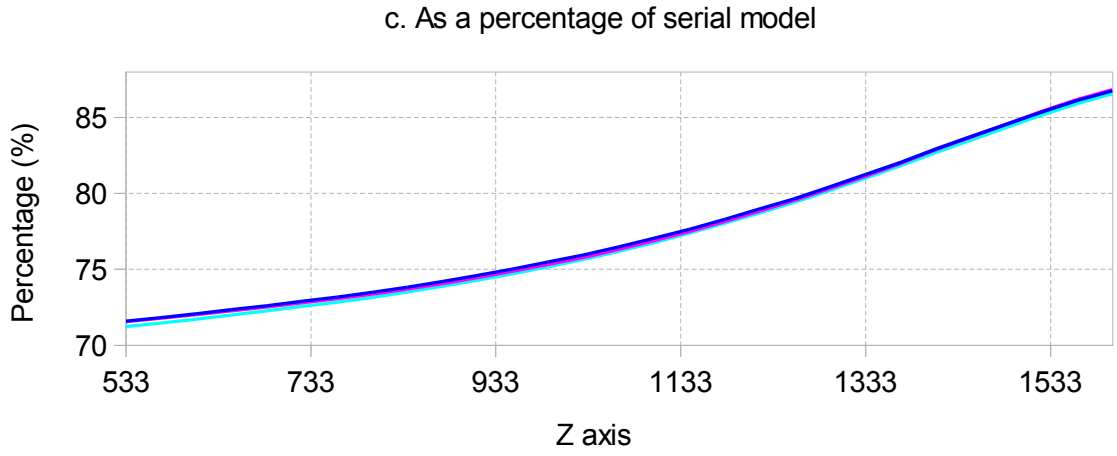
Multi-radial line paths, heavy motors, 5s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	-242.66	-243.03	-245.8
571.23	-232.58	-233.06	-235.54
609.31	-222.83	-223.34	-225.7
647.39	-213.51	-214.11	-216.22
685.47	-204.58	-205.26	-207.15
723.56	-195.87	-196.62	-198.3
761.64	-187.75	-188.51	-190.06
799.72	-179.67	-180.49	-181.93
837.8	-171.81	-172.66	-173.95
875.88	-164.15	-165.03	-166.19
913.97	-156.7	-157.61	-158.64
952.05	-149.45	-150.37	-151.3
990.13	-142.41	-143.33	-144.17
1028.21	-135.55	-136.48	-137.24
1066.29	-128.89	-129.8	-130.45
1104.38	-122.29	-123.14	-123.76
1142.46	-115.81	-116.63	-117.22
1180.54	-109.46	-110.24	-110.82
1218.62	-103.34	-104.07	-104.64
1256.7	-97.54	-98.15	-98.73
1294.79	-91.7	-92.24	-92.85
1332.87	-86.11	-86.58	-87.22
1370.95	-80.89	-81.2	-81.9
1409.03	-75.65	-75.87	-76.64
1447.11	-71.1	-71.23	-72.07
1485.19	-66.89	-66.93	-67.85
1523.28	-63.16	-63	-64.04
1561.36	-60.09	-59.84	-60.97
1599.44	-57.93	-57.58	-58.83



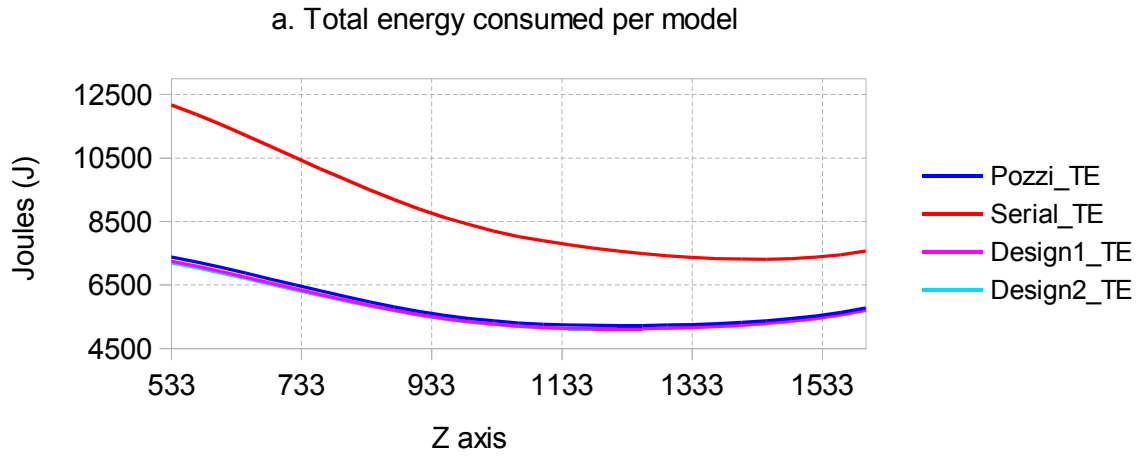
Multi-radial line paths, heavy motors, 5s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	71.6	71.56	71.23
571.23	71.83	71.78	71.47
609.31	72.08	72.02	71.72
647.39	72.34	72.26	71.99
685.47	72.61	72.52	72.26
723.56	72.9	72.79	72.56
761.64	73.17	73.06	72.84
799.72	73.49	73.37	73.16
837.8	73.84	73.71	73.51
875.88	74.21	74.07	73.89
913.97	74.61	74.46	74.29
952.05	75.03	74.88	74.72
990.13	75.48	75.32	75.18
1028.21	75.96	75.8	75.66
1066.29	76.48	76.31	76.19
1104.38	77.04	76.88	76.77
1142.46	77.65	77.49	77.38
1180.54	78.3	78.15	78.04
1218.62	78.98	78.84	78.72
1256.7	79.68	79.55	79.43
1294.79	80.45	80.34	80.21
1332.87	81.25	81.15	81.01
1370.95	82.05	81.98	81.83
1409.03	82.93	82.88	82.71
1447.11	83.75	83.72	83.53
1485.19	84.57	84.56	84.35
1523.28	85.37	85.41	85.17
1561.36	86.12	86.17	85.91
1599.44	86.76	86.84	86.56



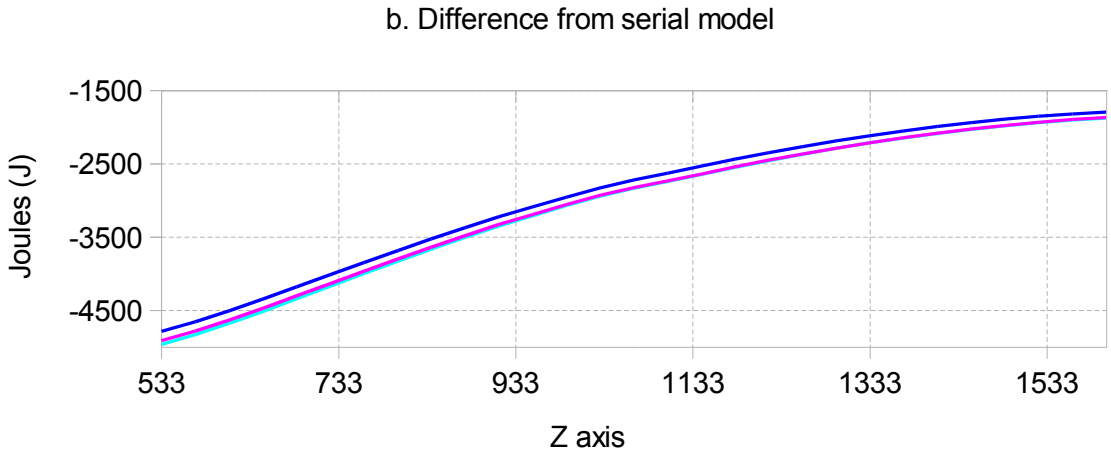
Multi-radial line paths, heavy motors, 0.25s

Z	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
533.15	7381.93	12164.61	7253.1	7200.03
571.23	7225.1	11876.57	7096.83	7047.21
609.31	7053.04	11555.97	6926.06	6880.04
647.39	6872.12	11215.61	6746.99	6704.58
685.47	6687.31	10865.49	6564.4	6525.59
723.56	6502.4	10513.32	6381.98	6346.69
761.64	6320.21	10164.9	6202.46	6170.58
799.72	6142.81	9824.47	6027.84	5999.21
837.8	5971.71	9495.06	5861.09	5835.35
875.88	5811.88	9180.58	5705.44	5682.48
913.97	5666.69	8888.39	5562.94	5542.59
952.05	5545.74	8632.43	5442.73	5425.13
990.13	5449.13	8403.03	5349.26	5334.13
1028.21	5370.84	8198.85	5270.74	5257.9
1066.29	5309.32	8028.2	5208.05	5197.34
1104.38	5265.54	7891.4	5162.08	5153.31
1142.46	5242.06	7771.91	5134.11	5127.14
1180.54	5230.56	7665.08	5124.23	5118.73
1218.62	5225.77	7571.2	5122.13	5117.77
1256.7	5228.17	7490.64	5127.24	5123.76
1294.79	5238.39	7423.91	5140.2	5137.33
1332.87	5257.19	7371.69	5161.79	5159.24
1370.95	5285.61	7334.97	5193.02	5190.52
1409.03	5324.97	7315.07	5235.23	5232.5
1447.11	5377.06	7313.89	5290.22	5286.95
1485.19	5444.33	7334.08	5360.44	5356.32
1523.28	5530.28	7379.48	5449.37	5444.08
1561.36	5640.02	7455.82	5562.14	5555.3
1599.44	5781.57	7572.13	5706.76	5697.97



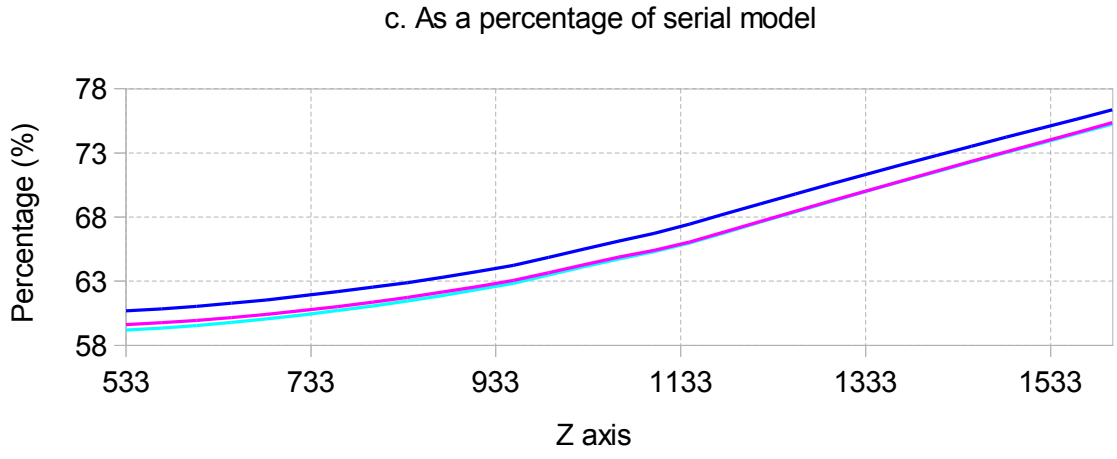
Multi-radial line paths, heavy motors, 0.25s

Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	-4782.68	-4911.51	-4964.58
571.23	-4651.47	-4779.74	-4829.35
609.31	-4502.93	-4629.91	-4675.94
647.39	-4343.48	-4468.62	-4511.03
685.47	-4178.18	-4301.08	-4339.9
723.56	-4010.93	-4131.34	-4166.63
761.64	-3844.69	-3962.44	-3994.32
799.72	-3681.66	-3796.64	-3825.26
837.8	-3523.36	-3633.97	-3659.71
875.88	-3368.7	-3475.14	-3498.1
913.97	-3221.7	-3325.45	-3345.79
952.05	-3086.68	-3189.7	-3207.3
990.13	-2953.9	-3053.77	-3068.9
1028.21	-2828	-2928.11	-2940.94
1066.29	-2718.87	-2820.15	-2830.86
1104.38	-2625.85	-2729.32	-2738.08
1142.46	-2529.85	-2637.8	-2644.78
1180.54	-2434.52	-2540.85	-2546.35
1218.62	-2345.43	-2449.08	-2453.43
1256.7	-2262.47	-2363.4	-2366.88
1294.79	-2185.52	-2283.7	-2286.58
1332.87	-2114.5	-2209.9	-2212.45
1370.95	-2049.36	-2141.95	-2144.44
1409.03	-1990.11	-2079.84	-2082.58
1447.11	-1936.84	-2023.68	-2026.95
1485.19	-1889.75	-1973.65	-1977.76
1523.28	-1849.21	-1930.12	-1935.41
1561.36	-1815.8	-1893.68	-1900.52
1599.44	-1790.56	-1865.37	-1874.17



Multi-radial line paths, heavy motors, 0.25s

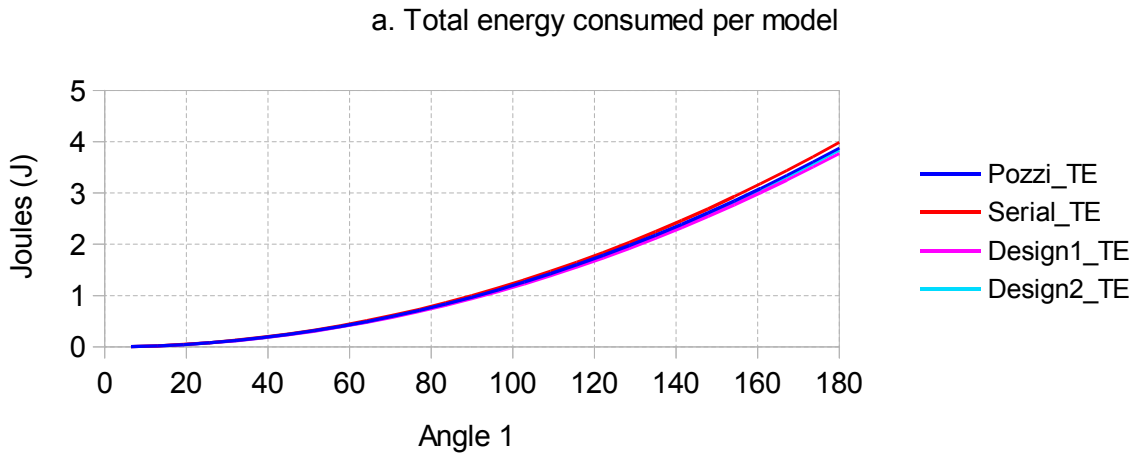
Z	Pozzi_TE	Design1_TE	Design2_TE
533.15	60.68	59.62	59.19
571.23	60.83	59.75	59.34
609.31	61.03	59.93	59.54
647.39	61.27	60.16	59.78
685.47	61.55	60.42	60.06
723.56	61.85	60.7	60.37
761.64	62.18	61.02	60.7
799.72	62.53	61.36	61.06
837.8	62.89	61.73	61.46
875.88	63.31	62.15	61.9
913.97	63.75	62.59	62.36
952.05	64.24	63.05	62.85
990.13	64.85	63.66	63.48
1028.21	65.51	64.29	64.13
1066.29	66.13	64.87	64.74
1104.38	66.73	65.41	65.3
1142.46	67.45	66.06	65.97
1180.54	68.24	66.85	66.78
1218.62	69.02	67.65	67.6
1256.7	69.8	68.45	68.4
1294.79	70.56	69.24	69.2
1332.87	71.32	70.02	69.99
1370.95	72.06	70.8	70.76
1409.03	72.79	71.57	71.53
1447.11	73.52	72.33	72.29
1485.19	74.23	73.09	73.03
1523.28	74.94	73.84	73.77
1561.36	75.65	74.6	74.51
1599.44	76.35	75.37	75.25



Varying angle 1, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.004942	0.005087	0.004798	0.004888
12.86	0.019769	0.020350	0.019229	0.019554
19.29	0.044479	0.045787	0.043264	0.043995
25.71	0.079074	0.081398	0.076915	0.078214
32.14	0.123554	0.127185	0.120179	0.122209
38.57	0.177917	0.183146	0.173058	0.175981
45	0.242165	0.249282	0.235551	0.239530
51.43	0.316297	0.325593	0.307658	0.312855
57.86	0.400314	0.412079	0.389380	0.395958
64.29	0.494215	0.508739	0.480716	0.488837
70.71	0.598000	0.615575	0.581666	0.591492
77.14	0.711669	0.732585	0.692231	0.703925
83.57	0.835222	0.859770	0.812409	0.826134
90	0.968660	0.997129	0.942203	0.958119
96.43	1.111983	1.144664	1.081610	1.099882
102.86	1.265189	1.302373	1.230632	1.251421
109.29	1.428280	1.470257	1.389268	1.412737
115.71	1.601255	1.648315	1.557519	1.583830
122.14	1.784114	1.836549	1.735384	1.764700
128.57	1.976858	2.034957	1.922863	1.955346
135	2.179486	2.243541	2.119956	2.155769
141.43	2.391998	2.462298	2.326664	2.365968
147.86	2.614395	2.691231	2.542986	2.585945
154.29	2.846675	2.930339	2.768922	2.815698
160.71	3.088840	3.179621	3.004473	3.055228
167.14	3.340890	3.439078	3.249638	3.304534
173.57	3.602823	3.708710	3.504417	3.563618
180	3.874641	3.988516	3.768811	3.832478

Max	3.988516
Min	0.004798

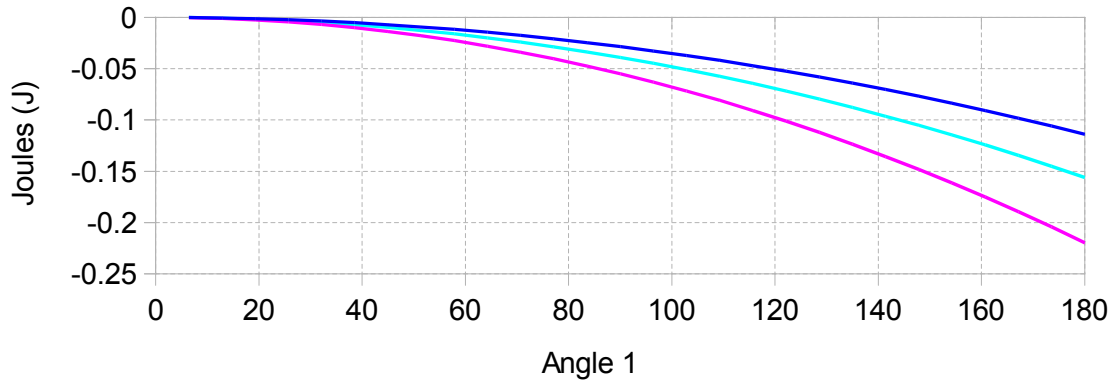


Varying angle 1, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.000145	-0.000289	-0.000199
12.86	-0.000581	-0.001121	-0.000796
19.29	-0.001307	-0.002522	-0.001791
25.71	-0.002324	-0.004484	-0.003185
32.14	-0.003631	-0.007006	-0.004976
38.57	-0.005229	-0.010089	-0.007165
45	-0.007117	-0.013732	-0.009752
51.43	-0.009296	-0.017935	-0.012738
57.86	-0.011765	-0.022699	-0.016121
64.29	-0.014525	-0.028024	-0.019903
70.71	-0.017575	-0.033909	-0.024083
77.14	-0.020916	-0.040354	-0.028660
83.57	-0.024547	-0.047360	-0.033636
90	-0.028469	-0.054926	-0.039010
96.43	-0.032681	-0.063053	-0.044782
102.86	-0.037184	-0.071741	-0.050951
109.29	-0.041977	-0.080988	-0.057519
115.71	-0.047061	-0.090797	-0.064485
122.14	-0.052435	-0.101166	-0.071849
128.57	-0.058100	-0.112095	-0.079612
135	-0.064055	-0.123585	-0.087772
141.43	-0.070300	-0.135635	-0.096330
147.86	-0.076837	-0.148245	-0.105286
154.29	-0.083663	-0.161417	-0.114641
160.71	-0.090780	-0.175148	-0.124393
167.14	-0.098188	-0.189440	-0.134544
173.57	-0.105886	-0.204293	-0.145092
180	-0.113875	-0.219706	-0.156039

Max	-0.000145
Min	-0.219706

b. Difference from serial model

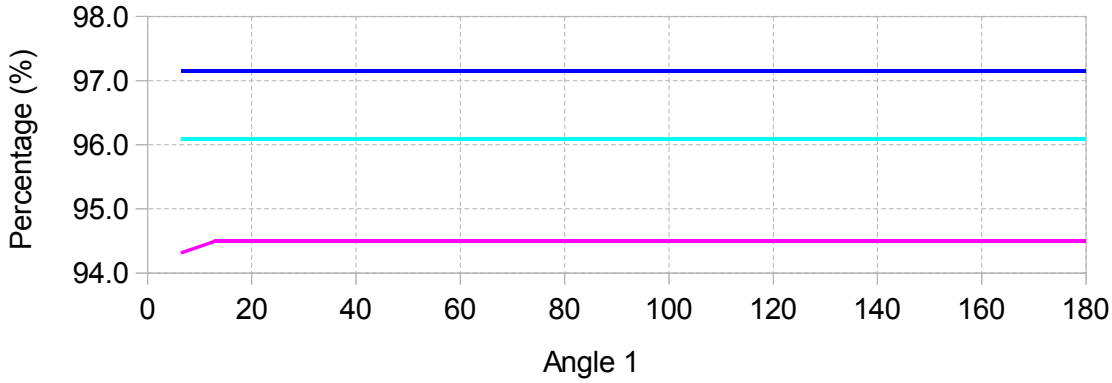


Varying angle 1, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	97.14	94.31	96.09
12.86	97.14	94.49	96.09
19.29	97.15	94.49	96.09
25.71	97.14	94.49	96.09
32.14	97.14	94.49	96.09
38.57	97.14	94.49	96.09
45	97.14	94.49	96.09
51.43	97.14	94.49	96.09
57.86	97.14	94.49	96.09
64.29	97.14	94.49	96.09
70.71	97.14	94.49	96.09
77.14	97.14	94.49	96.09
83.57	97.14	94.49	96.09
90	97.14	94.49	96.09
96.43	97.14	94.49	96.09
102.86	97.14	94.49	96.09
109.29	97.14	94.49	96.09
115.71	97.14	94.49	96.09
122.14	97.14	94.49	96.09
128.57	97.14	94.49	96.09
135	97.14	94.49	96.09
141.43	97.14	94.49	96.09
147.86	97.14	94.49	96.09
154.29	97.14	94.49	96.09
160.71	97.14	94.49	96.09
167.14	97.14	94.49	96.09
173.57	97.14	94.49	96.09
180	97.14	94.49	96.09

Max	97.15
Min	94.31

c. As a percentage of serial model

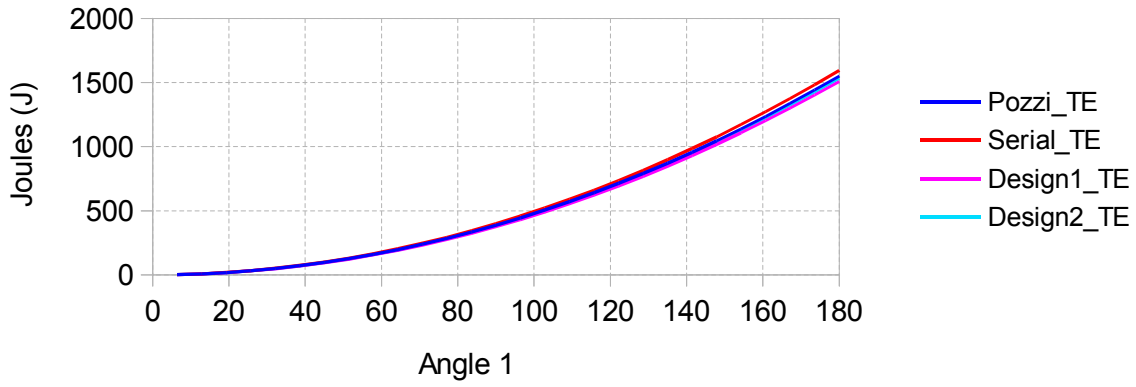


Varying angle 1, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	1.976858	2.034957	1.922863	1.955346
12.86	7.907431	8.139829	7.691450	7.821383
19.29	17.791720	18.314616	17.305763	17.598112
25.71	31.629725	32.559317	30.765801	31.285533
32.14	49.421445	50.873933	48.071564	48.883645
38.57	71.166882	73.258464	69.223053	70.392449
45	96.866033	99.712909	94.220266	95.811945
51.43	126.518900	130.237270	123.063200	125.142130
57.86	160.125480	164.831540	155.751870	158.383010
64.29	197.685780	203.495730	192.286260	195.534580
70.71	239.199800	246.229840	232.666370	236.596840
77.14	284.667530	293.033860	276.892210	281.569800
83.57	334.088970	343.907790	324.963770	330.453440
90	387.464130	398.851640	376.881060	383.247780
96.43	444.793010	457.865400	432.644080	439.952810
102.86	506.075600	520.949080	492.252820	500.568530
109.29	571.311910	588.102670	555.707280	565.094940
115.71	640.501930	659.326170	623.007470	633.532040
122.14	713.645670	734.619600	694.153390	705.879840
128.57	790.743130	813.982930	769.145030	782.138320
135	871.794300	897.416180	847.982390	862.307500
141.43	956.799180	984.919350	930.665480	946.387370
147.86	1045.757800	1076.492400	1017.194300	1034.377900
154.29	1138.670100	1172.135400	1107.568800	1126.279200
160.71	1235.536100	1271.848300	1201.789100	1222.091100
167.14	1336.355900	1375.631200	1299.855100	1321.813800
173.57	1441.129400	1483.483900	1401.766800	1425.447100
180	1549.856500	1595.406500	1507.524300	1532.991100

Max	1595.406500
Min	1.922863

a. Total energy consumed per model

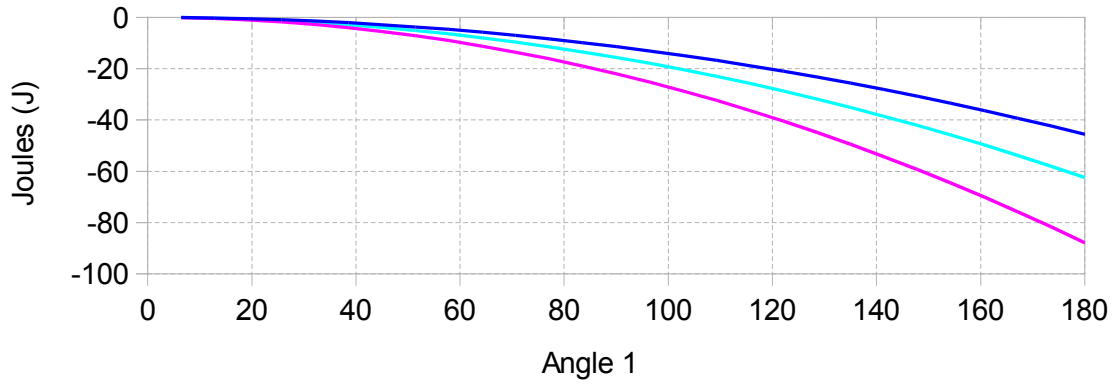


Varying angle 1, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.058100	-0.112095	-0.079612
12.86	-0.232398	-0.448379	-0.318446
19.29	-0.522896	-1.008853	-0.716504
25.71	-0.929592	-1.793516	-1.273784
32.14	-1.452488	-2.802369	-1.990288
38.57	-2.091582	-4.035411	-2.866015
45	-2.846876	-5.492643	-3.900964
51.43	-3.718370	-7.174070	-5.095140
57.86	-4.706060	-9.079670	-6.448530
64.29	-5.809950	-11.209470	-7.961150
70.71	-7.030040	-13.563470	-9.633000
77.14	-8.366330	-16.141650	-11.464060
83.57	-9.818820	-18.944020	-13.454350
90	-11.387510	-21.970580	-15.603860
96.43	-13.072390	-25.221320	-17.912590
102.86	-14.873480	-28.696260	-20.380550
109.29	-16.790760	-32.395390	-23.007730
115.71	-18.824240	-36.318700	-25.794130
122.14	-20.973930	-40.466210	-28.739760
128.57	-23.239800	-44.837900	-31.844610
135	-25.621880	-49.433790	-35.108680
141.43	-28.120170	-54.253870	-38.531980
147.86	-30.734600	-59.298100	-42.114500
154.29	-33.465300	-64.566600	-45.856200
160.71	-36.312200	-70.059200	-49.757200
167.14	-39.275300	-75.776100	-53.817400
173.57	-42.354500	-81.717100	-58.036800
180	-45.550000	-87.882200	-62.415400

Max	-0.058100
Min	-87.882200

b. Difference from serial model

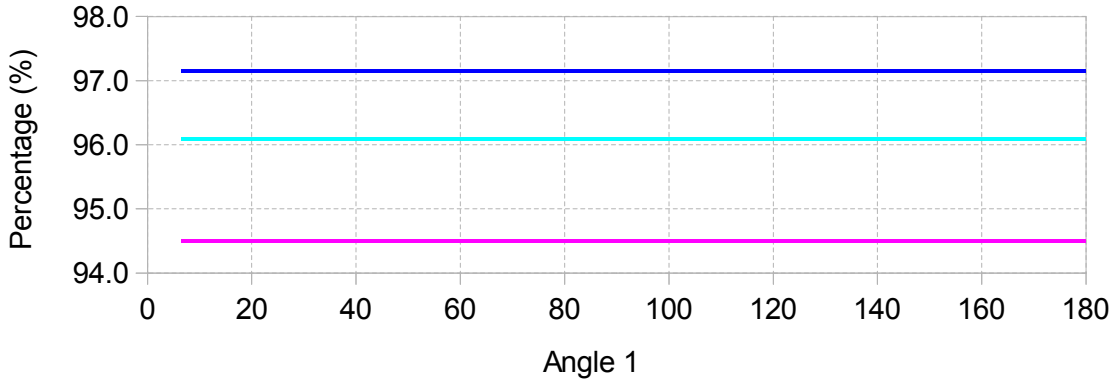


Varying angle 1, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	97.14	94.49	96.09
12.86	97.14	94.49	96.09
19.29	97.14	94.49	96.09
25.71	97.14	94.49	96.09
32.14	97.14	94.49	96.09
38.57	97.14	94.49	96.09
45	97.14	94.49	96.09
51.43	97.14	94.49	96.09
57.86	97.14	94.49	96.09
64.29	97.14	94.49	96.09
70.71	97.14	94.49	96.09
77.14	97.14	94.49	96.09
83.57	97.14	94.49	96.09
90	97.14	94.49	96.09
96.43	97.14	94.49	96.09
102.86	97.14	94.49	96.09
109.29	97.14	94.49	96.09
115.71	97.14	94.49	96.09
122.14	97.14	94.49	96.09
128.57	97.14	94.49	96.09
135	97.14	94.49	96.09
141.43	97.14	94.49	96.09
147.86	97.14	94.49	96.09
154.29	97.14	94.49	96.09
160.71	97.14	94.49	96.09
167.14	97.14	94.49	96.09
173.57	97.14	94.49	96.09
180	97.14	94.49	96.09

Max	97.14
Min	94.49

c. As a percentage of serial model

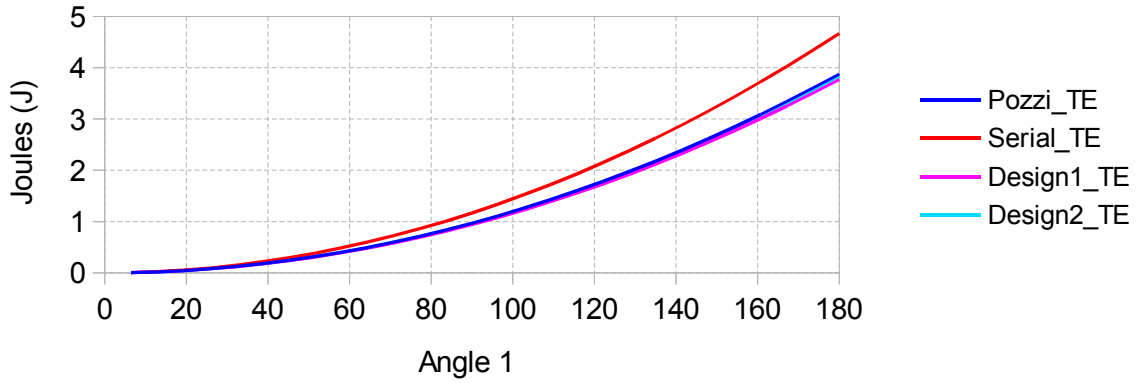


Varying angle 1, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.004942	0.005956	0.004807	0.004888
12.86	0.019769	0.023824	0.019229	0.019554
19.29	0.044479	0.053605	0.043264	0.043995
25.71	0.079074	0.095297	0.076915	0.078214
32.14	0.123554	0.148902	0.120179	0.122209
38.57	0.177917	0.214419	0.173058	0.175981
45	0.242165	0.291847	0.235551	0.239530
51.43	0.316297	0.381188	0.307658	0.312855
57.86	0.400314	0.482442	0.389380	0.395958
64.29	0.494215	0.595607	0.480716	0.488837
70.71	0.598000	0.720684	0.581666	0.591492
77.14	0.711669	0.857674	0.692231	0.703925
83.57	0.835222	1.006576	0.812409	0.826134
90	0.968660	1.167390	0.942203	0.958119
96.43	1.111983	1.340116	1.081610	1.099882
102.86	1.265189	1.524754	1.230632	1.251421
109.29	1.428280	1.721304	1.389268	1.412737
115.71	1.601255	1.929766	1.557519	1.583830
122.14	1.784114	2.150141	1.735384	1.764700
128.57	1.976858	2.382428	1.922863	1.955346
135	2.179486	2.626626	2.119956	2.155769
141.43	2.391998	2.882737	2.326664	2.365968
147.86	2.614395	3.150760	2.542986	2.585945
154.29	2.846675	3.430696	2.768922	2.815698
160.71	3.088840	3.722543	3.004473	3.055228
167.14	3.340890	4.026303	3.249638	3.304534
173.57	3.602823	4.341974	3.504417	3.563618
180	3.874641	4.669558	3.768811	3.832478

Max	4.669558
Min	0.004807

a. Total energy consumed per model

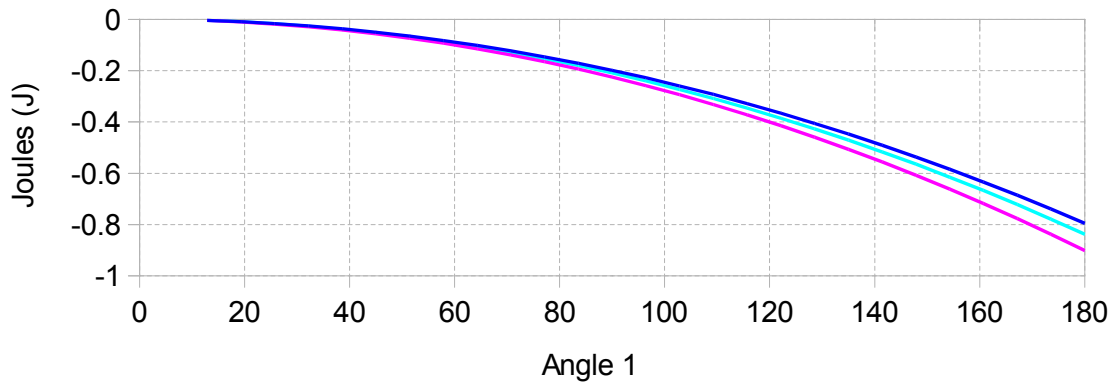


Varying angle 1, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.001014	-0.001149	-0.001068
12.86	-0.004056	-0.004596	-0.004271
19.29	-0.009125	-0.010340	-0.009609
25.71	-0.016223	-0.018383	-0.017083
32.14	-0.025348	-0.028723	-0.026693
38.57	-0.036501	-0.041361	-0.038437
45	-0.049682	-0.056297	-0.052318
51.43	-0.064891	-0.073530	-0.068333
57.86	-0.082128	-0.093062	-0.086484
64.29	-0.101392	-0.114891	-0.106770
70.71	-0.122685	-0.139018	-0.129192
77.14	-0.146005	-0.165443	-0.153749
83.57	-0.171353	-0.194166	-0.180442
90	-0.198729	-0.225187	-0.209270
96.43	-0.228133	-0.258505	-0.240234
102.86	-0.259565	-0.294122	-0.273332
109.29	-0.293024	-0.332036	-0.308567
115.71	-0.328512	-0.372248	-0.345936
122.14	-0.366027	-0.414757	-0.385441
128.57	-0.405570	-0.459565	-0.427082
135	-0.447141	-0.506670	-0.470858
141.43	-0.490739	-0.556074	-0.516769
147.86	-0.536366	-0.607775	-0.564816
154.29	-0.584020	-0.661774	-0.614998
160.71	-0.633703	-0.718070	-0.667315
167.14	-0.685413	-0.776665	-0.721768
173.57	-0.739151	-0.837557	-0.778356
180	-0.794917	-0.900747	-0.837080

Max	-0.001014
Min	-0.900747

b. Difference from serial model

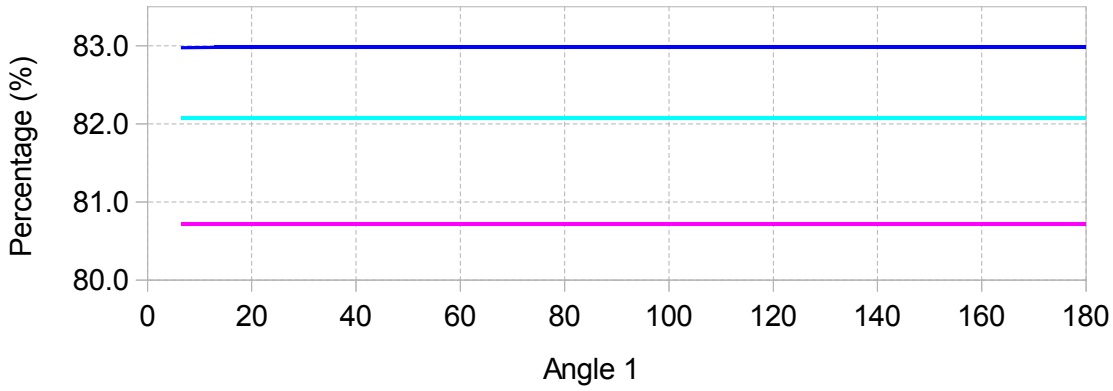


Varying angle 1, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	82.98	80.71	82.07
12.86	82.98	80.71	82.07
19.29	82.98	80.71	82.07
25.71	82.98	80.71	82.07
32.14	82.98	80.71	82.07
38.57	82.98	80.71	82.07
45	82.98	80.71	82.07
51.43	82.98	80.71	82.07
57.86	82.98	80.71	82.07
64.29	82.98	80.71	82.07
70.71	82.98	80.71	82.07
77.14	82.98	80.71	82.07
83.57	82.98	80.71	82.07
90	82.98	80.71	82.07
96.43	82.98	80.71	82.07
102.86	82.98	80.71	82.07
109.29	82.98	80.71	82.07
115.71	82.98	80.71	82.07
122.14	82.98	80.71	82.07
128.57	82.98	80.71	82.07
135	82.98	80.71	82.07
141.43	82.98	80.71	82.07
147.86	82.98	80.71	82.07
154.29	82.98	80.71	82.07
160.71	82.98	80.71	82.07
167.14	82.98	80.71	82.07
173.57	82.98	80.71	82.07
180	82.98	80.71	82.07

Max	82.98
Min	80.71

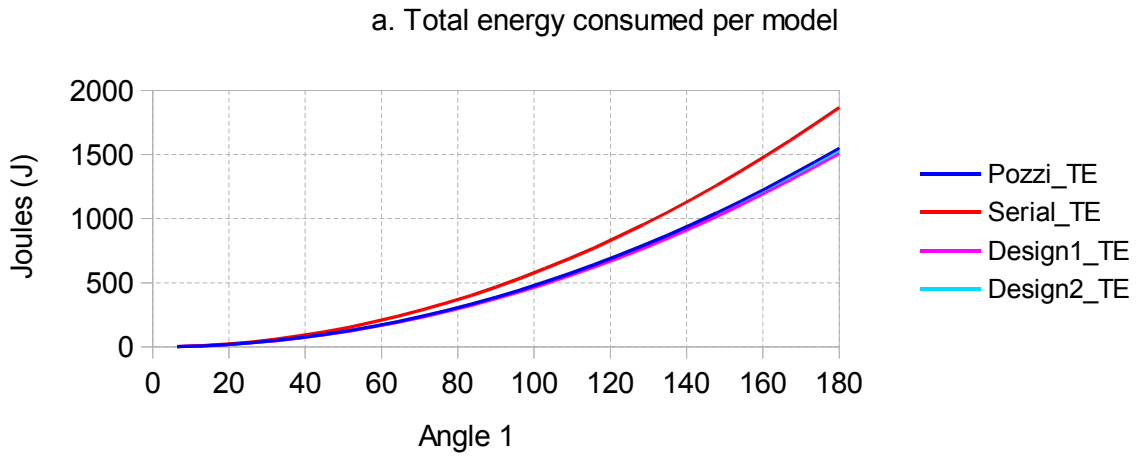
c. As a percentage of serial model



Varying angle 1, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	1.976858	2.382428	1.922863	1.955346
12.86	7.907431	9.529710	7.691450	7.821383
19.29	17.791720	21.441847	17.305763	17.598112
25.71	31.629725	38.118840	30.765801	31.285533
32.14	49.421445	59.560687	48.071564	48.883645
38.57	71.166882	85.767390	69.223053	70.392449
45	96.866033	116.738950	94.220266	95.811945
51.43	126.518900	152.475360	123.063200	125.142130
57.86	160.125480	192.976630	155.751870	158.383010
64.29	197.685780	238.242750	192.286260	195.534580
70.71	239.199800	288.273730	232.666370	236.596840
77.14	284.667530	343.069560	276.892210	281.569800
83.57	334.088970	402.630250	324.963770	330.453440
90	387.464130	466.955790	376.881060	383.247780
96.43	444.793010	536.046190	432.644080	439.952810
102.86	506.075600	609.901440	492.252820	500.568530
109.29	571.311910	688.521550	555.707280	565.094940
115.71	640.501930	771.906510	623.007470	633.532040
122.14	713.645670	860.056330	694.153390	705.879840
128.57	790.743130	952.971000	769.145030	782.138320
135	871.794300	1050.650500	847.982390	862.307500
141.43	956.799180	1153.094900	930.665480	946.387370
147.86	1045.757800	1260.304100	1017.194300	1034.377900
154.29	1138.670100	1372.278200	1107.568800	1126.279200
160.71	1235.536100	1489.017200	1201.789100	1222.091100
167.14	1336.355900	1610.521000	1299.855100	1321.813800
173.57	1441.129400	1736.789600	1401.766800	1425.447100
180	1549.856500	1867.823200	1507.524300	1532.991100

Max	1867.823200
Min	1.922863

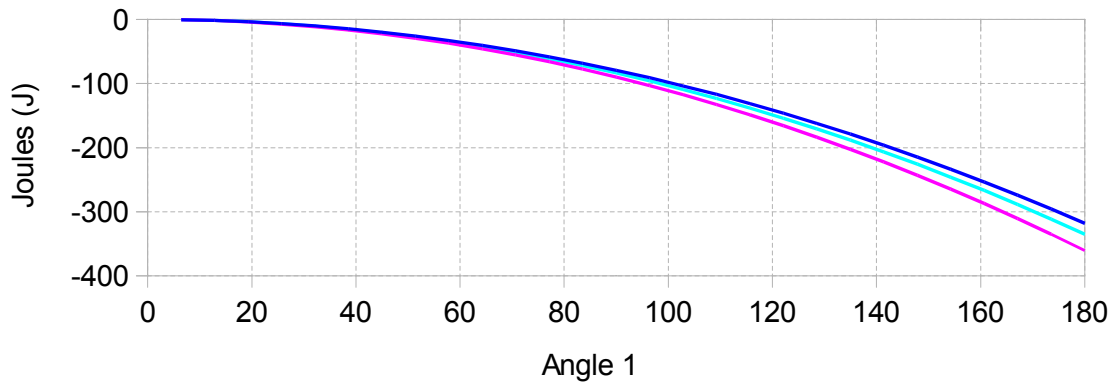


Varying angle 1, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.405570	-0.459565	-0.427082
12.86	-1.622279	-1.838260	-1.708327
19.29	-3.650127	-4.136084	-3.843735
25.71	-6.489115	-7.353039	-6.833307
32.14	-10.139242	-11.489123	-10.677042
38.57	-14.600508	-16.544337	-15.374941
45	-19.872917	-22.518684	-20.927005
51.43	-25.956460	-29.412160	-27.333230
57.86	-32.851150	-37.224760	-34.593620
64.29	-40.556970	-45.956490	-42.708170
70.71	-49.073930	-55.607360	-51.676890
77.14	-58.402030	-66.177350	-61.499760
83.57	-68.541280	-77.666480	-72.176810
90	-79.491660	-90.074730	-83.708010
96.43	-91.253180	-103.402110	-96.093380
102.86	-103.825840	-117.648620	-109.332910
109.29	-117.209640	-132.814270	-123.426610
115.71	-131.404580	-148.899040	-138.374470
122.14	-146.410660	-165.902940	-154.176490
128.57	-162.227870	-183.825970	-170.832680
135	-178.856200	-202.668110	-188.343000
141.43	-196.295720	-222.429420	-206.707530
147.86	-214.546300	-243.109800	-225.926200
154.29	-233.608100	-264.709400	-245.999000
160.71	-253.481100	-287.228100	-266.926100
167.14	-274.165100	-310.665900	-288.707200
173.57	-295.660200	-335.022800	-311.342500
180	-317.966700	-360.298900	-334.832100

Max	-0.405570
Min	-360.298900

b. Difference from serial model

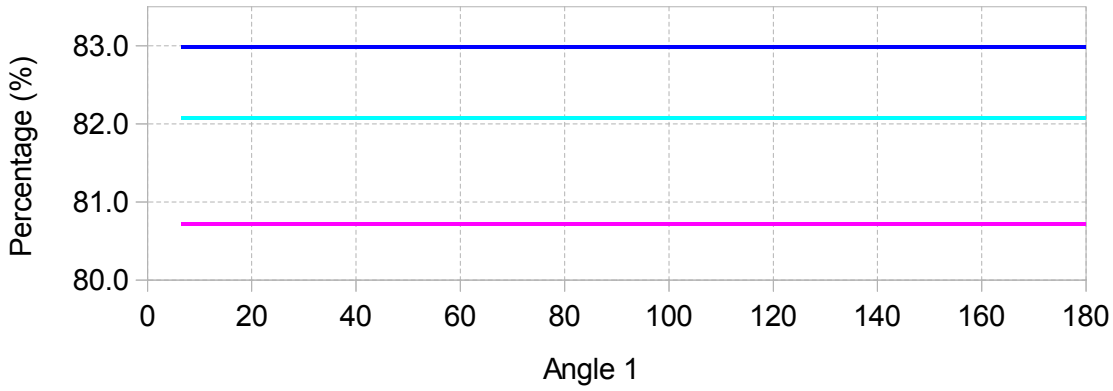


Varying angle 1, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	82.98	80.71	82.07
12.86	82.98	80.71	82.07
19.29	82.98	80.71	82.07
25.71	82.98	80.71	82.07
32.14	82.98	80.71	82.07
38.57	82.98	80.71	82.07
45	82.98	80.71	82.07
51.43	82.98	80.71	82.07
57.86	82.98	80.71	82.07
64.29	82.98	80.71	82.07
70.71	82.98	80.71	82.07
77.14	82.98	80.71	82.07
83.57	82.98	80.71	82.07
90	82.98	80.71	82.07
96.43	82.98	80.71	82.07
102.86	82.98	80.71	82.07
109.29	82.98	80.71	82.07
115.71	82.98	80.71	82.07
122.14	82.98	80.71	82.07
128.57	82.98	80.71	82.07
135	82.98	80.71	82.07
141.43	82.98	80.71	82.07
147.86	82.98	80.71	82.07
154.29	82.98	80.71	82.07
160.71	82.98	80.71	82.07
167.14	82.98	80.71	82.07
173.57	82.98	80.71	82.07
180	82.98	80.71	82.07

Max	82.98
Min	80.71

c. As a percentage of serial model

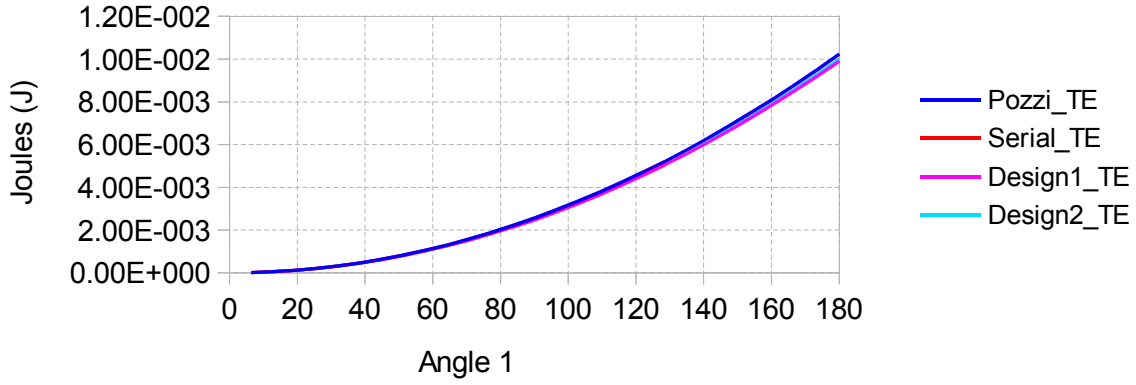


Varying angle 1, angle(2) = 90, light motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.000013	0.000013	0.000013	0.000013
12.86	0.000052	0.000052	0.000051	0.000051
19.29	0.000118	0.000117	0.000114	0.000115
25.71	0.000209	0.000209	0.000202	0.000204
32.14	0.000327	0.000326	0.000316	0.000319
38.57	0.000470	0.000470	0.000455	0.000460
45	0.000640	0.000639	0.000619	0.000626
51.43	0.000836	0.000835	0.000809	0.000817
57.86	0.001058	0.001057	0.001023	0.001034
64.29	0.001307	0.001304	0.001263	0.001277
70.71	0.001581	0.001578	0.001529	0.001545
77.14	0.001882	0.001878	0.001819	0.001838
83.57	0.002208	0.002204	0.002135	0.002157
90	0.002561	0.002557	0.002476	0.002502
96.43	0.002940	0.002935	0.002843	0.002872
102.86	0.003345	0.003339	0.003234	0.003268
109.29	0.003776	0.003770	0.003651	0.003689
115.71	0.004234	0.004226	0.004094	0.004136
122.14	0.004717	0.004709	0.004561	0.004608
128.57	0.005227	0.005218	0.005054	0.005106
135	0.005763	0.005752	0.005572	0.005629
141.43	0.006325	0.006313	0.006115	0.006178
147.86	0.006913	0.006900	0.006684	0.006753
154.29	0.007527	0.007513	0.007277	0.007353
160.71	0.008167	0.008152	0.007896	0.007978
167.14	0.008833	0.008818	0.008541	0.008629
173.57	0.009526	0.009509	0.009210	0.009306
180	0.010245	0.010226	0.009905	0.010008

Max	0.010245
Min	0.000013

a. Total energy consumed per model

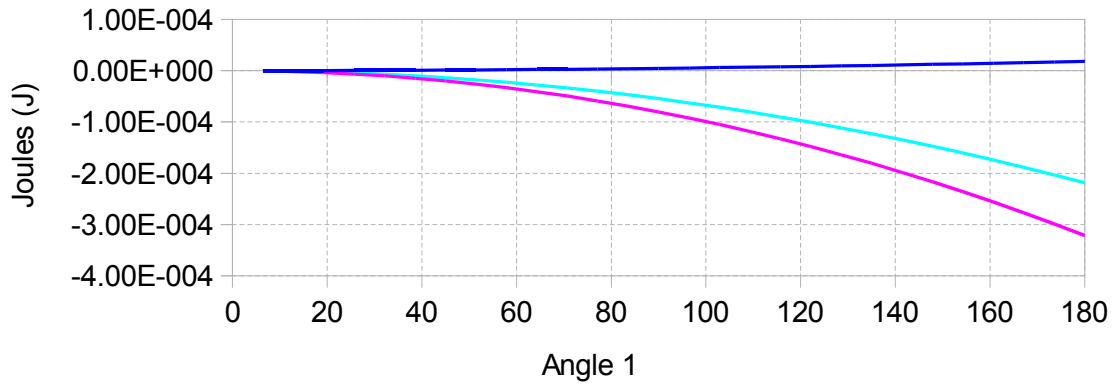


Varying angle 1, angle(2) = 90, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	0.000000	0.000000	0.000000
12.86	0.000000	-0.000002	-0.000001
19.29	0.000000	-0.000004	-0.000002
25.71	0.000000	-0.000007	-0.000004
32.14	0.000001	-0.000010	-0.000007
38.57	0.000001	-0.000015	-0.000010
45	0.000001	-0.000020	-0.000014
51.43	0.000002	-0.000026	-0.000018
57.86	0.000002	-0.000033	-0.000023
64.29	0.000002	-0.000041	-0.000028
70.71	0.000003	-0.000050	-0.000034
77.14	0.000003	-0.000059	-0.000040
83.57	0.000004	-0.000069	-0.000047
90	0.000004	-0.000080	-0.000055
96.43	0.000005	-0.000092	-0.000063
102.86	0.000006	-0.000105	-0.000071
109.29	0.000007	-0.000118	-0.000081
115.71	0.000008	-0.000133	-0.000090
122.14	0.000008	-0.000148	-0.000101
128.57	0.000009	-0.000164	-0.000112
135	0.000010	-0.000181	-0.000123
141.43	0.000011	-0.000198	-0.000135
147.86	0.000012	-0.000217	-0.000148
154.29	0.000013	-0.000236	-0.000161
160.71	0.000015	-0.000256	-0.000174
167.14	0.000016	-0.000277	-0.000189
173.57	0.000017	-0.000299	-0.000203
180	0.000018	-0.000321	-0.000219

Max	0.000018
Min	-0.000321

b. Difference from serial model

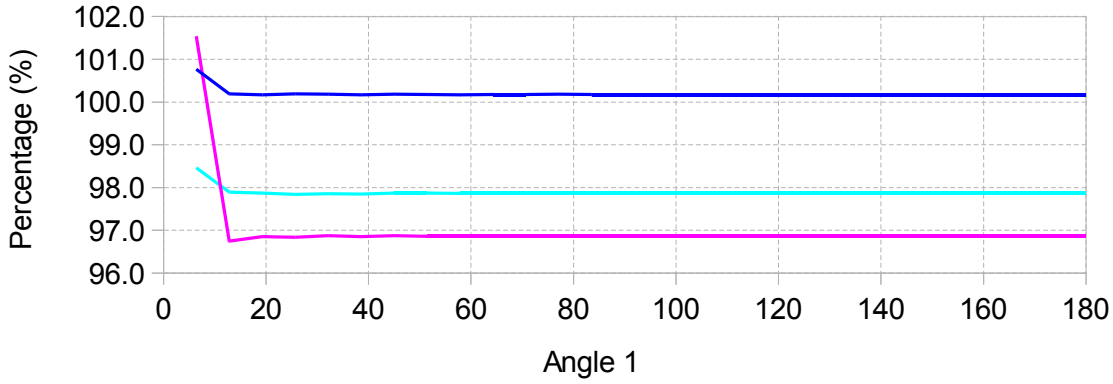


Varying angle 1, angle(2) = 90, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	100.77	101.54	98.46
12.86	100.19	96.74	97.89
19.29	100.17	96.85	97.87
25.71	100.19	96.84	97.84
32.14	100.18	96.87	97.85
38.57	100.17	96.85	97.85
45	100.19	96.87	97.87
51.43	100.18	96.86	97.87
57.86	100.17	96.86	97.86
64.29	100.18	96.86	97.86
70.71	100.18	96.86	97.86
77.14	100.18	96.86	97.87
83.57	100.18	96.86	97.86
90	100.18	96.86	97.86
96.43	100.18	96.86	97.86
102.86	100.18	96.86	97.86
109.29	100.18	96.86	97.86
115.71	100.18	96.86	97.86
122.14	100.18	96.86	97.86
128.57	100.18	96.86	97.86
135	100.18	96.86	97.86
141.43	100.18	96.86	97.86
147.86	100.18	96.86	97.86
154.29	100.18	96.86	97.86
160.71	100.18	96.86	97.86
167.14	100.18	96.86	97.86
173.57	100.18	96.86	97.86
180	100.18	96.86	97.86

Max	101.54
Min	96.74

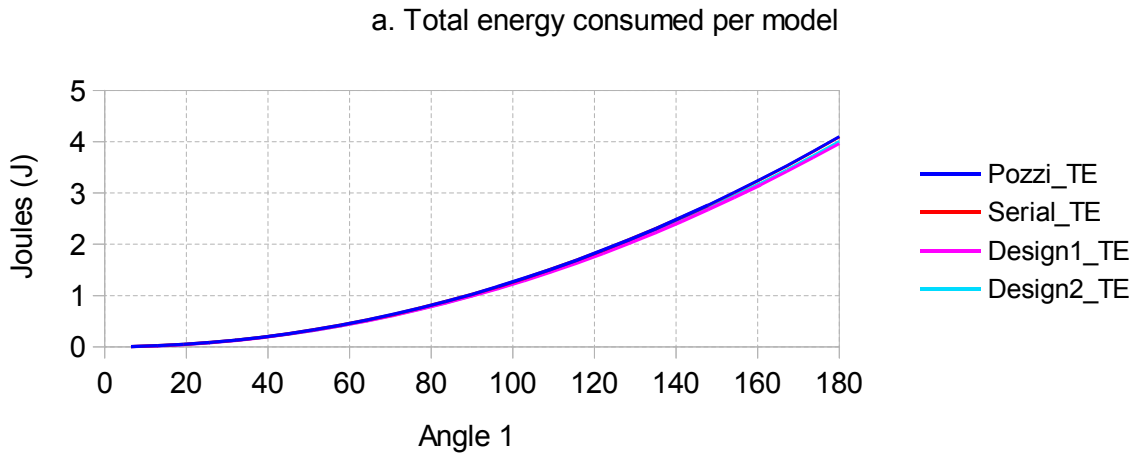
c. As a percentage of serial model



Varying angle 1, angle(2) = 90, light motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.005227	0.005218	0.005054	0.005106
12.86	0.020907	0.020870	0.020215	0.020424
19.29	0.047041	0.046958	0.045483	0.045954
25.71	0.083629	0.083481	0.080859	0.081696
32.14	0.130671	0.130439	0.126343	0.127651
38.57	0.188166	0.187832	0.181933	0.183817
45	0.256114	0.255660	0.247631	0.250195
51.43	0.334517	0.333923	0.323437	0.326786
57.86	0.423373	0.422622	0.409350	0.413588
64.29	0.522683	0.521755	0.505370	0.510602
70.71	0.632446	0.631324	0.611498	0.617829
77.14	0.752663	0.751327	0.727733	0.735267
83.57	0.883334	0.881766	0.854076	0.862918
90	1.024458	1.022640	0.990525	1.000781
96.43	1.176036	1.173949	1.137083	1.148855
102.86	1.338067	1.335693	1.293748	1.307142
109.29	1.510553	1.507872	1.460520	1.475641
115.71	1.693491	1.690487	1.637399	1.654352
122.14	1.886884	1.883536	1.824386	1.843274
128.57	2.090730	2.087020	2.021481	2.042409
135	2.305030	2.300940	2.228682	2.251756
141.43	2.529783	2.525295	2.445991	2.471315
147.86	2.764991	2.760084	2.673408	2.701086
154.29	3.010651	3.005309	2.910932	2.941069
160.71	3.266766	3.260969	3.158563	3.191264
167.14	3.533334	3.527064	3.416302	3.451672
173.57	3.810356	3.803595	3.684148	3.722291
180	4.097831	4.090560	3.962102	4.003122

Max	4.097831
Min	0.005054

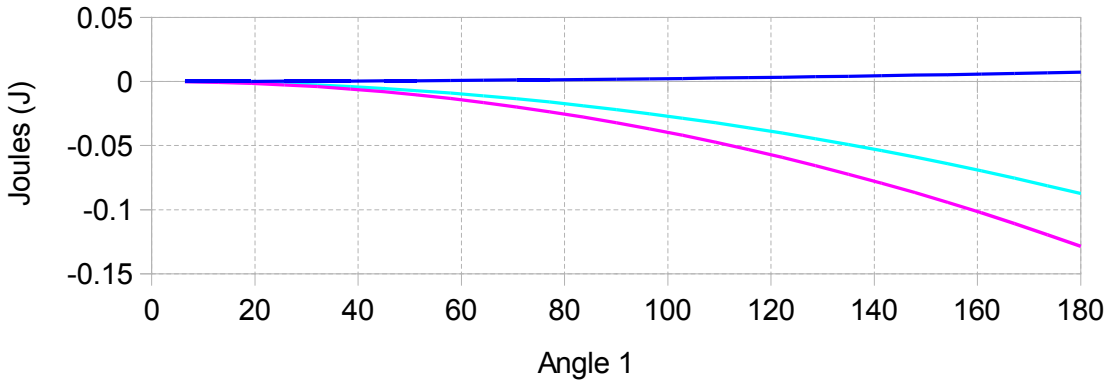


Varying angle 1, angle(2) = 90, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	0.000009	-0.000164	-0.000112
12.86	0.000037	-0.000655	-0.000446
19.29	0.000083	-0.001475	-0.001004
25.71	0.000148	-0.002622	-0.001784
32.14	0.000232	-0.004096	-0.002788
38.57	0.000334	-0.005899	-0.004015
45	0.000454	-0.008029	-0.005465
51.43	0.000594	-0.010486	-0.007138
57.86	0.000751	-0.013272	-0.009034
64.29	0.000927	-0.016385	-0.011153
70.71	0.001122	-0.019826	-0.013495
77.14	0.001336	-0.023594	-0.016060
83.57	0.001567	-0.027691	-0.018848
90	0.001818	-0.032115	-0.021860
96.43	0.002087	-0.036866	-0.025094
102.86	0.002374	-0.041946	-0.028551
109.29	0.002680	-0.047353	-0.032232
115.71	0.003005	-0.053087	-0.036135
122.14	0.003348	-0.059150	-0.040262
128.57	0.003710	-0.065540	-0.044611
135	0.004090	-0.072258	-0.049184
141.43	0.004489	-0.079303	-0.053980
147.86	0.004906	-0.086677	-0.058998
154.29	0.005342	-0.094377	-0.064240
160.71	0.005796	-0.102406	-0.069705
167.14	0.006269	-0.110762	-0.075393
173.57	0.006761	-0.119447	-0.081304
180	0.007271	-0.128458	-0.087438

Max	0.007271
Min	-0.128458

b. Difference from serial model

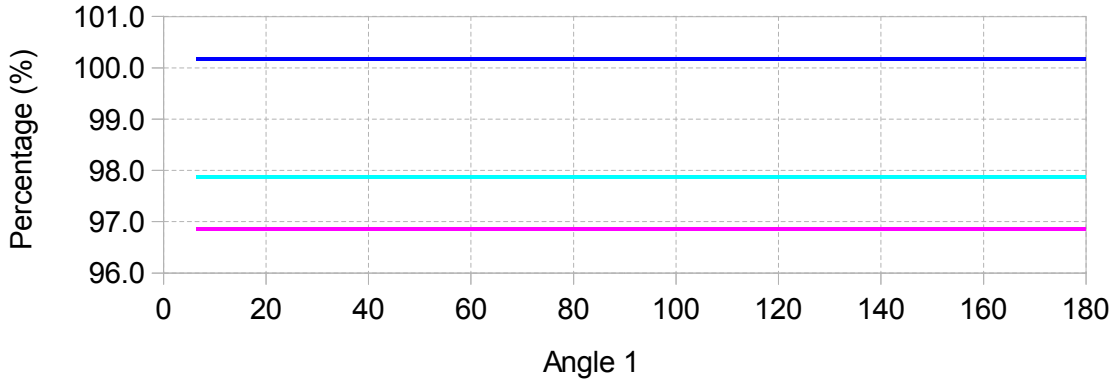


Varying angle 1, angle(2) = 90, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	100.18	96.86	97.86
12.86	100.18	96.86	97.86
19.29	100.18	96.86	97.86
25.71	100.18	96.86	97.86
32.14	100.18	96.86	97.86
38.57	100.18	96.86	97.86
45	100.18	96.86	97.86
51.43	100.18	96.86	97.86
57.86	100.18	96.86	97.86
64.29	100.18	96.86	97.86
70.71	100.18	96.86	97.86
77.14	100.18	96.86	97.86
83.57	100.18	96.86	97.86
90	100.18	96.86	97.86
96.43	100.18	96.86	97.86
102.86	100.18	96.86	97.86
109.29	100.18	96.86	97.86
115.71	100.18	96.86	97.86
122.14	100.18	96.86	97.86
128.57	100.18	96.86	97.86
135	100.18	96.86	97.86
141.43	100.18	96.86	97.86
147.86	100.18	96.86	97.86
154.29	100.18	96.86	97.86
160.71	100.18	96.86	97.86
167.14	100.18	96.86	97.86
173.57	100.18	96.86	97.86
180	100.18	96.86	97.86

Max	100.18
Min	96.86

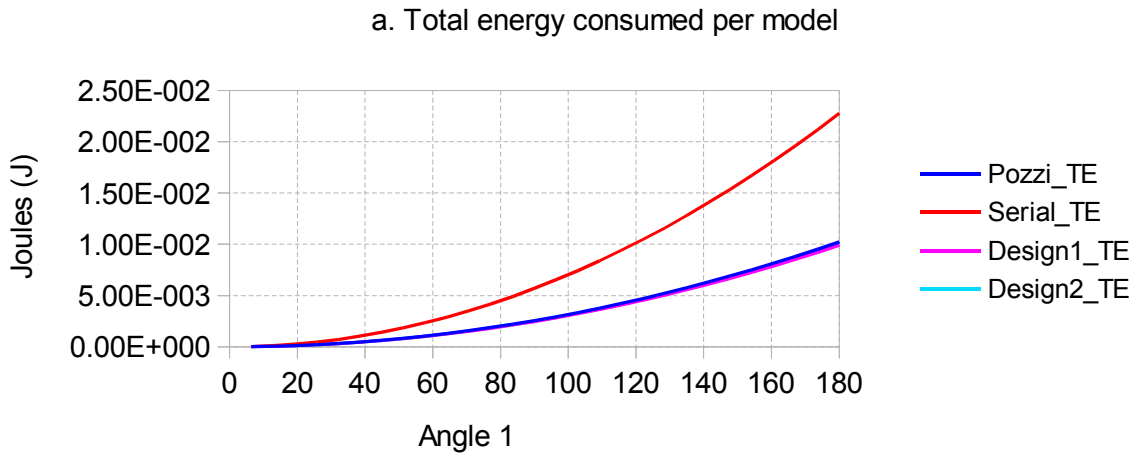
c. As a percentage of serial model



Varying angle 1, angle(2) = 90, heavy motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.000013	0.000029	0.000013	0.000013
12.86	0.000052	0.000116	0.000051	0.000051
19.29	0.000118	0.000261	0.000114	0.000115
25.71	0.000209	0.000465	0.000202	0.000204
32.14	0.000327	0.000726	0.000316	0.000319
38.57	0.000470	0.001045	0.000455	0.000460
45	0.000640	0.001423	0.000619	0.000626
51.43	0.000836	0.001858	0.000809	0.000817
57.86	0.001058	0.002352	0.001023	0.001034
64.29	0.001307	0.002903	0.001263	0.001277
70.71	0.001581	0.003513	0.001529	0.001545
77.14	0.001882	0.004181	0.001819	0.001838
83.57	0.002208	0.004906	0.002135	0.002157
90	0.002561	0.005690	0.002476	0.002502
96.43	0.002940	0.006532	0.002843	0.002872
102.86	0.003345	0.007432	0.003234	0.003268
109.29	0.003776	0.008390	0.003651	0.003689
115.71	0.004234	0.009406	0.004094	0.004136
122.14	0.004717	0.010481	0.004561	0.004608
128.57	0.005227	0.011613	0.005054	0.005106
135	0.005763	0.012803	0.005572	0.005629
141.43	0.006325	0.014051	0.006115	0.006178
147.86	0.006913	0.015358	0.006684	0.006753
154.29	0.007527	0.016722	0.007277	0.007353
160.71	0.008167	0.018145	0.007896	0.007978
167.14	0.008833	0.019626	0.008541	0.008629
173.57	0.009526	0.021164	0.009210	0.009306
180	0.010245	0.022761	0.009905	0.010008

Max	0.022761
Min	0.000013

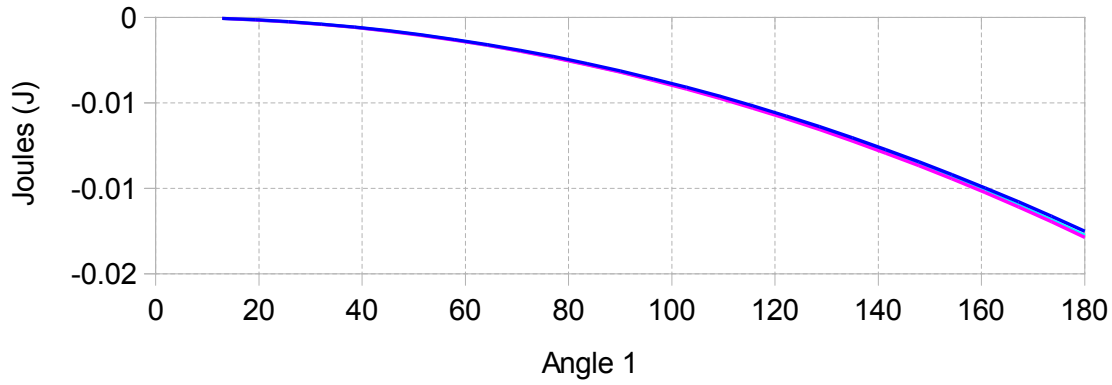


Varying angle 1, angle(2) = 90, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.000016	-0.000016	-0.000016
12.86	-0.000064	-0.000066	-0.000065
19.29	-0.000144	-0.000148	-0.000146
25.71	-0.000255	-0.000262	-0.000260
32.14	-0.000399	-0.000410	-0.000407
38.57	-0.000575	-0.000590	-0.000586
45	-0.000782	-0.000804	-0.000797
51.43	-0.001022	-0.001049	-0.001041
57.86	-0.001293	-0.001328	-0.001318
64.29	-0.001597	-0.001640	-0.001627
70.71	-0.001932	-0.001984	-0.001968
77.14	-0.002299	-0.002361	-0.002342
83.57	-0.002698	-0.002771	-0.002749
90	-0.003129	-0.003214	-0.003188
96.43	-0.003592	-0.003690	-0.003660
102.86	-0.004087	-0.004198	-0.004164
109.29	-0.004614	-0.004739	-0.004701
115.71	-0.005173	-0.005313	-0.005270
122.14	-0.005763	-0.005920	-0.005872
128.57	-0.006386	-0.006559	-0.006507
135	-0.007040	-0.007231	-0.007174
141.43	-0.007727	-0.007936	-0.007873
147.86	-0.008445	-0.008674	-0.008605
154.29	-0.009196	-0.009445	-0.009370
160.71	-0.009978	-0.010249	-0.010167
167.14	-0.010792	-0.011085	-0.010996
173.57	-0.011638	-0.011954	-0.011859
180	-0.012516	-0.012856	-0.012753

Max	-0.000016
Min	-0.012856

b. Difference from serial model

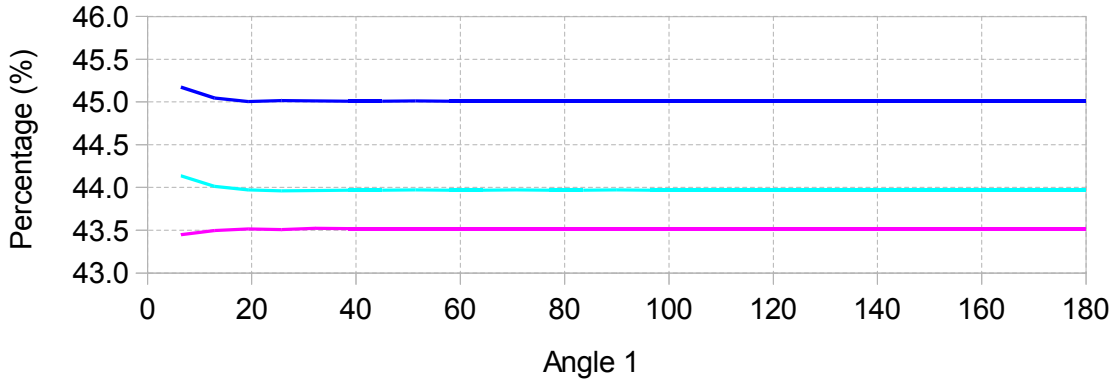


Varying angle 1, angle(2) = 90, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	45.17	43.45	44.14
12.86	45.05	43.5	44.01
19.29	45.01	43.51	43.97
25.71	45.02	43.51	43.96
32.14	45.01	43.52	43.97
38.57	45.01	43.52	43.97
45	45.01	43.52	43.97
51.43	45.01	43.52	43.97
57.86	45.01	43.52	43.97
64.29	45.01	43.52	43.97
70.71	45.01	43.52	43.97
77.14	45.01	43.52	43.97
83.57	45.01	43.52	43.97
90	45.01	43.52	43.97
96.43	45.01	43.52	43.97
102.86	45.01	43.52	43.97
109.29	45.01	43.52	43.97
115.71	45.01	43.52	43.97
122.14	45.01	43.52	43.97
128.57	45.01	43.52	43.97
135	45.01	43.52	43.97
141.43	45.01	43.52	43.97
147.86	45.01	43.52	43.97
154.29	45.01	43.52	43.97
160.71	45.01	43.52	43.97
167.14	45.01	43.52	43.97
173.57	45.01	43.52	43.97
180	45.01	43.52	43.97

Max	45.17
Min	43.45

c. As a percentage of serial model

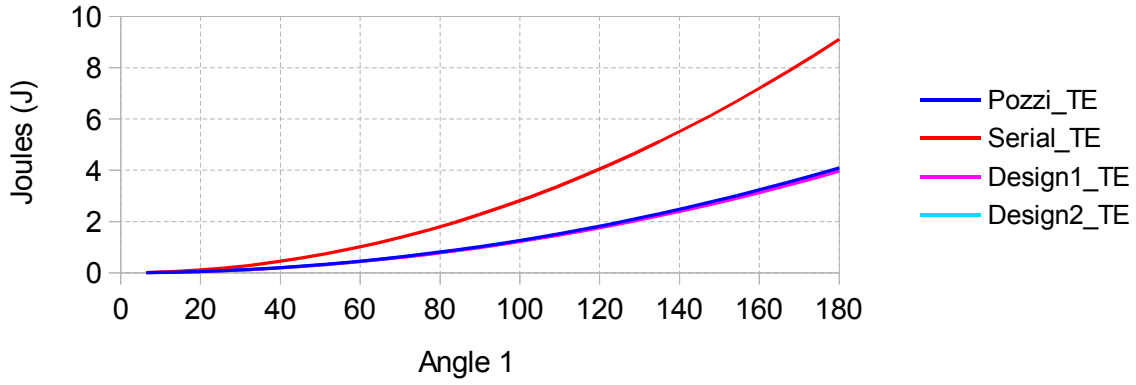


Varying angle 1, angle(2) = 90, heavy motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
6.43	0.005227	0.011613	0.005054	0.005106
12.86	0.020907	0.046451	0.020215	0.020424
19.29	0.047041	0.104515	0.045483	0.045954
25.71	0.083629	0.185804	0.080859	0.081696
32.14	0.130671	0.290318	0.126343	0.127651
38.57	0.188166	0.418058	0.181933	0.183817
45	0.256114	0.569023	0.247631	0.250195
51.43	0.334517	0.743214	0.323437	0.326786
57.86	0.423373	0.940630	0.409350	0.413588
64.29	0.522683	1.161272	0.505370	0.510602
70.71	0.632446	1.405139	0.611498	0.617829
77.14	0.752663	1.672231	0.727733	0.735267
83.57	0.883334	1.962549	0.854076	0.862918
90	1.024458	2.276093	0.990525	1.000781
96.43	1.176036	2.612861	1.137083	1.148855
102.86	1.338067	2.972856	1.293748	1.307142
109.29	1.510553	3.356075	1.460520	1.475641
115.71	1.693491	3.762520	1.637399	1.654352
122.14	1.886884	4.192191	1.824386	1.843274
128.57	2.090730	4.645087	2.021481	2.042409
135	2.305030	5.121208	2.228682	2.251756
141.43	2.529783	5.620555	2.445991	2.471315
147.86	2.764991	6.143127	2.673408	2.701086
154.29	3.010651	6.688925	2.910932	2.941069
160.71	3.266766	7.257948	3.158563	3.191264
167.14	3.533334	7.850197	3.416302	3.451672
173.57	3.810356	8.465671	3.684148	3.722291
180	4.097831	9.104370	3.962102	4.003122

Max	9.104370
Min	0.005054

a. Total energy consumed per model

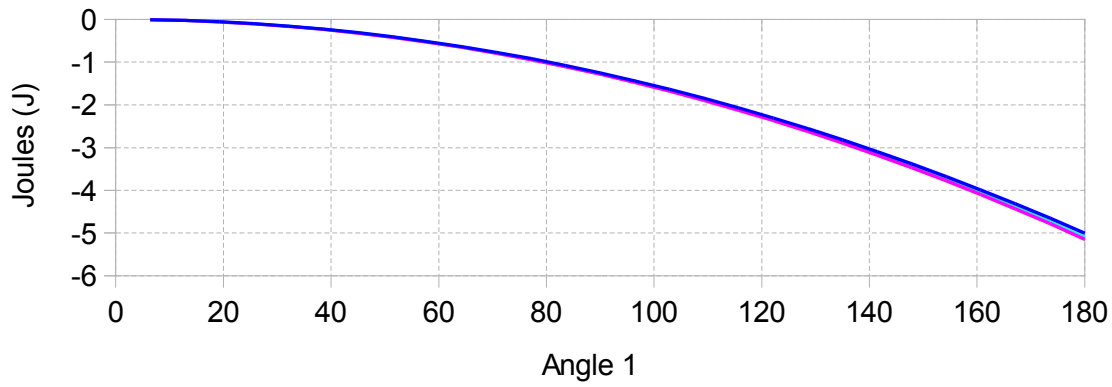


Varying angle 1, angle(2) = 90, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	-0.006386	-0.006559	-0.006507
12.86	-0.025544	-0.026236	-0.026027
19.29	-0.057473	-0.059031	-0.058560
25.71	-0.102174	-0.104944	-0.104107
32.14	-0.159647	-0.163975	-0.162667
38.57	-0.229892	-0.236125	-0.234241
45	-0.312909	-0.321392	-0.318828
51.43	-0.408697	-0.419777	-0.416428
57.86	-0.517257	-0.531280	-0.527042
64.29	-0.638589	-0.655902	-0.650669
70.71	-0.772693	-0.793641	-0.787310
77.14	-0.919568	-0.944498	-0.936964
83.57	-1.079216	-1.108474	-1.099631
90	-1.251635	-1.285567	-1.275312
96.43	-1.436826	-1.475779	-1.464006
102.86	-1.634788	-1.679108	-1.665714
109.29	-1.845523	-1.895556	-1.880435
115.71	-2.069029	-2.125121	-2.108169
122.14	-2.305307	-2.367805	-2.348917
128.57	-2.554357	-2.623606	-2.602678
135	-2.816178	-2.892526	-2.869452
141.43	-3.090772	-3.174564	-3.149240
147.86	-3.378137	-3.469719	-3.442041
154.29	-3.678274	-3.777993	-3.747856
160.71	-3.991182	-4.099385	-4.066684
167.14	-4.316863	-4.433895	-4.398525
173.57	-4.655315	-4.781523	-4.743380
180	-5.006539	-5.142268	-5.101248

Max	-0.006386
Min	-5.142268

b. Difference from serial model

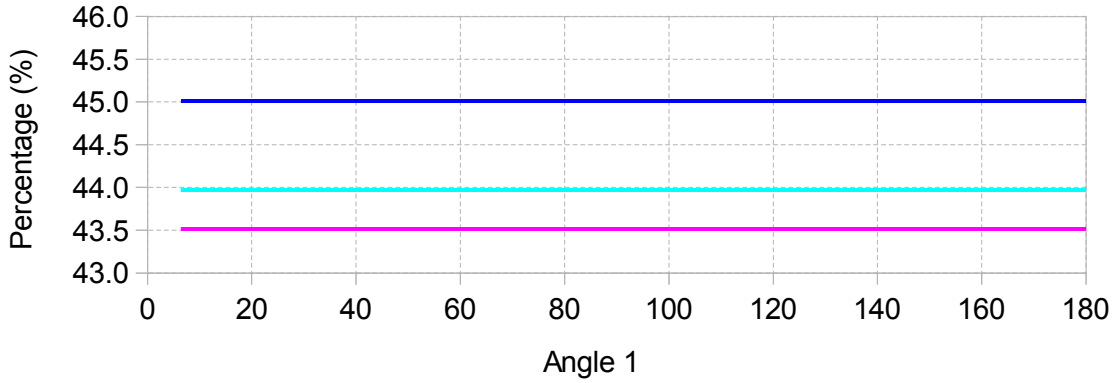


Varying angle 1, angle(2) = 90, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
6.43	45.01	43.52	43.97
12.86	45.01	43.52	43.97
19.29	45.01	43.52	43.97
25.71	45.01	43.52	43.97
32.14	45.01	43.52	43.97
38.57	45.01	43.52	43.97
45	45.01	43.52	43.97
51.43	45.01	43.52	43.97
57.86	45.01	43.52	43.97
64.29	45.01	43.52	43.97
70.71	45.01	43.52	43.97
77.14	45.01	43.52	43.97
83.57	45.01	43.52	43.97
90	45.01	43.52	43.97
96.43	45.01	43.52	43.97
102.86	45.01	43.52	43.97
109.29	45.01	43.52	43.97
115.71	45.01	43.52	43.97
122.14	45.01	43.52	43.97
128.57	45.01	43.52	43.97
135	45.01	43.52	43.97
141.43	45.01	43.52	43.97
147.86	45.01	43.52	43.97
154.29	45.01	43.52	43.97
160.71	45.01	43.52	43.97
167.14	45.01	43.52	43.97
173.57	45.01	43.52	43.97
180	45.01	43.52	43.97

Max	45.01
Min	43.52

c. As a percentage of serial model

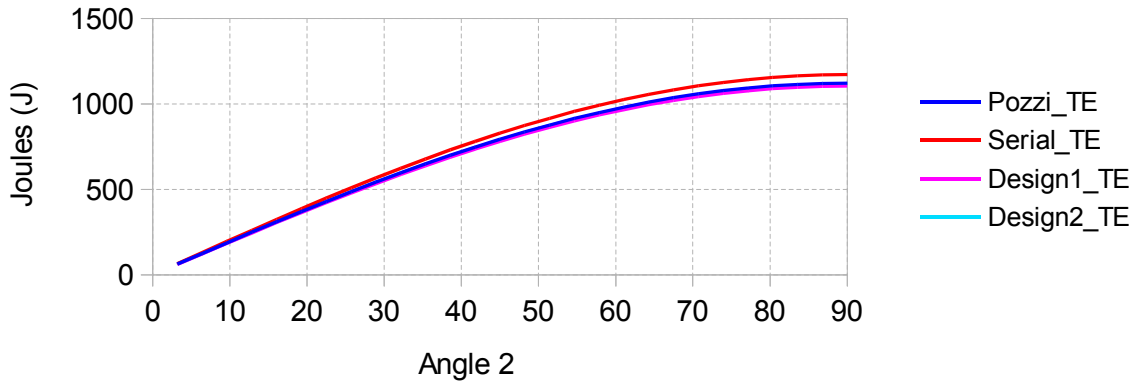


Varying angle 2, angle(3) = 0, light motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	62.876606	65.708679	61.914316	62.640612
6.43	125.555210	131.210440	123.633660	125.083960
9.64	187.838470	196.299040	184.963710	187.133460
12.86	249.530300	260.769590	245.711380	248.593740
16.07	310.436470	324.419080	305.685420	309.271310
19.29	370.365240	387.047150	364.697020	368.975150
22.5	429.127920	448.456610	422.560380	427.517290
25.71	486.539530	508.454150	479.093340	484.713420
28.93	542.419330	566.850870	534.117930	540.383480
32.14	596.591400	623.462950	587.460930	594.352230
35.36	648.885200	678.112170	638.954410	646.449760
38.57	699.136130	730.626500	688.436290	696.512090
41.79	747.186000	780.840630	735.750790	744.381620
45	792.883580	828.596520	780.749000	789.907690
48.21	836.085030	873.743840	823.289290	832.946990
51.43	876.654380	916.140520	863.237760	873.364080
54.64	914.463970	955.653130	900.468700	911.031760
57.86	949.394810	992.157330	934.864950	945.831500
61.07	981.336990	1025.538300	966.318290	977.653800
64.29	1010.190000	1055.690900	994.729750	1006.398500
67.5	1035.863100	1082.520400	1020.010000	1031.975300
70.71	1058.275600	1105.942400	1042.079400	1054.303700
73.93	1077.356900	1125.883200	1060.868700	1073.313400
77.14	1093.047200	1142.280100	1076.318900	1088.944700
80.36	1105.297000	1155.081700	1088.381200	1101.148600
83.57	1114.068000	1164.247800	1097.018000	1109.886700
86.79	1119.332600	1169.749600	1102.202100	1115.131500
90	1121.074400	1171.569800	1103.917200	1116.866800

Max	1171.569800
Min	61.914316

a. Total energy consumed per model

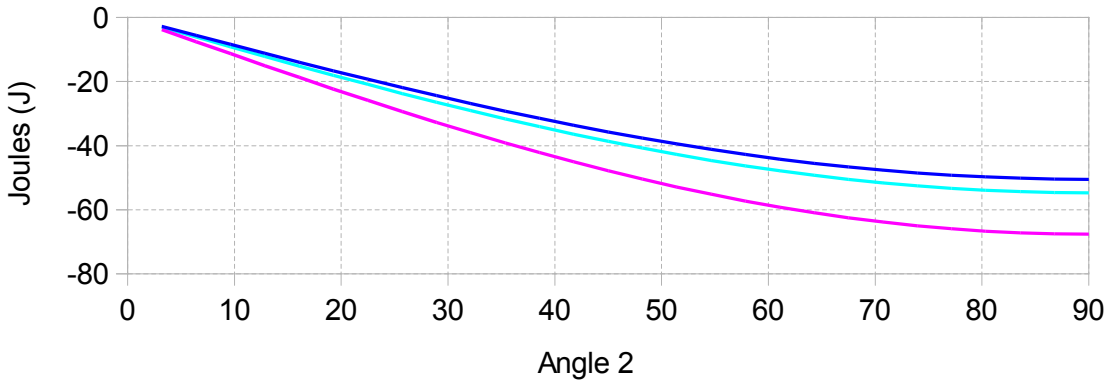


Varying angle 2, angle(3) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-2.832073	-3.794363	-3.068067
6.43	-5.655230	-7.576780	-6.126480
9.64	-8.460570	-11.335330	-9.165580
12.86	-11.239290	-15.058210	-12.175850
16.07	-13.982610	-18.733660	-15.147770
19.29	-16.681910	-22.350130	-18.072000
22.5	-19.328690	-25.896230	-20.939320
25.71	-21.914620	-29.360810	-23.740730
28.93	-24.431540	-32.732940	-26.467390
32.14	-26.871550	-36.002020	-29.110720
35.36	-29.226970	-39.157760	-31.662410
38.57	-31.490370	-42.190210	-34.114410
41.79	-33.654630	-45.089840	-36.459010
45	-35.712940	-47.847520	-38.688830
48.21	-37.658810	-50.454550	-40.796850
51.43	-39.486140	-52.902760	-42.776440
54.64	-41.189160	-55.184430	-44.621370
57.86	-42.762520	-57.292380	-46.325830
61.07	-44.201310	-59.220010	-47.884500
64.29	-45.500900	-60.961150	-49.292400
67.5	-46.657300	-62.510400	-50.545100
70.71	-47.666800	-63.863000	-51.638700
73.93	-48.526300	-65.014500	-52.569800
77.14	-49.232900	-65.961200	-53.335400
80.36	-49.784700	-66.700500	-53.933100
83.57	-50.179800	-67.229800	-54.361100
86.79	-50.417000	-67.547500	-54.618100
90	-50.495400	-67.652600	-54.703000

Max	-2.832073
Min	-67.652600

b. Difference from serial model

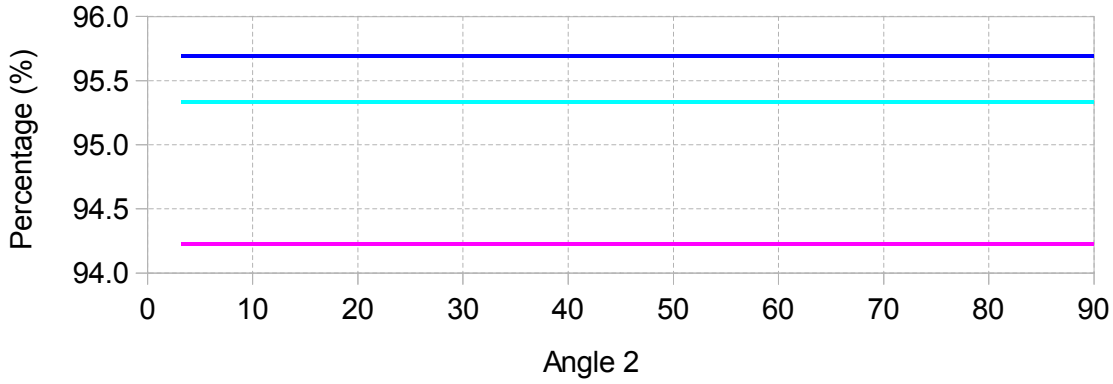


Varying angle 2, angle(3) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	95.69	94.23	95.33
6.43	95.69	94.23	95.33
9.64	95.69	94.23	95.33
12.86	95.69	94.23	95.33
16.07	95.69	94.23	95.33
19.29	95.69	94.23	95.33
22.5	95.69	94.23	95.33
25.71	95.69	94.23	95.33
28.93	95.69	94.23	95.33
32.14	95.69	94.23	95.33
35.36	95.69	94.23	95.33
38.57	95.69	94.23	95.33
41.79	95.69	94.23	95.33
45	95.69	94.23	95.33
48.21	95.69	94.23	95.33
51.43	95.69	94.23	95.33
54.64	95.69	94.23	95.33
57.86	95.69	94.23	95.33
61.07	95.69	94.23	95.33
64.29	95.69	94.23	95.33
67.5	95.69	94.23	95.33
70.71	95.69	94.23	95.33
73.93	95.69	94.23	95.33
77.14	95.69	94.23	95.33
80.36	95.69	94.23	95.33
83.57	95.69	94.23	95.33
86.79	95.69	94.23	95.33
90	95.69	94.23	95.33

Max	95.69
Min	94.23

c. As a percentage of serial model

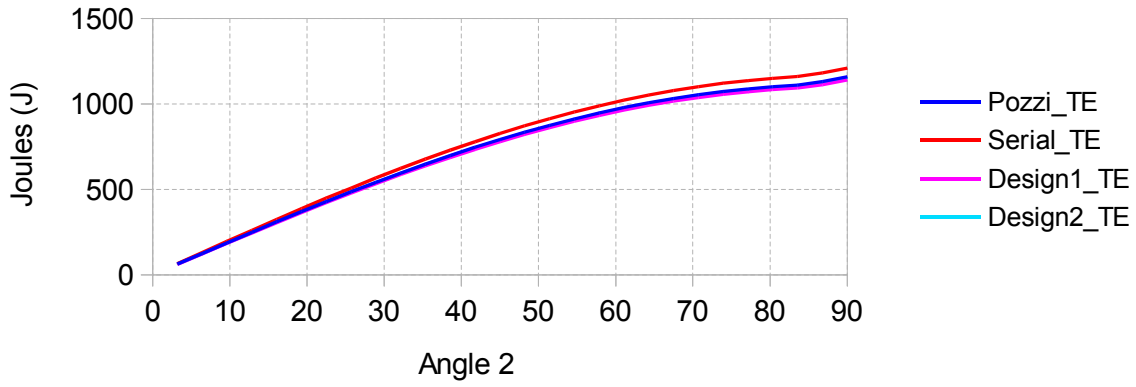


Varying angle 2, angle(3) = 0, light motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	62.867391	65.699189	61.905354	62.631499
6.43	125.518340	131.172470	123.597810	125.047510
9.64	187.755530	196.213630	184.883060	187.051440
12.86	249.382850	260.617740	245.568000	248.447930
16.07	310.206080	324.181830	305.461390	309.043480
19.29	370.033470	386.705500	364.374410	368.647080
22.5	428.676350	447.991600	422.121270	427.070740
25.71	485.949730	507.846780	478.519810	484.130170
28.93	541.672850	566.082170	533.392060	539.645310
32.14	595.669820	622.513940	586.564790	593.440910
35.36	647.770100	676.963860	637.870090	645.347060
38.57	697.809060	729.259920	687.145850	695.199780
41.79	745.628540	779.236800	734.236320	742.841490
45	791.077300	826.736450	778.992570	788.121500
48.21	834.011490	871.608560	821.272970	830.896520
51.43	874.295160	913.711050	860.943640	871.031100
54.64	911.800620	952.910480	897.878860	908.398040
57.86	946.408910	989.082520	931.961460	942.878820
61.07	978.010110	1022.112300	963.083220	974.363930
64.29	1006.503700	1051.894800	991.145190	1002.753300
67.5	1031.799000	1078.335200	1016.058000	1027.956400
70.71	1053.815200	1101.349100	1037.742100	1049.892900
73.93	1072.481800	1120.862900	1056.128200	1068.492500
77.14	1087.738900	1136.813800	1071.157100	1083.695500
80.36	1099.537200	1149.150400	1082.780400	1095.452800
83.57	1110.723800	1160.466700	1093.494400	1106.416800
86.79	1130.235000	1180.217400	1112.192200	1125.546200
90	1159.038400	1209.806000	1140.142400	1153.994200

Max	1209.806000
Min	61.905354

a. Total energy consumed per model

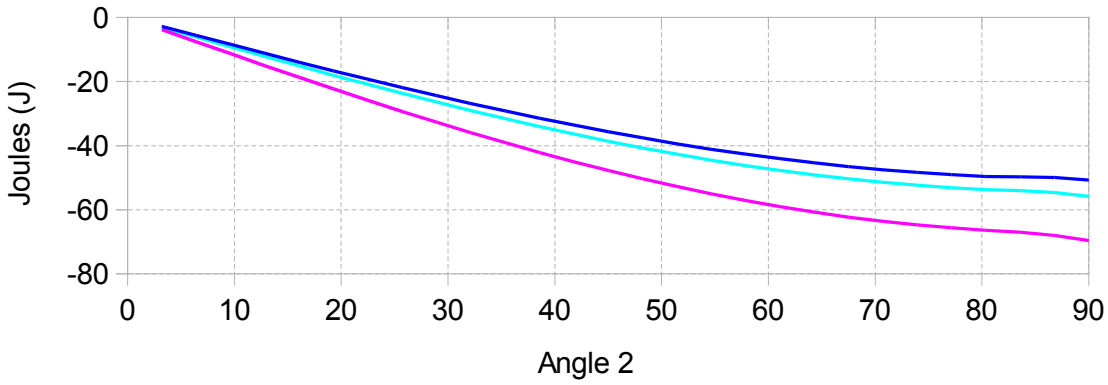


Varying angle 2, angle(3) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-2.831798	-3.793835	-3.067690
6.43	-5.654130	-7.574660	-6.124960
9.64	-8.458100	-11.330570	-9.162190
12.86	-11.234890	-15.049740	-12.169810
16.07	-13.975750	-18.720440	-15.138350
19.29	-16.672030	-22.331090	-18.058420
22.5	-19.315250	-25.870330	-20.920860
25.71	-21.897050	-29.326970	-23.716610
28.93	-24.409320	-32.690110	-26.436860
32.14	-26.844120	-35.949150	-29.073030
35.36	-29.193760	-39.093770	-31.616800
38.57	-31.450860	-42.114070	-34.060140
41.79	-33.608260	-45.000480	-36.395310
45	-35.659150	-47.743880	-38.614950
48.21	-37.597070	-50.335590	-40.712040
51.43	-39.415890	-52.767410	-42.679950
54.64	-41.109860	-55.031620	-44.512440
57.86	-42.673610	-57.121060	-46.203700
61.07	-44.102190	-59.029080	-47.748370
64.29	-45.391100	-60.749610	-49.141500
67.5	-46.536200	-62.277200	-50.378800
70.71	-47.533900	-63.607000	-51.456200
73.93	-48.381100	-64.734700	-52.370400
77.14	-49.074900	-65.656700	-53.118300
80.36	-49.613200	-66.370000	-53.697600
83.57	-49.742900	-66.972300	-54.049900
86.79	-49.982400	-68.025200	-54.671200
90	-50.767600	-69.663600	-55.811800

Max	-2.831798
Min	-69.663600

b. Difference from serial model

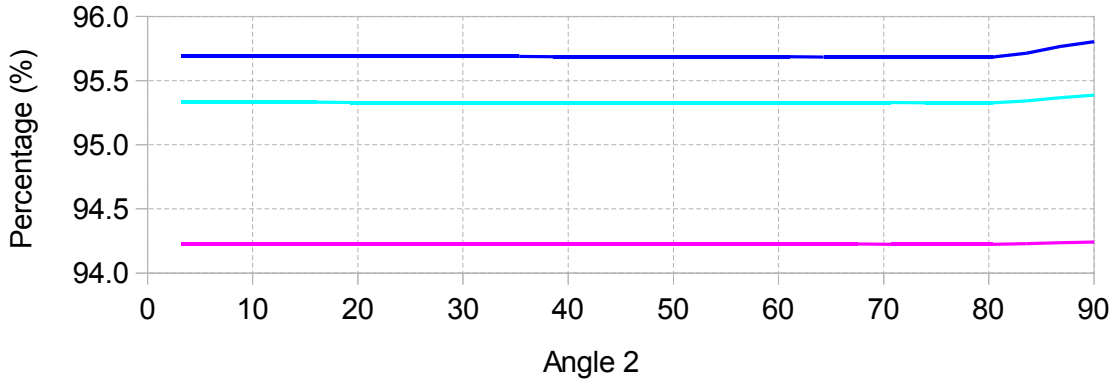


Varying angle 2, angle(3) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	95.69	94.23	95.33
6.43	95.69	94.23	95.33
9.64	95.69	94.23	95.33
12.86	95.69	94.23	95.33
16.07	95.69	94.23	95.33
19.29	95.69	94.23	95.33
22.5	95.69	94.23	95.33
25.71	95.69	94.23	95.33
28.93	95.69	94.23	95.33
32.14	95.69	94.23	95.33
35.36	95.69	94.23	95.33
38.57	95.69	94.23	95.33
41.79	95.69	94.23	95.33
45	95.69	94.23	95.33
48.21	95.69	94.22	95.33
51.43	95.69	94.22	95.33
54.64	95.69	94.22	95.33
57.86	95.69	94.22	95.33
61.07	95.69	94.22	95.33
64.29	95.68	94.22	95.33
67.5	95.68	94.22	95.33
70.71	95.68	94.22	95.33
73.93	95.68	94.22	95.33
77.14	95.68	94.22	95.33
80.36	95.68	94.22	95.33
83.57	95.71	94.23	95.34
86.79	95.76	94.24	95.37
90	95.8	94.24	95.39

Max	95.8
Min	94.22

c. As a percentage of serial model

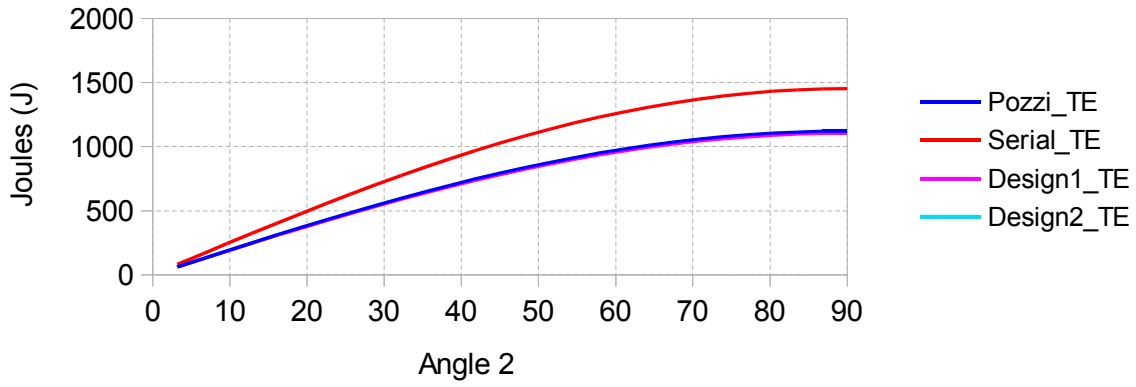


Varying angle 2, angle(3) = 0, heavy motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	62.876606	81.421803	61.914316	62.640612
6.43	125.555210	162.587200	123.633660	125.083960
9.64	187.838470	243.240670	184.963710	187.133460
12.86	249.530300	323.128260	245.711380	248.593740
16.07	310.436470	401.998480	305.685420	309.271310
19.29	370.365240	479.603010	364.697020	368.975150
22.5	429.127920	555.697540	422.560380	427.517290
25.71	486.539530	630.042500	479.093340	484.713420
28.93	542.419330	702.403850	534.117930	540.383480
32.14	596.591400	772.553780	587.460930	594.352230
35.36	648.885200	840.271470	638.954410	646.449760
38.57	699.136130	905.343760	688.436290	696.512090
41.79	747.186000	967.565800	735.750790	744.381620
45	792.883580	1026.741800	780.749000	789.907690
48.21	836.085030	1082.685400	823.289290	832.946990
51.43	876.654380	1135.220500	863.237760	873.364080
54.64	914.463970	1184.182000	900.468700	911.031760
57.86	949.394810	1229.415600	934.864950	945.831500
61.07	981.336990	1270.779100	966.318290	977.653800
64.29	1010.190000	1308.142300	994.729750	1006.398500
67.5	1035.863100	1341.387700	1020.010000	1031.975300
70.71	1058.275600	1370.410700	1042.079400	1054.303700
73.93	1077.356900	1395.120100	1060.868700	1073.313400
77.14	1093.047200	1415.438200	1076.318900	1088.944700
80.36	1105.297000	1431.301200	1088.381200	1101.148600
83.57	1114.068000	1442.659300	1097.018000	1109.886700
86.79	1119.332600	1449.476800	1102.202100	1115.131500
90	1121.074400	1451.732500	1103.917200	1116.866800

Max 1451.732500
Min 61.914316

a. Total energy consumed per model

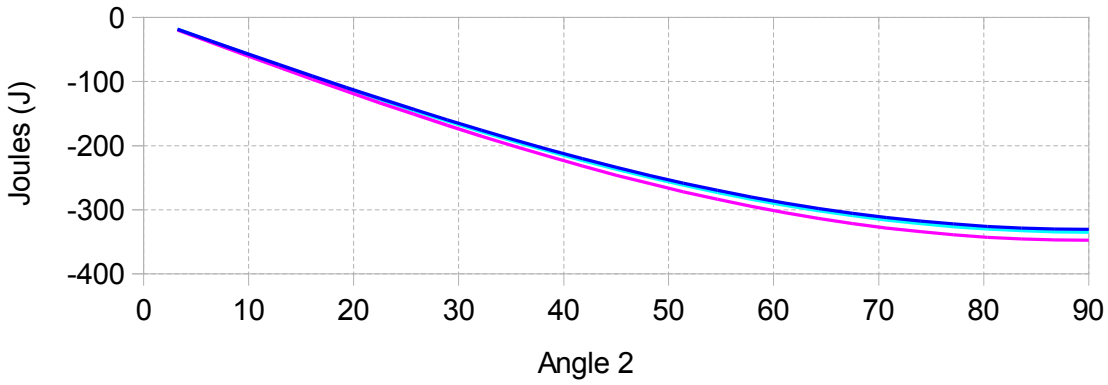


Varying angle 2, angle(3) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-18.545197	-19.507487	-18.781191
6.43	-37.031990	-38.953540	-37.503240
9.64	-55.402200	-58.276960	-56.107210
12.86	-73.597960	-77.416880	-74.534520
16.07	-91.562010	-96.313060	-92.727170
19.29	-109.237770	-114.905990	-110.627860
22.5	-126.569620	-133.137160	-128.180250
25.71	-143.502970	-150.949160	-145.329080
28.93	-159.984520	-168.285920	-162.020370
32.14	-175.962380	-185.092850	-178.201550
35.36	-191.386270	-201.317060	-193.821710
38.57	-206.207630	-216.907470	-208.831670
41.79	-220.379800	-231.815010	-223.184180
45	-233.858220	-245.992800	-236.834110
48.21	-246.600370	-259.396110	-249.738410
51.43	-258.566120	-271.982740	-261.856420
54.64	-269.718030	-283.713300	-273.150240
57.86	-280.020790	-294.550650	-283.584100
61.07	-289.442110	-304.460810	-293.125300
64.29	-297.952300	-313.412550	-301.743800
67.5	-305.524600	-321.377700	-309.412400
70.71	-312.135100	-328.331300	-316.107000
73.93	-317.763200	-334.251400	-321.806700
77.14	-322.391000	-339.119300	-326.493500
80.36	-326.004200	-342.920000	-330.152600
83.57	-328.591300	-345.641300	-332.772600
86.79	-330.144200	-347.274700	-334.345300
90	-330.658100	-347.815300	-334.865700

Max	-18.545197
Min	-347.815300

b. Difference from serial model

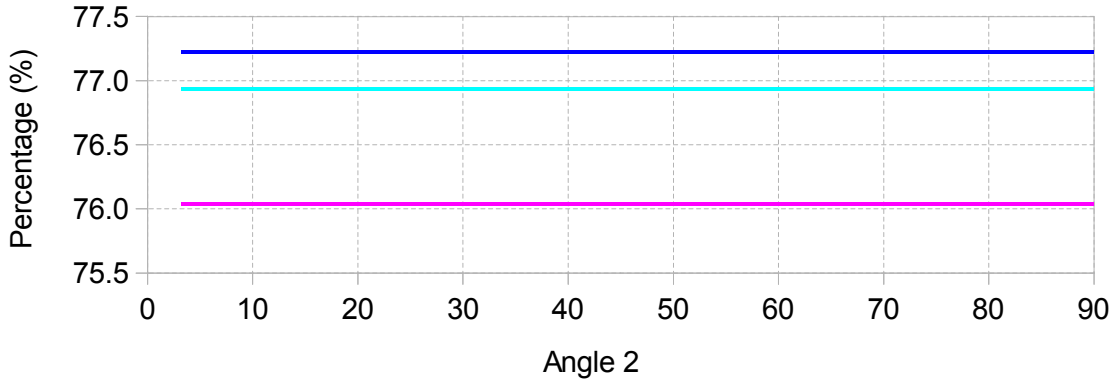


Varying angle 2, angle(3) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	77.22	76.04	76.93
6.43	77.22	76.04	76.93
9.64	77.22	76.04	76.93
12.86	77.22	76.04	76.93
16.07	77.22	76.04	76.93
19.29	77.22	76.04	76.93
22.5	77.22	76.04	76.93
25.71	77.22	76.04	76.93
28.93	77.22	76.04	76.93
32.14	77.22	76.04	76.93
35.36	77.22	76.04	76.93
38.57	77.22	76.04	76.93
41.79	77.22	76.04	76.93
45	77.22	76.04	76.93
48.21	77.22	76.04	76.93
51.43	77.22	76.04	76.93
54.64	77.22	76.04	76.93
57.86	77.22	76.04	76.93
61.07	77.22	76.04	76.93
64.29	77.22	76.04	76.93
67.5	77.22	76.04	76.93
70.71	77.22	76.04	76.93
73.93	77.22	76.04	76.93
77.14	77.22	76.04	76.93
80.36	77.22	76.04	76.93
83.57	77.22	76.04	76.93
86.79	77.22	76.04	76.93
90	77.22	76.04	76.93

Max	77.22
Min	76.04

c. As a percentage of serial model

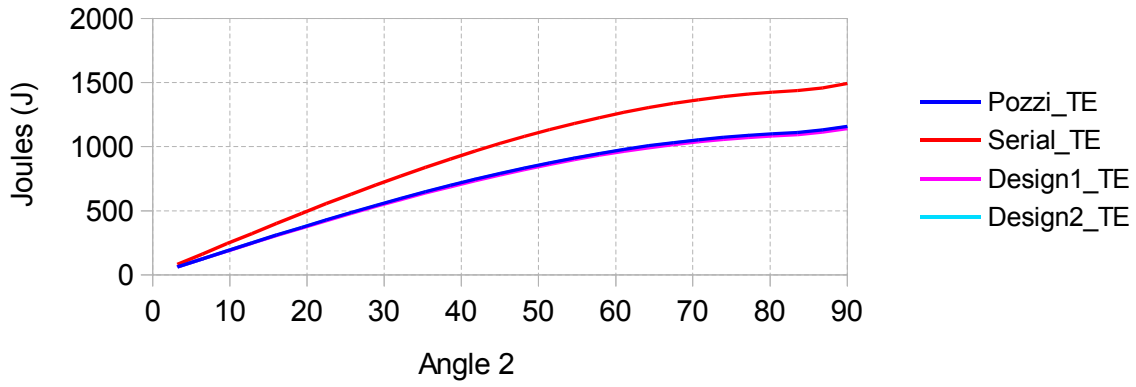


Varying angle 2, angle(3) = 0, heavy motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	62.867391	81.410782	61.905354	62.631499
6.43	125.518340	162.543120	123.597810	125.047510
9.64	187.755530	243.141480	184.883060	187.051440
12.86	249.382850	322.951930	245.568000	248.447930
16.07	310.206080	401.722970	305.461390	309.043480
19.29	370.033470	479.206280	364.374410	368.647080
22.5	428.676350	555.157540	422.121270	427.070740
25.71	485.949730	629.337190	478.519810	484.130170
28.93	541.672850	701.511200	533.392060	539.645310
32.14	595.669820	771.451750	586.564790	593.440910
35.36	647.770100	838.938010	637.870090	645.347060
38.57	697.809060	903.756830	687.145850	695.199780
41.79	745.628540	965.703360	734.236320	742.841490
45	791.077300	1024.581800	778.992570	788.121500
48.21	834.011490	1080.205800	821.272970	830.896520
51.43	874.295160	1132.399300	860.943640	871.031100
54.64	911.800620	1180.997100	897.878860	908.398040
57.86	946.408910	1225.845000	931.961460	942.878820
61.07	978.010110	1266.800700	963.083220	974.363930
64.29	1006.503700	1303.734100	991.145190	1002.753300
67.5	1031.799000	1336.527700	1016.058000	1027.956400
70.71	1053.815200	1365.076900	1037.742100	1049.892900
73.93	1072.481800	1389.290300	1056.128200	1068.492500
77.14	1087.738900	1409.090500	1071.157100	1083.695500
80.36	1099.537200	1424.413500	1082.780400	1095.452800
83.57	1110.723800	1436.469900	1093.494400	1106.416800
86.79	1130.235000	1457.584500	1112.192200	1125.546200
90	1159.038400	1491.558000	1140.142400	1153.994200

Max 1491.558000
Min 61.905354

a. Total energy consumed per model

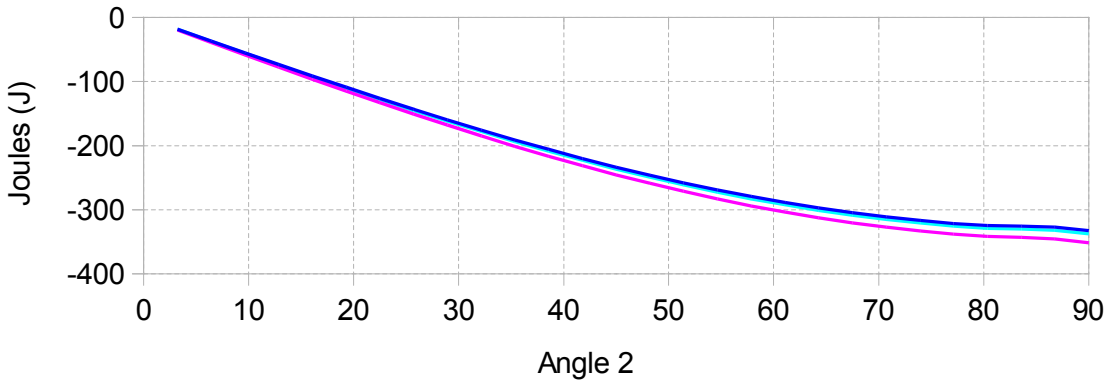


Varying angle 2, angle(3) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-18.543391	-19.505428	-18.779283
6.43	-37.024780	-38.945310	-37.495610
9.64	-55.385950	-58.258420	-56.090040
12.86	-73.569080	-77.383930	-74.504000
16.07	-91.516890	-96.261580	-92.679490
19.29	-109.172810	-114.831870	-110.559200
22.5	-126.481190	-133.036270	-128.086800
25.71	-143.387460	-150.817380	-145.207020
28.93	-159.838350	-168.119140	-161.865890
32.14	-175.781930	-184.886960	-178.010840
35.36	-191.167910	-201.067920	-193.590950
38.57	-205.947770	-216.610980	-208.557050
41.79	-220.074820	-231.467040	-222.861870
45	-233.504500	-245.589230	-236.460300
48.21	-246.194310	-258.932830	-249.309280
51.43	-258.104140	-271.455660	-261.368200
54.64	-269.196480	-283.118240	-272.599060
57.86	-279.436090	-293.883540	-282.966180
61.07	-288.790590	-303.717480	-292.436770
64.29	-297.230400	-312.588910	-300.980800
67.5	-304.728700	-320.469700	-308.571300
70.71	-311.261700	-327.334800	-315.184000
73.93	-316.808500	-333.162100	-320.797800
77.14	-321.351600	-337.933400	-325.395000
80.36	-324.876300	-341.633100	-328.960700
83.57	-325.746100	-342.975500	-330.053100
86.79	-327.349500	-345.392300	-332.038300
90	-332.519600	-351.415600	-337.563800

Max	-18.543391
Min	-351.415600

b. Difference from serial model

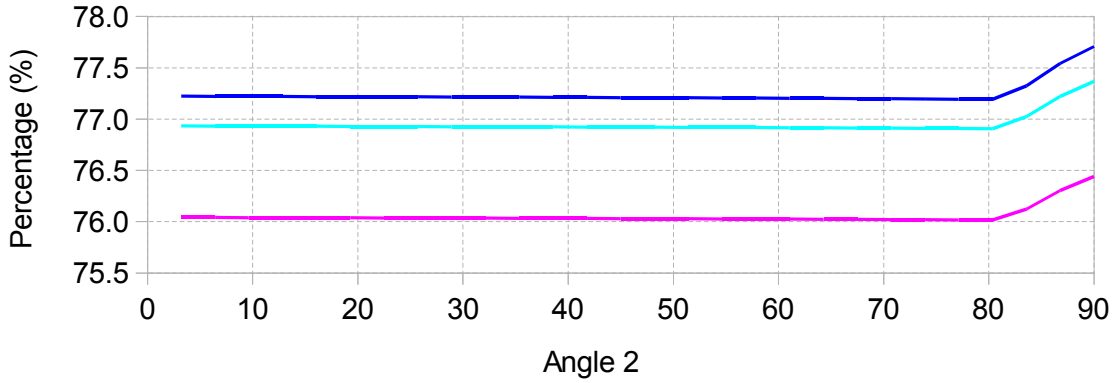


Varying angle 2, angle(3) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	77.22	76.04	76.93
6.43	77.22	76.04	76.93
9.64	77.22	76.04	76.93
12.86	77.22	76.04	76.93
16.07	77.22	76.04	76.93
19.29	77.22	76.04	76.93
22.5	77.22	76.04	76.93
25.71	77.22	76.04	76.93
28.93	77.22	76.03	76.93
32.14	77.21	76.03	76.93
35.36	77.21	76.03	76.92
38.57	77.21	76.03	76.92
41.79	77.21	76.03	76.92
45	77.21	76.03	76.92
48.21	77.21	76.03	76.92
51.43	77.21	76.03	76.92
54.64	77.21	76.03	76.92
57.86	77.2	76.03	76.92
61.07	77.2	76.02	76.92
64.29	77.2	76.02	76.91
67.5	77.2	76.02	76.91
70.71	77.2	76.02	76.91
73.93	77.2	76.02	76.91
77.14	77.19	76.02	76.91
80.36	77.19	76.02	76.91
83.57	77.32	76.12	77.02
86.79	77.54	76.3	77.22
90	77.71	76.44	77.37

Max	77.71
Min	76.02

c. As a percentage of serial model

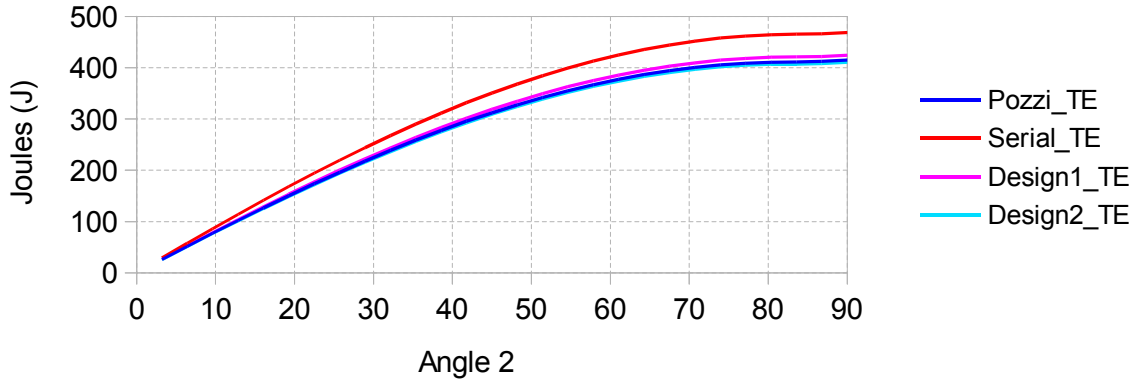


Varying angle 2, angle(3) = 170, light motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	25.996945	29.032461	26.478720	25.760951
6.43	51.731419	57.793888	52.697783	51.260174
9.64	77.122402	86.193732	78.574642	76.417390
12.86	102.089950	114.142580	104.027830	101.153400
16.07	126.555470	141.552440	128.977210	125.390320
19.29	150.441930	168.337020	153.344240	149.051850
22.5	173.674130	194.412010	177.052200	172.063500
25.71	196.178940	219.695300	200.026470	194.352820
28.93	217.885510	244.107310	222.194710	215.849660
32.14	238.725510	267.571200	243.487150	236.486340
35.36	258.633340	290.013090	263.836750	256.197900
38.57	277.546340	311.362360	283.179480	274.922300
41.79	295.404980	331.551810	301.454440	292.600600
45	312.153050	350.517880	318.604110	309.177160
48.21	327.737850	368.200910	334.574540	324.599810
51.43	342.110330	384.545220	349.315450	338.820030
54.64	355.225270	399.499410	362.780480	351.793070
57.86	367.041410	413.016420	374.927240	363.478110
61.07	377.521580	425.053730	385.717550	373.838400
64.29	386.632830	435.573480	395.117450	382.841350
67.5	394.346490	444.542590	403.097390	390.458660
70.71	400.638330	451.932870	409.632300	396.666390
73.93	405.488580	457.721100	414.701630	401.445030
77.14	408.882020	461.889100	418.289480	404.779590
80.36	410.808010	464.423810	420.384600	406.659600
83.57	411.260530	465.317300	420.980440	407.079220
86.79	412.252840	466.062260	421.864530	408.089210
90	414.767900	468.450640	424.279280	410.612010

Max	468.450640
Min	25.760951

a. Total energy consumed per model

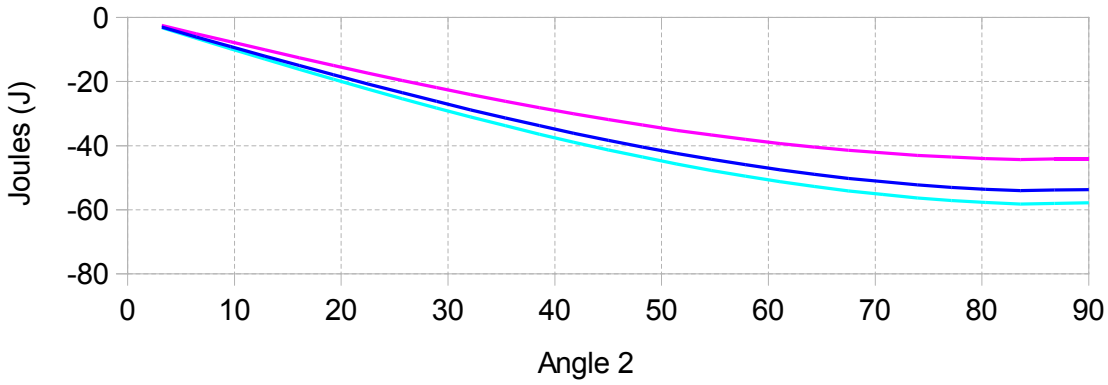


Varying angle 2, angle(3) = 170, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-3.035516	-2.553741	-3.271510
6.43	-6.062469	-5.096105	-6.533714
9.64	-9.071330	-7.619090	-9.776342
12.86	-12.052630	-10.114750	-12.989180
16.07	-14.996970	-12.575230	-16.162120
19.29	-17.895090	-14.992780	-19.285170
22.5	-20.737880	-17.359810	-22.348510
25.71	-23.516360	-19.668830	-25.342480
28.93	-26.221800	-21.912600	-28.257650
32.14	-28.845690	-24.084050	-31.084860
35.36	-31.379750	-26.176340	-33.815190
38.57	-33.816020	-28.182880	-36.440060
41.79	-36.146830	-30.097370	-38.951210
45	-38.364830	-31.913770	-41.340720
48.21	-40.463060	-33.626370	-43.601100
51.43	-42.434890	-35.229770	-45.725190
54.64	-44.274140	-36.718930	-47.706340
57.86	-45.975010	-38.089180	-49.538310
61.07	-47.532150	-39.336180	-51.215330
64.29	-48.940650	-40.456030	-52.732130
67.5	-50.196100	-41.445200	-54.083930
70.71	-51.294540	-42.300570	-55.266480
73.93	-52.232520	-43.019470	-56.276070
77.14	-53.007080	-43.599620	-57.109510
80.36	-53.615800	-44.039210	-57.764210
83.57	-54.056770	-44.336860	-58.238080
86.79	-53.809420	-44.197730	-57.973050
90	-53.682740	-44.171360	-57.838630

Max	-2.553741
Min	-58.238080

b. Difference from serial model

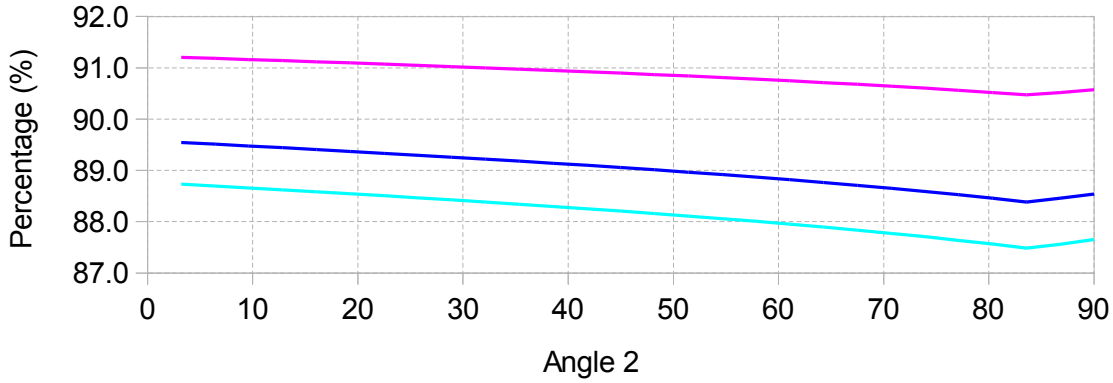


Varying angle 2, angle(3) = 170, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	89.54	91.2	88.73
6.43	89.51	91.18	88.69
9.64	89.48	91.16	88.66
12.86	89.44	91.14	88.62
16.07	89.41	91.12	88.58
19.29	89.37	91.09	88.54
22.5	89.33	91.07	88.5
25.71	89.3	91.05	88.46
28.93	89.26	91.02	88.42
32.14	89.22	91	88.38
35.36	89.18	90.97	88.34
38.57	89.14	90.95	88.3
41.79	89.1	90.92	88.25
45	89.05	90.9	88.21
48.21	89.01	90.87	88.16
51.43	88.96	90.84	88.11
54.64	88.92	90.81	88.06
57.86	88.87	90.78	88.01
61.07	88.82	90.75	87.95
64.29	88.76	90.71	87.89
67.5	88.71	90.68	87.83
70.71	88.65	90.64	87.77
73.93	88.59	90.6	87.71
77.14	88.52	90.56	87.64
80.36	88.46	90.52	87.56
83.57	88.38	90.47	87.48
86.79	88.45	90.52	87.56
90	88.54	90.57	87.65

Max	91.2
Min	87.48

c. As a percentage of serial model

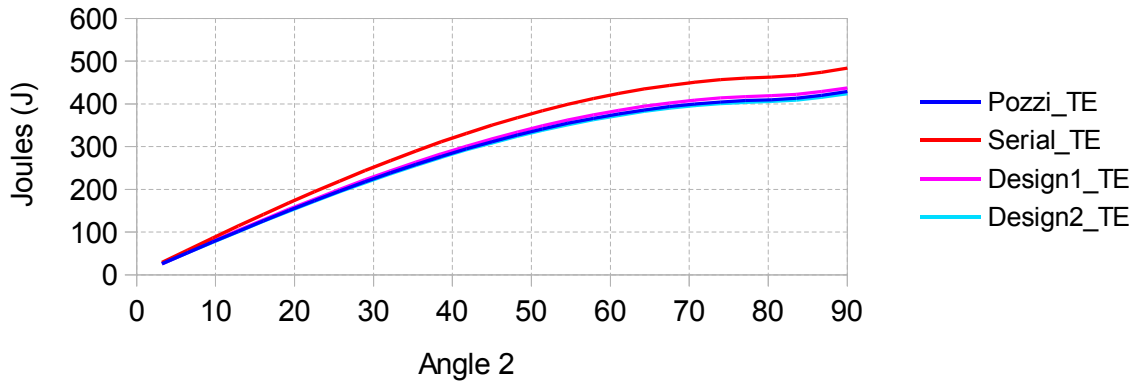


Varying angle 2, angle(3) = 170, light motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	25.995086	29.030286	26.476892	25.759194
6.43	51.723982	57.785191	52.690472	51.253147
9.64	77.105668	86.174164	78.558193	76.401579
12.86	102.060210	114.107790	103.998590	101.125290
16.07	126.508990	141.498090	128.931520	125.346400
19.29	150.375000	168.258750	153.278440	148.988600
22.5	173.583030	194.305470	176.962640	171.977410
25.71	196.059950	219.556140	199.909490	194.240390
28.93	217.734910	243.931190	222.046670	215.707360
32.14	238.539580	267.353770	243.304380	236.310660
35.36	258.408370	289.750000	263.615600	255.985330
38.57	277.278600	311.049260	282.916290	274.669320
41.79	295.090760	331.184350	301.145560	292.303700
45	311.788630	350.091720	318.245890	308.832830
48.21	327.319510	367.711690	334.163310	324.204530
51.43	341.634350	383.988610	348.847560	338.370290
54.64	354.687930	398.871040	362.252270	351.285350
57.86	366.439000	412.311950	374.335070	362.908910
61.07	376.850380	424.268810	385.057750	373.204200
64.29	385.889100	434.703770	394.386370	382.138640
67.5	393.526540	443.583730	402.291380	389.683920
70.71	399.738430	450.880520	408.747690	395.816100
73.93	404.505010	456.570900	413.734780	400.515690
77.14	407.811060	460.636710	417.236730	403.767670
80.36	409.645940	463.064880	419.242290	405.561610
83.57	413.510830	466.726950	422.549570	409.125320
86.79	420.270190	474.169030	429.077990	415.581320
90	428.961990	483.757000	437.525270	423.917760

Max	483.757000
Min	25.759194

a. Total energy consumed per model

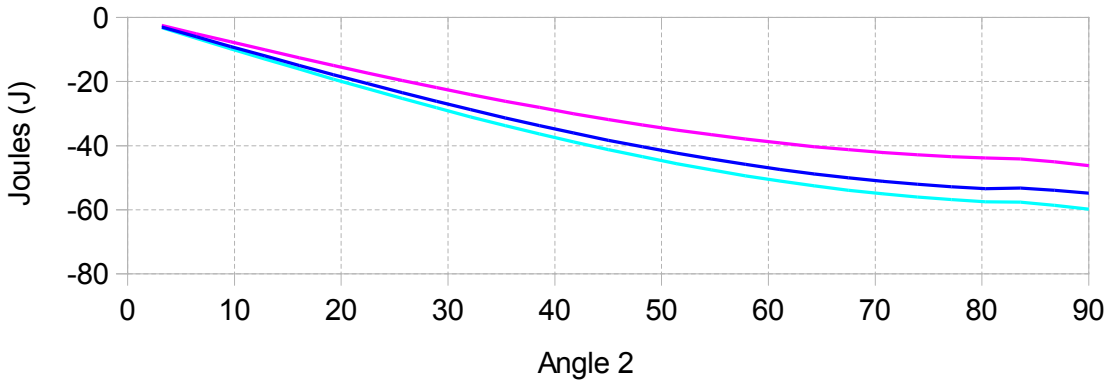


Varying angle 2, angle(3) = 170, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-3.035200	-2.553394	-3.271092
6.43	-6.061209	-5.094719	-6.532044
9.64	-9.068496	-7.615971	-9.772585
12.86	-12.047580	-10.109200	-12.982500
16.07	-14.989100	-12.566570	-16.151690
19.29	-17.883750	-14.980310	-19.270150
22.5	-20.722440	-17.342830	-22.328060
25.71	-23.496190	-19.646650	-25.315750
28.93	-26.196280	-21.884520	-28.223830
32.14	-28.814190	-24.049390	-31.043110
35.36	-31.341630	-26.134400	-33.764670
38.57	-33.770660	-28.132970	-36.379940
41.79	-36.093590	-30.038790	-38.880650
45	-38.303090	-31.845830	-41.258890
48.21	-40.392180	-33.548380	-43.507160
51.43	-42.354260	-35.141050	-45.618320
54.64	-44.183110	-36.618770	-47.585690
57.86	-45.872950	-37.976880	-49.403040
61.07	-47.418430	-39.211060	-51.064610
64.29	-48.814670	-40.317400	-52.565130
67.5	-50.057190	-41.292350	-53.899810
70.71	-51.142090	-42.132830	-55.064420
73.93	-52.065890	-42.836120	-56.055210
77.14	-52.825650	-43.399980	-56.869040
80.36	-53.418940	-43.822590	-57.503270
83.57	-53.216120	-44.177380	-57.601630
86.79	-53.898840	-45.091040	-58.587710
90	-54.795010	-46.231730	-59.839240

Max	-2.553394
Min	-59.839240

b. Difference from serial model

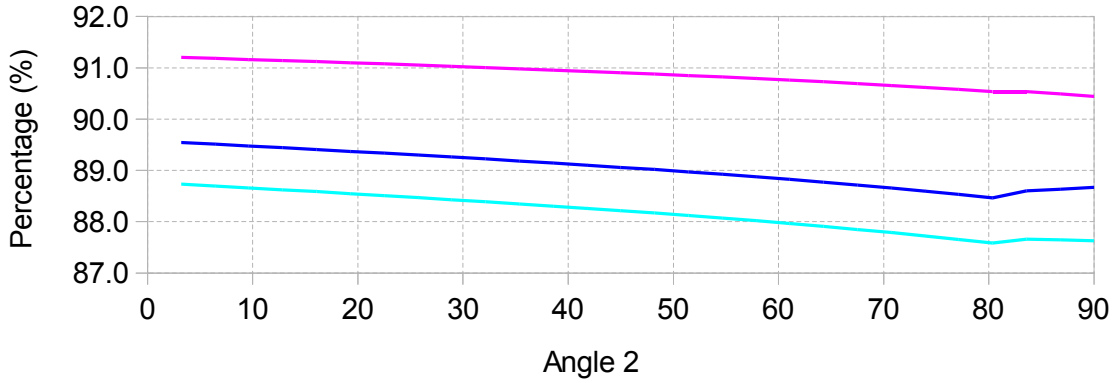


Varying angle 2, angle(3) = 170, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	89.54	91.2	88.73
6.43	89.51	91.18	88.7
9.64	89.48	91.16	88.66
12.86	89.44	91.14	88.62
16.07	89.41	91.12	88.59
19.29	89.37	91.1	88.55
22.5	89.34	91.07	88.51
25.71	89.3	91.05	88.47
28.93	89.26	91.03	88.43
32.14	89.22	91	88.39
35.36	89.18	90.98	88.35
38.57	89.14	90.96	88.3
41.79	89.1	90.93	88.26
45	89.06	90.9	88.21
48.21	89.02	90.88	88.17
51.43	88.97	90.85	88.12
54.64	88.92	90.82	88.07
57.86	88.87	90.79	88.02
61.07	88.82	90.76	87.96
64.29	88.77	90.73	87.91
67.5	88.72	90.69	87.85
70.71	88.66	90.66	87.79
73.93	88.6	90.62	87.72
77.14	88.53	90.58	87.65
80.36	88.46	90.54	87.58
83.57	88.6	90.53	87.66
86.79	88.63	90.49	87.64
90	88.67	90.44	87.63

Max	91.2
Min	87.58

c. As a percentage of serial model

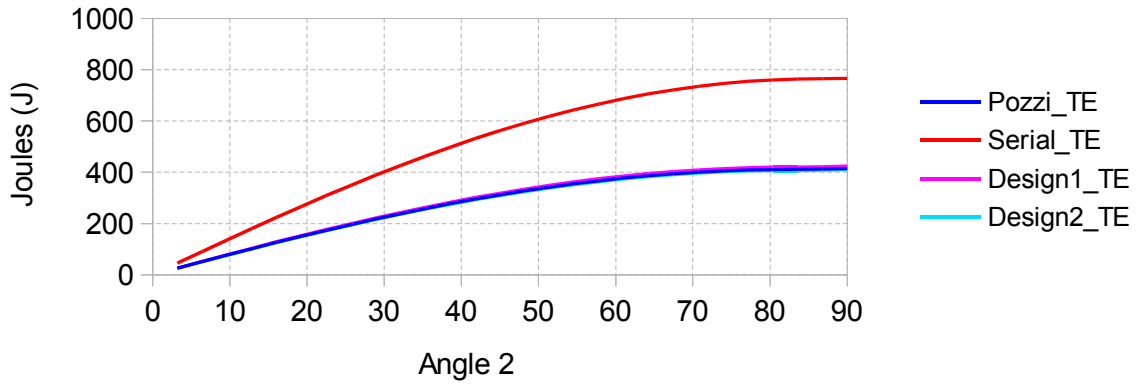


Varying angle 2, angle(3) = 170, heavy motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	25.996945	45.780870	26.478720	25.760951
6.43	51.731419	91.243038	52.697783	51.260174
9.64	77.122402	136.243370	78.574642	76.417390
12.86	102.089950	180.640200	104.027830	101.153400
16.07	126.555470	224.293740	128.977210	125.390320
19.29	150.441930	267.066570	153.344240	149.051850
22.5	173.674130	308.824030	177.052200	172.063500
25.71	196.178940	349.434650	200.026470	194.352820
28.93	217.885510	388.770590	222.194710	215.849660
32.14	238.725510	426.708030	243.487150	236.486340
35.36	258.633340	463.127550	263.836750	256.197900
38.57	277.546340	497.914490	283.179480	274.922300
41.79	295.404980	530.959370	301.454440	292.600600
45	312.153050	562.158190	318.604110	309.177160
48.21	327.737850	591.412740	334.574540	324.599810
51.43	342.110330	618.630960	349.315450	338.820030
54.64	355.225270	643.727210	362.780480	351.793070
57.86	367.041410	666.622530	374.927240	363.478110
61.07	377.521580	687.244870	385.717550	373.838400
64.29	386.632830	705.529380	395.117450	382.841350
67.5	394.346490	721.418540	403.097390	390.458660
70.71	400.638330	734.862410	409.632300	396.666390
73.93	405.488580	745.818720	414.701630	401.445030
77.14	408.882020	754.253070	418.289480	404.779590
80.36	410.808010	760.138970	420.384600	406.659600
83.57	411.260530	763.457980	420.980440	407.079220
86.79	412.252840	764.199730	421.864530	408.089210
90	414.767900	766.023630	424.279280	410.612010

Max	766.023630
Min	25.760951

a. Total energy consumed per model

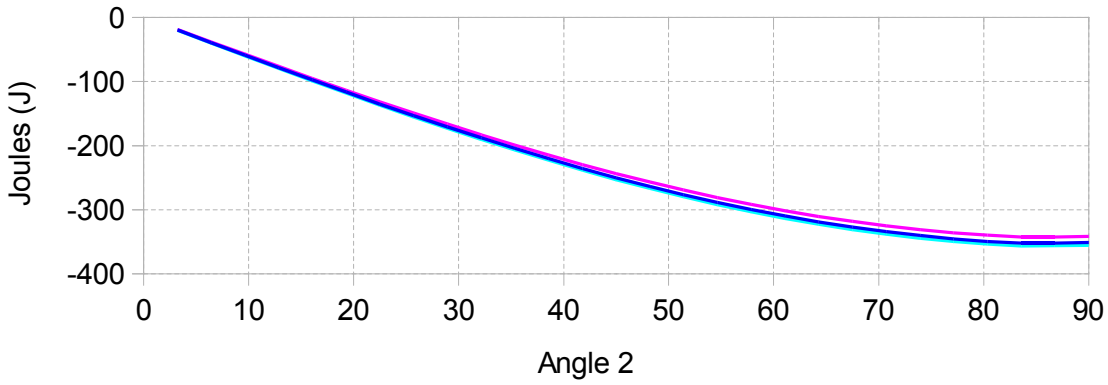


Varying angle 2, angle(3) = 170, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-19.783925	-19.302150	-20.019919
6.43	-39.511619	-38.545255	-39.982864
9.64	-59.120968	-57.668728	-59.825980
12.86	-78.550250	-76.612370	-79.486800
16.07	-97.738270	-95.316530	-98.903420
19.29	-116.624640	-113.722330	-118.014720
22.5	-135.149900	-131.771830	-136.760530
25.71	-153.255710	-149.408180	-155.081830
28.93	-170.885080	-166.575880	-172.920930
32.14	-187.982520	-183.220880	-190.221690
35.36	-204.494210	-199.290800	-206.929650
38.57	-220.368150	-214.735010	-222.992190
41.79	-235.554390	-229.504930	-238.358770
45	-250.005140	-243.554080	-252.981030
48.21	-263.674890	-256.838200	-266.812930
51.43	-276.520630	-269.315510	-279.810930
54.64	-288.501940	-280.946730	-291.934140
57.86	-299.581120	-291.695290	-303.144420
61.07	-309.723290	-301.527320	-313.406470
64.29	-318.896550	-310.411930	-322.688030
67.5	-327.072050	-318.321150	-330.959880
70.71	-334.224080	-325.230110	-338.196020
73.93	-340.330140	-331.117090	-344.373690
77.14	-345.371050	-335.963590	-349.473480
80.36	-349.330960	-339.754370	-353.479370
83.57	-352.197450	-342.477540	-356.378760
86.79	-351.946890	-342.335200	-356.110520
90	-351.255730	-341.744350	-355.411620

Max	-19.302150
Min	-356.378760

b. Difference from serial model

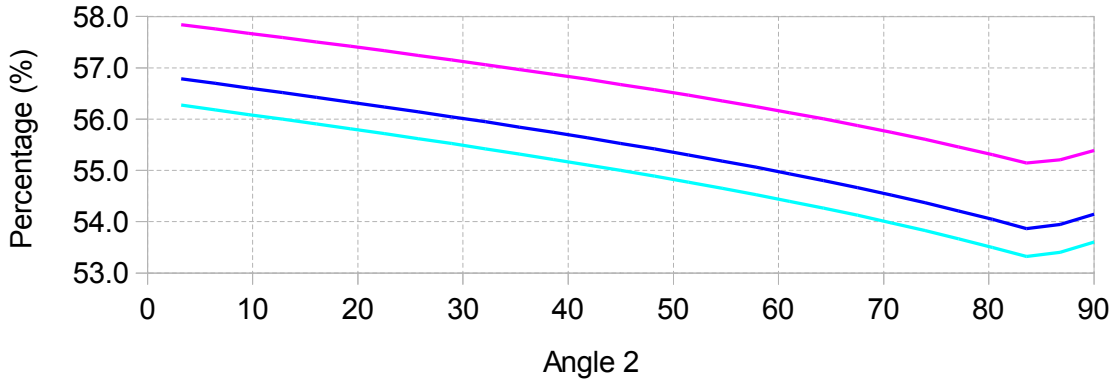


Varying angle 2, angle(3) = 170, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	56.79	57.84	56.27
6.43	56.7	57.76	56.18
9.64	56.61	57.67	56.09
12.86	56.52	57.59	56
16.07	56.42	57.5	55.9
19.29	56.33	57.42	55.81
22.5	56.24	57.33	55.72
25.71	56.14	57.24	55.62
28.93	56.04	57.15	55.52
32.14	55.95	57.06	55.42
35.36	55.84	56.97	55.32
38.57	55.74	56.87	55.21
41.79	55.64	56.78	55.11
45	55.53	56.68	55
48.21	55.42	56.57	54.89
51.43	55.3	56.47	54.77
54.64	55.18	56.36	54.65
57.86	55.06	56.24	54.53
61.07	54.93	56.13	54.4
64.29	54.8	56	54.26
67.5	54.66	55.88	54.12
70.71	54.52	55.74	53.98
73.93	54.37	55.6	53.83
77.14	54.21	55.46	53.67
80.36	54.04	55.3	53.5
83.57	53.87	55.14	53.32
86.79	53.95	55.2	53.4
90	54.15	55.39	53.6

Max	57.84
Min	53.32

c. As a percentage of serial model

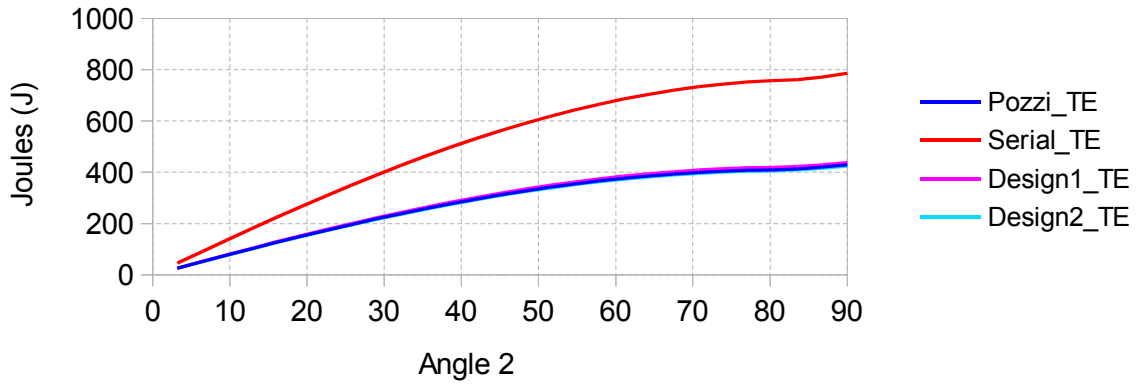


Varying angle 2, angle(3) = 170, heavy motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
3.21	25.995086	45.776959	26.476892	25.759194
6.43	51.723982	91.227394	52.690472	51.253147
9.64	77.105668	136.208170	78.558193	76.401579
12.86	102.060210	180.577620	103.998590	101.125290
16.07	126.508990	224.195970	128.931520	125.346400
19.29	150.375000	266.925780	153.278440	148.988600
22.5	173.583030	308.632390	176.962640	171.977410
25.71	196.059950	349.184340	199.909490	194.240390
28.93	217.734910	388.453800	222.046670	215.707360
32.14	238.539580	426.316930	243.304380	236.310660
35.36	258.408370	462.654310	263.615600	255.985330
38.57	277.278600	497.351310	282.916290	274.669320
41.79	295.090760	530.298410	301.145560	292.303700
45	311.788630	561.391630	318.245890	308.832830
48.21	327.319510	590.532760	334.163310	324.204530
51.43	341.634350	617.629740	348.847560	338.370290
54.64	354.687930	642.596930	362.252270	351.285350
57.86	366.439000	665.355360	374.335070	362.908910
61.07	376.850380	685.833000	385.057750	373.204200
64.29	385.889100	703.964970	394.386370	382.138640
67.5	393.526540	719.693790	402.291380	389.683920
70.71	399.738430	732.969480	408.747690	395.816100
73.93	404.505010	743.749800	413.734780	400.515690
77.14	407.811060	752.000330	417.236730	403.767670
80.36	409.645940	757.694590	419.242290	405.561610
83.57	413.510830	760.970420	422.549570	409.125320
86.79	420.270190	771.466230	429.077990	415.581320
90	428.961990	786.003650	437.525270	423.917760

Max	786.003650
Min	25.759194

a. Total energy consumed per model

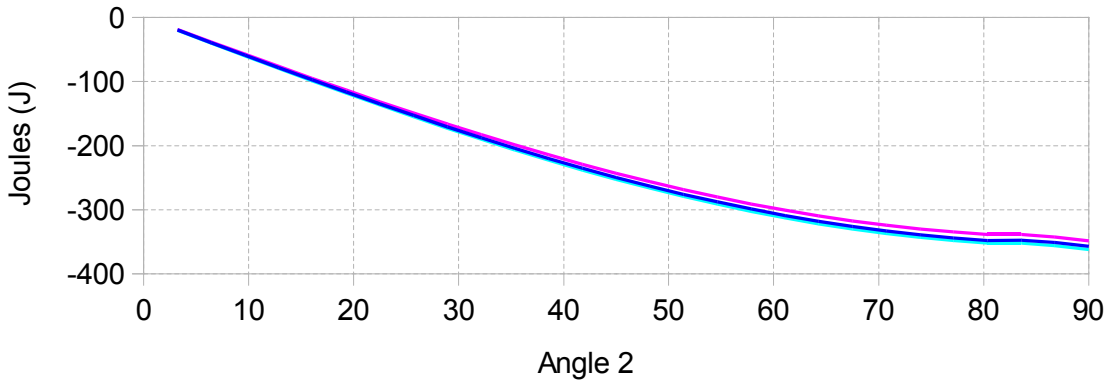


Varying angle 2, angle(3) = 170, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	-19.781873	-19.300067	-20.017765
6.43	-39.503412	-38.536922	-39.974247
9.64	-59.102502	-57.649977	-59.806591
12.86	-78.517410	-76.579030	-79.452330
16.07	-97.686980	-95.264450	-98.849570
19.29	-116.550780	-113.647340	-117.937180
22.5	-135.049360	-131.669750	-136.654980
25.71	-153.124390	-149.274850	-154.943950
28.93	-170.718890	-166.407130	-172.746440
32.14	-187.777350	-183.012550	-190.006270
35.36	-204.245940	-199.038710	-206.668980
38.57	-220.072710	-214.435020	-222.681990
41.79	-235.207650	-229.152850	-237.994710
45	-249.603000	-243.145740	-252.558800
48.21	-263.213250	-256.369450	-266.328230
51.43	-275.995390	-268.782180	-279.259450
54.64	-287.909000	-280.344660	-291.311580
57.86	-298.916360	-291.020290	-302.446450
61.07	-308.982620	-300.775250	-312.628800
64.29	-318.075870	-309.578600	-321.826330
67.5	-326.167250	-317.402410	-330.009870
70.71	-333.231050	-324.221790	-337.153380
73.93	-339.244790	-330.015020	-343.234110
77.14	-344.189270	-334.763600	-348.232660
80.36	-348.048650	-338.452300	-352.132980
83.57	-347.459590	-338.420850	-351.845100
86.79	-351.196040	-342.388240	-355.884910
90	-357.041660	-348.478380	-362.085890

Max	-19.300067
Min	-362.085890

b. Difference from serial model

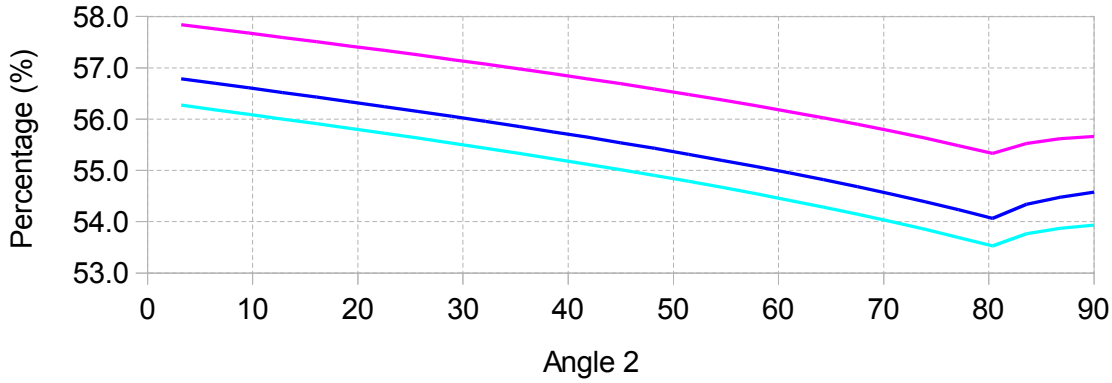


Varying angle 2, angle(3) = 170, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
3.21	56.79	57.84	56.27
6.43	56.7	57.76	56.18
9.64	56.61	57.68	56.09
12.86	56.52	57.59	56
16.07	56.43	57.51	55.91
19.29	56.34	57.42	55.82
22.5	56.24	57.34	55.72
25.71	56.15	57.25	55.63
28.93	56.05	57.16	55.53
32.14	55.95	57.07	55.43
35.36	55.85	56.98	55.33
38.57	55.75	56.88	55.23
41.79	55.65	56.79	55.12
45	55.54	56.69	55.01
48.21	55.43	56.59	54.9
51.43	55.31	56.48	54.79
54.64	55.2	56.37	54.67
57.86	55.07	56.26	54.54
61.07	54.95	56.14	54.42
64.29	54.82	56.02	54.28
67.5	54.68	55.9	54.15
70.71	54.54	55.77	54
73.93	54.39	55.63	53.85
77.14	54.23	55.48	53.69
80.36	54.06	55.33	53.53
83.57	54.34	55.53	53.76
86.79	54.48	55.62	53.87
90	54.58	55.66	53.93

Max	57.84
Min	53.53

c. As a percentage of serial model

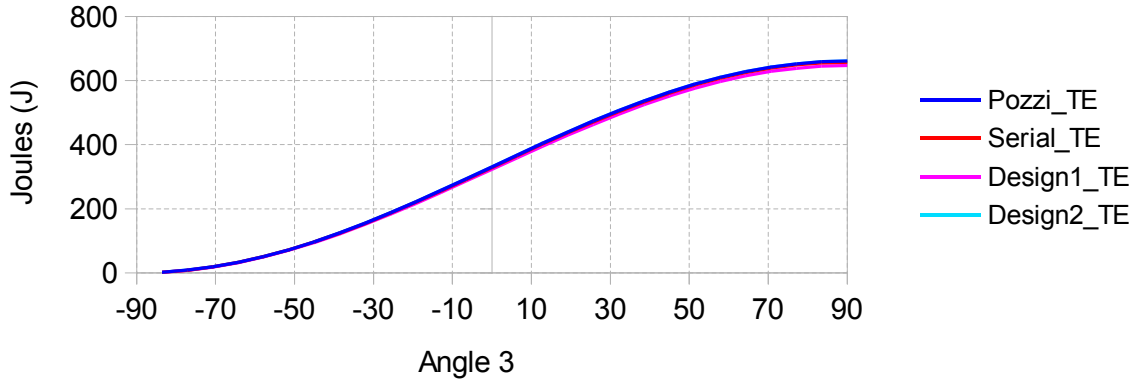


Varying angle 3, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
-83.57	2.078568	2.067102	2.034150	2.078568
-77.14	8.288099	8.242378	8.110986	8.288100
-70.71	18.550404	18.448072	18.153990	18.550407
-64.29	32.736266	32.555678	32.036706	32.736271
-57.86	50.667062	50.387560	49.584329	50.667070
-51.43	72.117021	71.719191	70.575912	72.117032
-45	96.816061	96.281981	94.747145	96.816077
-38.57	124.453200	123.766660	121.793690	124.453220
-32.14	154.680460	153.827180	151.375010	154.680490
-25.71	187.117280	186.085050	183.118660	187.117310
-19.29	221.355260	220.134160	216.625000	221.355300
-12.86	256.963360	255.545840	251.472170	256.963410
-6.43	293.493310	291.874270	287.221490	293.493360
0	330.485220	328.662120	323.422910	330.485290
6.43	367.473450	365.446300	359.620720	367.473520
12.86	403.992390	401.763790	395.359270	403.992480
19.29	439.582390	437.157460	430.188740	439.582480
25.71	473.795510	471.181830	463.670740	473.795610
32.14	506.201170	503.408720	495.383910	506.201280
38.57	536.391570	533.432580	524.929170	536.391690
45	563.986840	560.875620	551.934750	563.986980
51.43	588.639800	585.392580	576.060900	588.639950
57.86	610.040330	606.675060	597.004120	610.040500
64.29	627.919290	624.455380	614.501030	627.919470
70.71	642.051880	638.510010	628.331630	642.052080
77.14	652.260480	648.662290	638.322090	652.260690
83.57	658.416880	654.784720	644.346950	658.417110
90	660.443890	656.800540	646.330670	660.444140

Max	660.444140
Min	2.034150

a. Total energy consumed per model

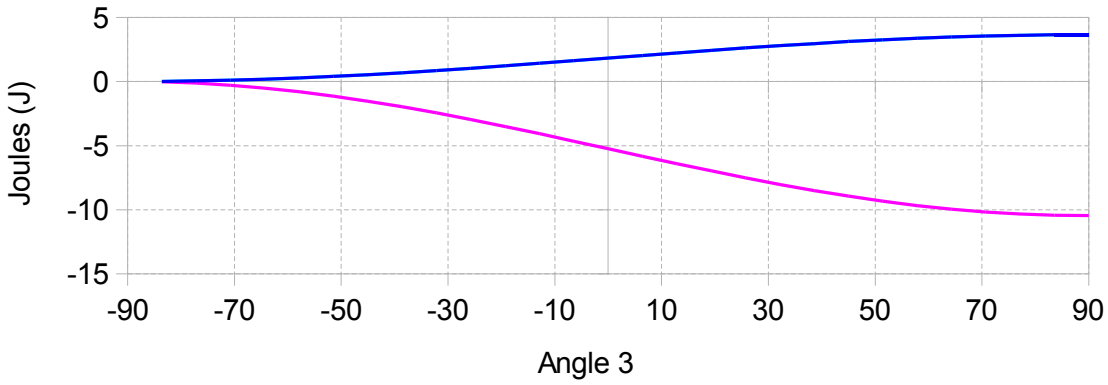


Varying angle 3, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	0.011466	-0.032952	0.011467
-77.14	0.045721	-0.131392	0.045722
-70.71	0.102332	-0.294082	0.102335
-64.29	0.180588	-0.518972	0.180593
-57.86	0.279502	-0.803231	0.279510
-51.43	0.397830	-1.143279	0.397841
-45	0.534080	-1.534836	0.534096
-38.57	0.686540	-1.972970	0.686560
-32.14	0.853280	-2.452170	0.853310
-25.71	1.032230	-2.966390	1.032260
-19.29	1.221100	-3.509160	1.221140
-12.86	1.417520	-4.073670	1.417570
-6.43	1.619040	-4.652780	1.619090
0	1.823100	-5.239210	1.823170
6.43	2.027150	-5.825580	2.027220
12.86	2.228600	-6.404520	2.228690
19.29	2.424930	-6.968720	2.425020
25.71	2.613680	-7.511090	2.613780
32.14	2.792450	-8.024810	2.792560
38.57	2.958990	-8.503410	2.959110
45	3.111220	-8.940870	3.111360
51.43	3.247220	-9.331680	3.247370
57.86	3.365270	-9.670940	3.365440
64.29	3.463910	-9.954350	3.464090
70.71	3.541870	-10.178380	3.542070
77.14	3.598190	-10.340200	3.598400
83.57	3.632160	-10.437770	3.632390
90	3.643350	-10.469870	3.643600

Max	3.643600
Min	-10.469870

b. Difference from serial model

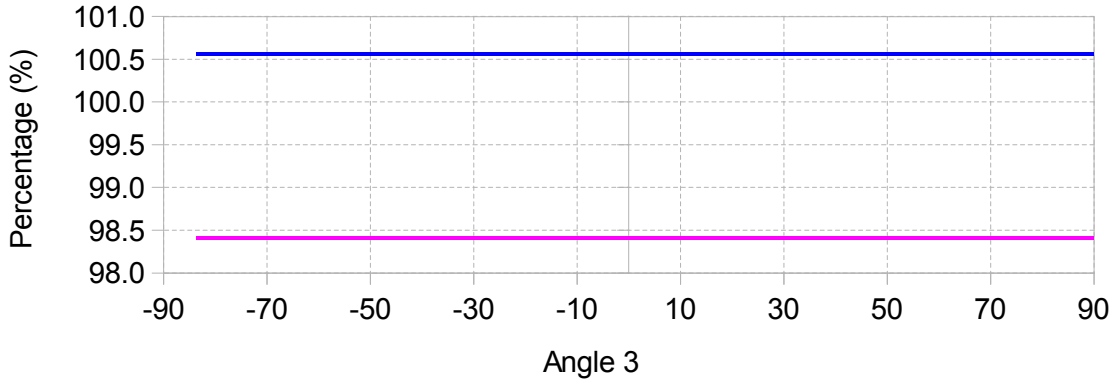


Varying angle 3, angle(2) = 0, light motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	100.55	98.41	100.55
-77.14	100.55	98.41	100.55
-70.71	100.55	98.41	100.55
-64.29	100.55	98.41	100.55
-57.86	100.55	98.41	100.55
-51.43	100.55	98.41	100.55
-45	100.55	98.41	100.55
-38.57	100.55	98.41	100.55
-32.14	100.55	98.41	100.55
-25.71	100.55	98.41	100.55
-19.29	100.55	98.41	100.55
-12.86	100.55	98.41	100.55
-6.43	100.55	98.41	100.55
0	100.55	98.41	100.55
6.43	100.55	98.41	100.55
12.86	100.55	98.41	100.55
19.29	100.55	98.41	100.55
25.71	100.55	98.41	100.55
32.14	100.55	98.41	100.55
38.57	100.55	98.41	100.55
45	100.55	98.41	100.55
51.43	100.55	98.41	100.55
57.86	100.55	98.41	100.55
64.29	100.55	98.41	100.55
70.71	100.55	98.41	100.55
77.14	100.55	98.41	100.55
83.57	100.55	98.41	100.55
90	100.55	98.41	100.55

Max	100.55
Min	98.41

c. As a percentage of serial model

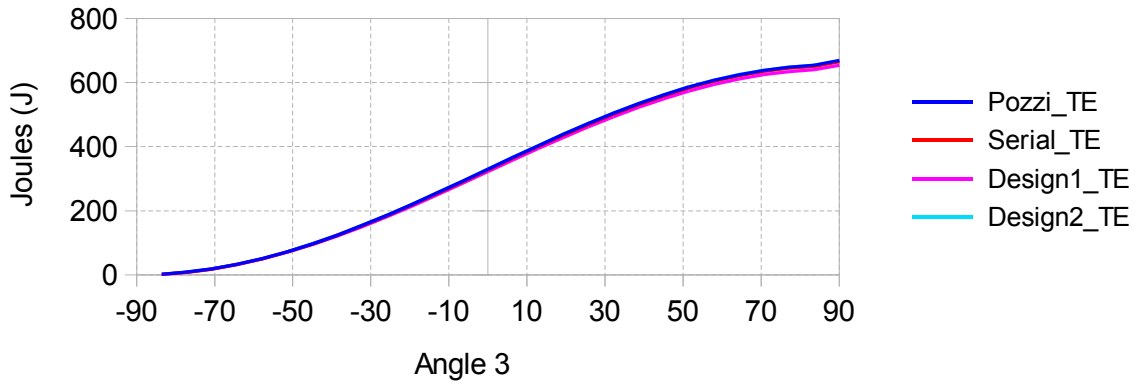


Varying angle 3, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
-83.57	2.072224	2.060752	2.028095	2.072350
-77.14	8.262722	8.216978	8.086767	8.263229
-70.71	18.493307	18.390923	18.099499	18.494447
-64.29	32.634760	32.454080	31.939833	32.636786
-57.86	50.508460	50.228814	49.432965	50.511626
-51.43	71.888634	71.490597	70.357948	71.893192
-45	96.505201	95.970838	94.450471	96.511406
-38.57	124.047180	123.360270	121.406200	124.055280
-32.14	154.166590	153.312840	150.884590	154.176850
-25.71	186.482870	185.450070	182.513210	186.495530
-19.29	220.587620	219.365830	215.892400	220.602950
-12.86	256.049810	254.631460	250.600320	256.068050
-6.43	292.421160	290.801140	286.198270	292.442560
0	329.241780	327.417550	322.236220	329.266600
6.43	366.046030	364.017590	358.258450	366.074520
12.86	402.368310	400.138230	393.809310	402.400720
19.29	437.748950	435.322350	428.438970	437.785550
25.71	471.740020	469.124480	461.709070	471.781050
32.14	503.910950	501.116430	493.198220	503.956660
38.57	533.853930	530.892640	522.507340	533.904580
45	561.189100	558.075340	549.264690	561.244940
51.43	585.569260	582.319260	573.130490	585.630550
57.86	606.684310	603.315990	593.801260	606.751290
64.29	624.265100	620.797870	611.013610	624.338030
70.71	638.086820	634.541350	624.547530	638.165970
77.14	647.971880	644.369790	634.229220	648.057480
83.57	653.792040	650.155680	639.933190	653.884350
90	668.828480	665.299130	653.930740	668.332110

Max	668.828480
Min	2.028095

a. Total energy consumed per model

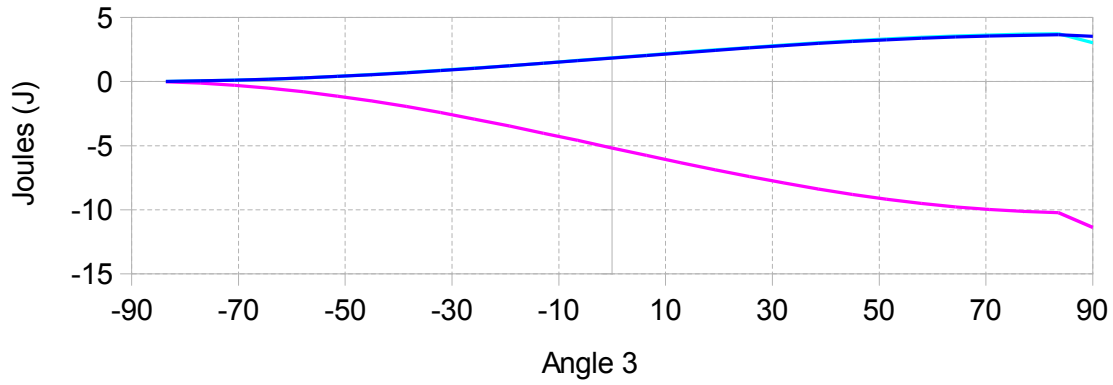


Varying angle 3, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	0.011472	-0.032656	0.011599
-77.14	0.045744	-0.130211	0.046250
-70.71	0.102384	-0.291424	0.103524
-64.29	0.180680	-0.514247	0.182706
-57.86	0.279646	-0.795849	0.282812
-51.43	0.398037	-1.132649	0.402595
-45	0.534363	-1.520367	0.540568
-38.57	0.686910	-1.954070	0.695010
-32.14	0.853750	-2.428250	0.864010
-25.71	1.032800	-2.936860	1.045460
-19.29	1.221790	-3.473430	1.237120
-12.86	1.418350	-4.031140	1.436590
-6.43	1.620020	-4.602870	1.641420
0	1.824230	-5.181330	1.849050
6.43	2.028440	-5.759140	2.056930
12.86	2.230080	-6.328920	2.262490
19.29	2.426600	-6.883380	2.463200
25.71	2.615540	-7.415410	2.656570
32.14	2.794520	-7.918210	2.840230
38.57	2.961290	-8.385300	3.011940
45	3.113760	-8.810650	3.169600
51.43	3.250000	-9.188770	3.311290
57.86	3.368320	-9.514730	3.435300
64.29	3.467230	-9.784260	3.540160
70.71	3.545470	-9.993820	3.624620
77.14	3.602090	-10.140570	3.687690
83.57	3.636360	-10.222490	3.728670
90	3.529350	-11.368390	3.032980

Max	3.728670
Min	-11.368390

b. Difference from serial model

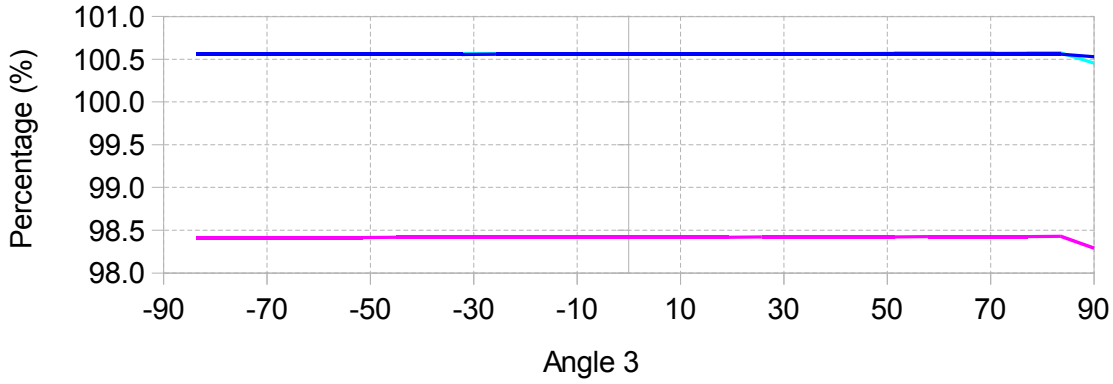


Varying angle 3, angle(2) = 0, light motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	100.56	98.42	100.56
-77.14	100.56	98.42	100.56
-70.71	100.56	98.42	100.56
-64.29	100.56	98.42	100.56
-57.86	100.56	98.42	100.56
-51.43	100.56	98.42	100.56
-45	100.56	98.42	100.56
-38.57	100.56	98.42	100.56
-32.14	100.56	98.42	100.56
-25.71	100.56	98.42	100.56
-19.29	100.56	98.42	100.56
-12.86	100.56	98.42	100.56
-6.43	100.56	98.42	100.56
0	100.56	98.42	100.56
6.43	100.56	98.42	100.57
12.86	100.56	98.42	100.57
19.29	100.56	98.42	100.57
25.71	100.56	98.42	100.57
32.14	100.56	98.42	100.57
38.57	100.56	98.42	100.57
45	100.56	98.42	100.57
51.43	100.56	98.42	100.57
57.86	100.56	98.42	100.57
64.29	100.56	98.42	100.57
70.71	100.56	98.43	100.57
77.14	100.56	98.43	100.57
83.57	100.56	98.43	100.57
90	100.53	98.29	100.46

Max	100.57
Min	98.29

c. As a percentage of serial model

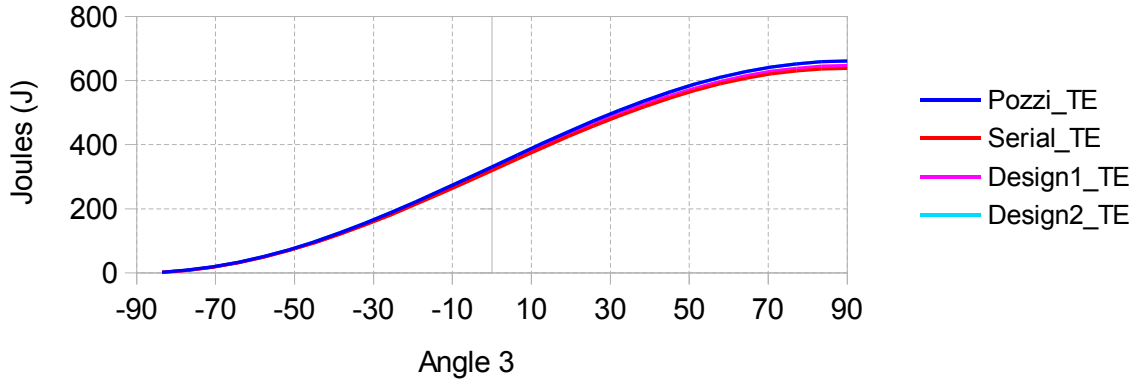


Varying angle 3, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
-83.57	2.078568	2.008751	2.034150	2.078568
-77.14	8.288099	8.009712	8.110986	8.288100
-70.71	18.550404	17.927320	18.153990	18.550407
-64.29	32.736266	31.636697	32.036706	32.736271
-57.86	50.667062	48.965221	49.584329	50.667070
-51.43	72.117021	69.694703	70.575912	72.117032
-45	96.816061	93.564134	94.747145	96.816077
-38.57	124.453200	120.272980	121.793690	124.453220
-32.14	154.680460	149.484940	151.375010	154.680490
-25.71	187.117280	180.832240	183.118660	187.117310
-19.29	221.355260	213.920210	216.625000	221.355300
-12.86	256.963360	248.332280	251.472170	256.963410
-6.43	293.493310	283.635230	287.221490	293.493360
0	330.485220	319.384630	323.422910	330.485290
6.43	367.473450	355.130470	359.620720	367.473520
12.86	403.992390	390.422780	395.359270	403.992480
19.29	439.582390	424.817340	430.188740	439.582480
25.71	473.795510	457.881270	463.670740	473.795610
32.14	506.201170	489.198450	495.383910	506.201280
38.57	536.391570	518.374780	524.929170	536.391690
45	563.986840	545.043150	551.934750	563.986980
51.43	588.639800	568.868030	576.060900	588.639950
57.86	610.040330	589.549720	597.004120	610.040500
64.29	627.919290	606.828120	614.501030	627.919470
70.71	642.051880	620.485990	628.331630	642.052080
77.14	652.260480	630.351670	638.322090	652.260690
83.57	658.416880	636.301250	644.346950	658.417110
90	660.443890	638.260140	646.330670	660.444140

Max	660.444140
Min	2.008751

a. Total energy consumed per model

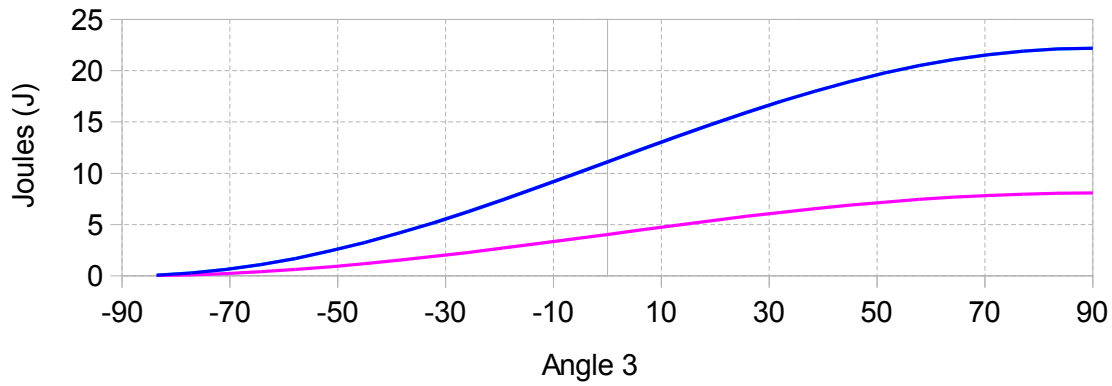


Varying angle 3, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	0.069816	0.025398	0.069817
-77.14	0.278387	0.101273	0.278388
-70.71	0.623084	0.226670	0.623087
-64.29	1.099569	0.400009	1.099574
-57.86	1.701841	0.619108	1.701849
-51.43	2.422318	0.881209	2.422329
-45	3.251927	1.183011	3.251943
-38.57	4.180220	1.520710	4.180240
-32.14	5.195520	1.890070	5.195550
-25.71	6.285040	2.286420	6.285070
-19.29	7.435050	2.704790	7.435090
-12.86	8.631080	3.139890	8.631130
-6.43	9.858080	3.586260	9.858130
0	11.100590	4.038280	11.100660
6.43	12.342980	4.490250	12.343050
12.86	13.569610	4.936490	13.569700
19.29	14.765050	5.371400	14.765140
25.71	15.914240	5.789470	15.914340
32.14	17.002720	6.185460	17.002830
38.57	18.016790	6.554390	18.016910
45	18.943690	6.891600	18.943830
51.43	19.771770	7.192870	19.771920
57.86	20.490610	7.454400	20.490780
64.29	21.091170	7.672910	21.091350
70.71	21.565890	7.845640	21.566090
77.14	21.908810	7.970420	21.909020
83.57	22.115630	8.045700	22.115860
90	22.183750	8.070530	22.184000

Max	22.184000
Min	0.025398

b. Difference from serial model

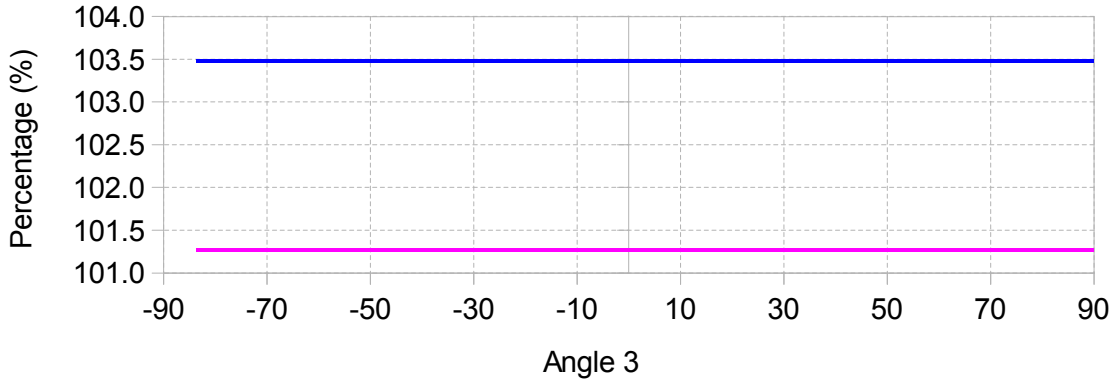


Varying angle 3, angle(2) = 0, heavy motors, 40s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	103.48	101.26	103.48
-77.14	103.48	101.26	103.48
-70.71	103.48	101.26	103.48
-64.29	103.48	101.26	103.48
-57.86	103.48	101.26	103.48
-51.43	103.48	101.26	103.48
-45	103.48	101.26	103.48
-38.57	103.48	101.26	103.48
-32.14	103.48	101.26	103.48
-25.71	103.48	101.26	103.48
-19.29	103.48	101.26	103.48
-12.86	103.48	101.26	103.48
-6.43	103.48	101.26	103.48
0	103.48	101.26	103.48
6.43	103.48	101.26	103.48
12.86	103.48	101.26	103.48
19.29	103.48	101.26	103.48
25.71	103.48	101.26	103.48
32.14	103.48	101.26	103.48
38.57	103.48	101.26	103.48
45	103.48	101.26	103.48
51.43	103.48	101.26	103.48
57.86	103.48	101.26	103.48
64.29	103.48	101.26	103.48
70.71	103.48	101.26	103.48
77.14	103.48	101.26	103.48
83.57	103.48	101.26	103.48
90	103.48	101.26	103.48

Max	103.48
Min	101.26

c. As a percentage of serial model

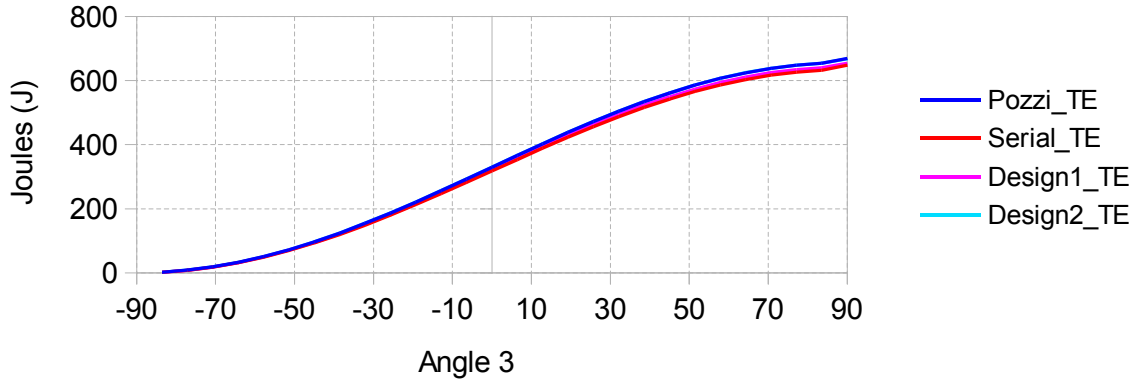


Varying angle 3, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Serial_TE	Design1_TE	Design2_TE
-83.57	2.072224	2.002365	2.028095	2.072350
-77.14	8.262722	7.984166	8.086767	8.263229
-70.71	18.493307	17.869841	18.099499	18.494447
-64.29	32.634760	31.534511	31.939833	32.636786
-57.86	50.508460	48.805556	49.432965	50.511626
-51.43	71.888634	69.464785	70.357948	71.893192
-45	96.505201	93.251191	94.450471	96.511406
-38.57	124.047180	119.864230	121.406200	124.055280
-32.14	154.166590	148.967630	150.884590	154.176850
-25.71	186.482870	180.193580	182.513210	186.495530
-19.29	220.587620	213.147430	215.892400	220.602950
-12.86	256.049810	247.412610	250.600320	256.068050
-6.43	292.421160	282.555900	286.198270	292.442560
0	329.241780	318.132860	322.236220	329.266600
6.43	366.046030	353.693480	358.258450	366.074520
12.86	402.368310	388.787810	393.809310	402.400720
19.29	437.748950	422.971620	428.438970	437.785550
25.71	471.740020	455.812010	461.709070	471.781050
32.14	503.910950	486.892890	493.198220	503.956660
38.57	533.853930	515.820140	522.507340	533.904580
45	561.189100	542.226660	549.264690	561.244940
51.43	585.569260	565.776910	573.130490	585.630550
57.86	606.684310	586.171210	593.801260	606.751290
64.29	624.265100	603.149440	611.013610	624.338030
70.71	638.086820	616.494360	624.547530	638.165970
77.14	647.971880	626.034320	634.229220	648.057480
83.57	653.792040	631.645420	639.933190	653.884350
90	668.828480	647.631790	653.930740	668.332110

Max	668.828480
Min	2.002365

a. Total energy consumed per model

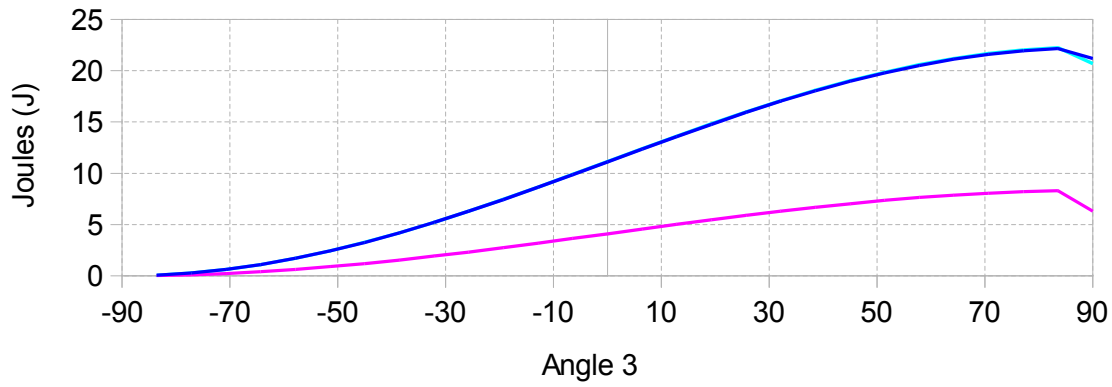


Varying angle 3, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	0.069859	0.025730	0.069986
-77.14	0.278557	0.102602	0.279063
-70.71	0.623466	0.229658	0.624606
-64.29	1.100249	0.405322	1.102275
-57.86	1.702904	0.627409	1.706070
-51.43	2.423849	0.893163	2.428407
-45	3.254010	1.199280	3.260215
-38.57	4.182950	1.541970	4.191050
-32.14	5.198960	1.916960	5.209220
-25.71	6.289290	2.319630	6.301950
-19.29	7.440190	2.744970	7.455520
-12.86	8.637200	3.187710	8.655440
-6.43	9.865260	3.642370	9.886660
0	11.108920	4.103360	11.133740
6.43	12.352550	4.564970	12.381040
12.86	13.580500	5.021500	13.612910
19.29	14.777330	5.467350	14.813930
25.71	15.928010	5.897060	15.969040
32.14	17.018060	6.305330	17.063770
38.57	18.033790	6.687200	18.084440
45	18.962440	7.038030	19.018280
51.43	19.792350	7.353580	19.853640
57.86	20.513100	7.630050	20.580080
64.29	21.115660	7.864170	21.188590
70.71	21.592460	8.053170	21.671610
77.14	21.937560	8.194900	22.023160
83.57	22.146620	8.287770	22.238930
90	21.196690	6.298950	20.700320

Max	22.238930
Min	0.025730

b. Difference from serial model

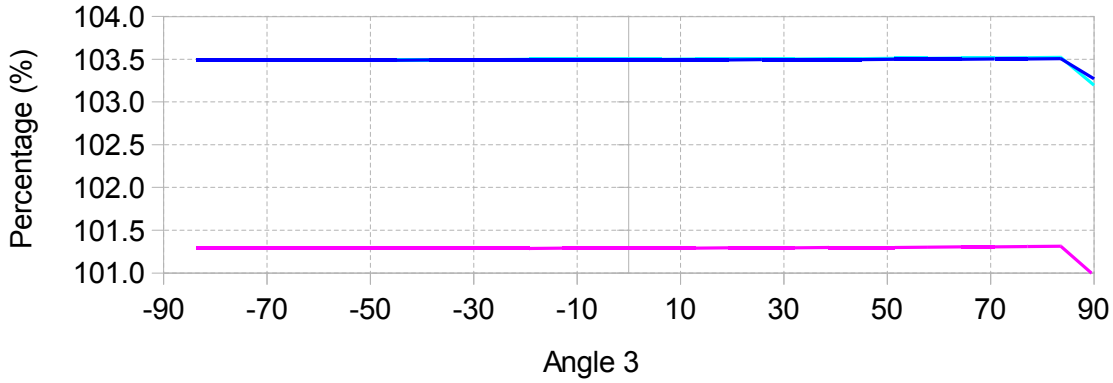


Varying angle 3, angle(2) = 0, heavy motors, 2s

Angle	Pozzi_TE	Design1_TE	Design2_TE
-83.57	103.49	101.29	103.5
-77.14	103.49	101.29	103.5
-70.71	103.49	101.29	103.5
-64.29	103.49	101.29	103.5
-57.86	103.49	101.29	103.5
-51.43	103.49	101.29	103.5
-45	103.49	101.29	103.5
-38.57	103.49	101.29	103.5
-32.14	103.49	101.29	103.5
-25.71	103.49	101.29	103.5
-19.29	103.49	101.29	103.5
-12.86	103.49	101.29	103.5
-6.43	103.49	101.29	103.5
0	103.49	101.29	103.5
6.43	103.49	101.29	103.5
12.86	103.49	101.29	103.5
19.29	103.49	101.29	103.5
25.71	103.49	101.29	103.5
32.14	103.5	101.3	103.5
38.57	103.5	101.3	103.51
45	103.5	101.3	103.51
51.43	103.5	101.3	103.51
57.86	103.5	101.3	103.51
64.29	103.5	101.3	103.51
70.71	103.5	101.31	103.52
77.14	103.5	101.31	103.52
83.57	103.51	101.31	103.52
90	103.27	100.97	103.2

Max	103.52
Min	100.97

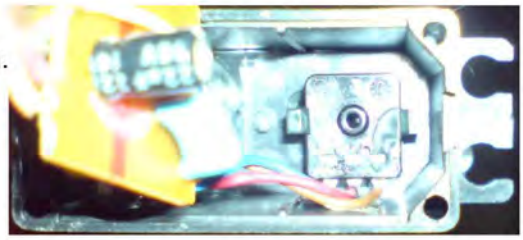
c. As a percentage of serial model



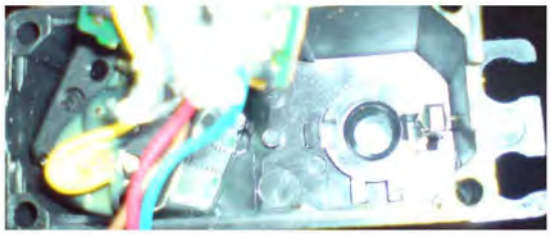
APPENDIX D: Modifying Servos

The most critical aspect of the scaled model builds were the fixed location of the motors at the base. The limited range in motion (about 200º) of the servos did not allow a straightforward placement and attachment to torque transfer links. This was a problem since all servos would then have a common reference position with regard to a global reference frame, and the robot joint angle that was controlled would then be with respect to that global reference frame. A reference to the previous link was needed, and that was accomplished by placing the servo's feedback sensor (potentiometer) at the joint that it controlled. The mechanical stop on the servo motor was removed so that the motor axis could rotate a full 360º. The servos could be placed at the base and control its intended axis which was further away along the arm linkage, using a series of movable links. Since additional mechanical parts were added between the motor output and the sensor output, the motor could be driven in a wrong direction that increased the error, and could potentially destroy it and other links if pushed to the limit of rotation. The power and ground wires to the potentiometer would then have to be swapped to solve that problem. The potentiometer could also be changed, with another having the same range of resistance (to maintain the exact same control circuit). For instance a ten turn potentiometer could be used for finer control. The modifications are illustrated in the following figure.

a.

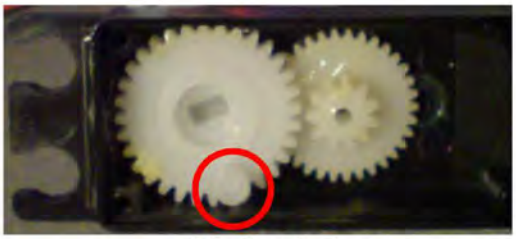


b.



c.

d.



Mechanical stop

e.

Servo modifications

a. Potentiometer
b. Potentiometer removed
c. Wired feedback sensor
d. Mechanical limitation on output
e. Modified servo with extention cable for feedback sensor