Title
The Kinematic Synthesis of Spatial Six-bar Mechanisms

Author
Wang, Peter Lee-Shien

Publication Date
2018

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

The Kinematic Synthesis of Spatial Six-Bar Mechanisms

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mechanical and Aerospace Engineering

by

Peter Lee-Shien Wang

Dissertation Committee:
Professor J. Michael McCarthy, Chair
Professor Haithem Taha
Professor Lorenzo Valdevit

2018

DEDICATION

To my wife Daria, my brothers Joey and Jimmy, my parents Melina and Jy-An, and my cat Sammy.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. J. Michael McCarthy for his support to me in my research and for his commitment to teaching kinematics. The robotics and automation lab is a place in which I thank Professor Haithem Taha for his inspiration and support of my research, his desire for a flapping wing mechanism to control both swing and pitch inspired the fundamental concept that founded the idea of the spatial six-bar mechanism. I thank Professor Lorenzo Valdevit and the rest of the IDMI team for providing me access and assisting with the use of the rapid prototyping equipment.

I next thank Benjamin Lui and Sihao Xu for their help in generating the geometric models. I thank Carrie Brewer and David Kincade for their help in creating the prototypes of the flapping wing mechanisms. I thank my labmates Jeff Glabe, Shrama Ghosh, Yang Liu and Dina Abulon for the great conversation and ideas.

CURRICULUM VITAE

Peter Lee-Shien Wang

EDUCATION

Doctor of Philosophy in Mechanical Engineering **2018**
University of California, Irvine *Irvine, California*

Master of Science in Civil Environmental Engineering **2009**
Leland Stanford Junior University *Stanford, California*

Bachelor of Science in Engineering **2008**
Harvey Mudd College *Claremont, California*

RESEARCH EXPERIENCE

Graduate Research Assistant **2015–2018**
University of California, Irvine *Irvine, California*

Undergraduate Research Assistant **2006–2008**
Harvey Mudd College *Claremont, California*

Student Researcher **2004,2006**
Oak Ridge National Laboratory *Oak Ridge, Tennessee*

TEACHING EXPERIENCE

Teaching Assistant **2016–2017**
University of California, Irvine *Irvine, California*

Teaching Assistant **2006–2008**
Harvey Mudd College *Claremont, California*

REFEREED JOURNAL PUBLICATIONS

Synthesis of a Flapping Wing Mechanism Using a Constrained Spatial RRR Serial Chain **2018**
Journal of Mechanisms and Robotics

Design of a Flapping Wing Mechanism to Coordinate Both Wing Swing and Wing Pitch **2018**
Journal of Mechanisms and Robotics

Design of a Spatial RPR-2SS Valve Mechanism **2018**
Journal of Mechanisms and Robotics

REFEREED CONFERENCE PUBLICATIONS

Design of a Spatial Six-Bar Flapping Wing Mechanism for Combined Control of Swing and Pitch **Aug 2017**
ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference

ABSTRACT OF THE DISSERTATION

The Kinematic Synthesis of Spatial Six-Bar Mechanisms

By

Peter Lee-Shien Wang

Doctor of Philosophy in Mechanical and Aerospace Engineering

University of California, Irvine, 2018

Professor J. Michael McCarthy, Chair

This dissertation provides a new methodology for the kinematic synthesis of spatial linkages constructed from six links and seven spatial joints, known as spatial six-bar linkages. Kinematic synthesis uses the required movement of components of the linkage to define geometric constraint equations that are solved to determine thes dimensions of the links, also known as dimensional synthesis. Here a new methodology for the dimensional synthesis spatial six-bar linkages is introduced that combines robotics with the kinematic synthesis of spatial constraints.

The new methodology defines the task of the linkage in terms of the movement of a spatial three-link serial chain, which is then constrained to define the six-bar linkage. The focus is on three spatial chains constructed from revolute, or hinged joints, and prismatic, or sliding, joints. These chains are denoted as the RRR, the RPR, and the PRP spatial serial chains. The spatial six-bar linkage is obtained by finding points in the moving links that lie on a sphere, which means that they can be constrained by a link connecting two spherical, or ball joints, known as an SS dyad.

The result of this research is the ability to design mechanical devices that provide new capabilities. This is demonstrated by the design of a new flapping wing mechanism that coordinates wing swing and wing pitch based on the RRR serial chain, the design of a new

valve mechanism that coordinates rotational and sliding movement to seal and close a foam injection port based on the RPR serial chain, and the biomimetic deployment of a patagium, the skin flap of a flying squirrel, based on the PRP serial chain.

Chapter 1

Introduction

1.1 Overview

This dissertation presents a new method for the kinematic synthesis of two-loop spatial mechanisms that have six bars and seven joints. Kinematic synthesis is a mathematical procedure that computes the linkage dimensions from constraint equations obtained from designer prescribed configurations of that achieve a task. Figure 1.1 is an illustration of one of the spatial six mechanisms that we can design with this new approach. It consists of a serial chain with three revolute, or hinged, joints (denoted by R) that is constrained by two links with spherical, or ball, joints at either end. The system is denoted as an RRR-2SS mechanism. The two other spatial mechanism are obtained by replacing the RRR serial chain by RPR and PRP serial chains, where P denotes a sliding, or prismatic, joint. These are denoted as the RPR-2SS and the PRP-2SS linkages, respectively.

This dissertation also shows how this synthesis methodology can provide new devices for a range of applications. Three examples are a flapping wing mechanism that controls both the flapping and pitch angles, a self-cleaning valve for a foam injection port, and a model for

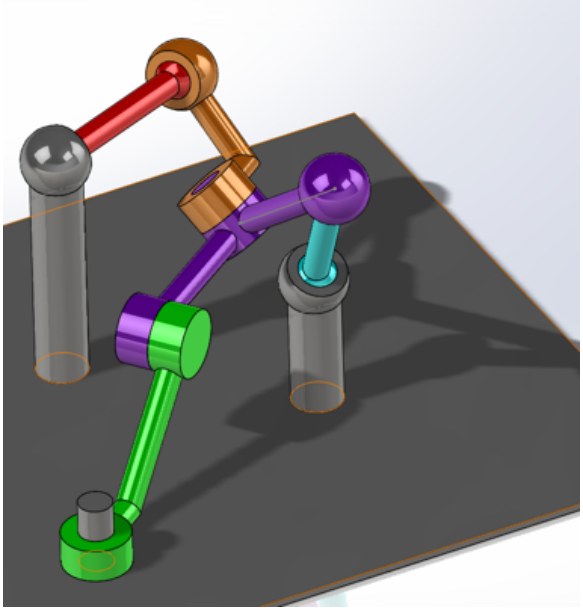


Figure 1.1: RRR-2SS Spatial Six-bar Mechanism, [90].

deploying patagium, or skin flap, of a flying squirrel in a jump-glide movement.

In this chapter, we present the literature on kinematic synthesis that is the foundation for this synthesis methodology, as well as the state-of-the-art for these new applications.

1.2 Synthesis of Mechanisms

1.2.1 Planar Six-bar Mechanisms

Finite position synthesis theory computes the dimensions of a linkage that moves an end-effector through a given set of task positions. In 1886 Schoenflies [75] and Burmester [10] were the first to state and the solve this synthesis problem. Burmester viewed a planar four bar linkage as a floating link with two points constrained to two circles, where the centers represent the ground pivots. He used graphical constructions to solve for the centers of the circles for up to five positions of the floating link. In 1948, Svoboda[82] created function

generators by fitting a linkage to the obtain a specific input and output that represented the target function. Freudenstein [35] developed a method which used the loop equations to fit a set of input/output angles to create a four bar function generator. This was further expanded by Denavit and Hartenburg [41]. McLarnan [55] applied the loop equations to Watt II, Stephenson II, and Stephenson III six-bar linkages and generated numerical solutions up to 8 input/output angles. Rao and Sandor [68] continued to build off of Freudenstein's work and generated an algebraic solution to six bar gear function generator of up to six input/output angles. Dingrah [26] applied Burmester-Ball theories to the Watt II, Stephenson II, and III six-bar linkages and generated solutions for up to nine input/output angles and solved the equations using homotopy continuation.

Aside from algebraic solutions, numerical optimization techniques have been applied and successfully used to find specific linkage solutions to specific functions. Hwang et. al [43] and Sancibrian [71] both have developed optimization methods that optimize a known linkage's dimensions to adjust its input and outputs to fit a desired curve. Bulatovic [9] applied the cuckoo search algorithm to optimize a Stephenson III six bar linkage to fit a desired path using 20 design variables. Shiakolas [77] utilized a technique called differential evolution to a Stephenson III linkage to obtain a specific function. Recently Tsuge, Plecnik and McCarthy [87] combined the 11 point synthesis equations with optimization techniques six bar linkage in order to utilize 60 accuracy points and applied it to a Stevenson III linkage to generate a specific trajectory.

Recently Plecnik and McCarthy [62] began to apply isotropic coordinates in combination with polynomial homotopy continuation to find the complete generalized solutions for the six bar linkages, Stephenson III [66], and Stephenson II [63] and found 834,441 and 1,521,037 non singular solutions, respectively. For the Watt I linkage [61] the generalized 8 position synthesis was not able to find a complete solution, but some solutions were found using Newton's Methods.

Bai [4] generated a unified formulation for motion generation for Stephenson linkages I, II, and III. Zhao, Purwar, and Ge [100] have developed a unified method of synthesizing four and six bar planar linkages with both revolute and prismatic joints using planar quaternions. Nafees has created dimensional synthesis of a six bar Stephenson II linkage [56] and Stephenson III linkage [57]

A RRR chain can be constrained using two RR links to reduce the degrees of freedom of the system to one [54, 86]. Soh and McCarthy [78] applied this process to Watt I and Stephenson I, II, and III six bar linkages which allows it to pass through a set of fix specified task positions. Plecnik and McCarthy [65] further developed this concept and created for the Stephenson II and III that utilized 11 path positions. Soh and Ying explore the configurations and design of one degree of freedom planar six bar mechanisms that utilize a prismatic joint. [79]

The research presented in this dissertation expands on the six-bar synthesis theory developed by Soh and McCarthy by constraining a spatial serial chain using SS dyads and creating a spatial six-bar mechanism [90, 89], see Figure 1.1.

1.2.2 Single Loop Spatial Mechanisms

These mechanisms consist of a linkage that moves through space and can have single or multiple degrees of freedom. They are composed of only binary links and form a single closed loop.

Denevit and Hartenberg [24] presented synthesis of the RSSR function generator with six prescribed input-output angles. Wilson[93] developed a synthesis method derived from curves traced by RS, SS, SR, and RR links and solving the resulting system of equations. Suh[81] applied kinematic inversion to develop a generalized matrix of functional displacement and

synthesizes a RSSR function generator using six input-output angles. Roa and Sandor[69] expanded on these developments and derived synthesis equations for seven and eight input/output angles. Gupta [40] analyzed the kinematics of RSSR mechanisms using coordinate transformation matrices.

Optimization synthesis methods have been pursued by Cossalter[20] who utilized a numerical optimization routine and Gupta's kinematic analysis to synthesize RSSR and RSSP mechanisms that generate specific functions. More recently Chu [17] used harmonic characteristic components and a numerical atlas database to synthesize spatial function generators using RSSR and RRSS. Mazzotti et al [53] performs the dimensional synthesis of an RSSR mechanism through defining design parameter ranges and optimizing for force transmission.

In 1995 Innocenti [44] found the polynomial solution of the spatial burmester problem finding the 20th order polynomial solution with a numerical example. This was followed on and expanded upon by Liao and McCarthy in 2001 who reduced the 60 x 60 resultant matrix to a 10 x 10 and use it to solve for a rigid body with a movement through seven specified spatial positions [50].

Yang [95] developed displacement equations for RCRCR mechanisms through modifications of Denavit and Harenberg's methods using 3 x 3 matrices with dual numbers. Duffy [28] confirms Yang's findings and synthesizes displacement equations for the RRCRC [29] and RCRRC [30]. Yuan [98] also confirmed Yang's finding of the RCRCR mechanism and also develops a solution to the RRCCR mechanism [97]. Pamidi and Freudenstein [59] algebraically derived the rotatability of the input and output links. In 2009 Davitashivili [22] calculated equations for trajectory of the connecting rods of a RSSRR mechanism.

In recent years Cervantes-Sanchez [11] [13] has synthesized RPSPR and RRRCR spatial function generators with six input/output angles. Perez-Garcia [60] created a synthesis method for the RPRP spatial linkage. Bai and Angeles [3] synthesized CCCC and RCCC

linkages for four and five poses. D'Alessio et al [21] generated an RRSS spatial linkage to follow a specific motion through space via an optimization routine for a prosthetic knee. Wei and Dai [91] a single loop spatial eight bar linkage with two degrees of freedom with an exact straight line motion. Ge built off of synthesis techniques for 5-SS platform linkages [48, 36] to synthesize Bennett 4R linkage [37].

1.2.3 Multi-loop Spatial Mechanisms

Sandor et al [72] design an RSSR-SS spatial six-bar, see Figure 1.2 and Chiang [16] designed an RSCC-RRS spatial six-bar, where C denotes a cylindric joint, see Figure 1.3. Both have presented design methodologies for two loop spatial chains but instead of coordinating joint angles they guide an end-effector along a specified trajectory. Similarly, Chung [18] constructs a UR-2SS two loop spatial mechanism that supports an end-effector to draw a specified curve. Cervantes-Sanchez et al [12] analyze a spatial linkage with two loops in which they analyze an RSSR-SC linkage for the full range of motion of the input link. Zhang et al [99] created Schoenflies motion parallel mechanisms, a skewed 3-PRP and 3-PUP mechanisms and utilized a Lie group method for the displacement analysis. These mechanisms are in the form of a platform mechanism in which a platform is constrained to ground by three parallel linkages.

Avila and Cuenca [1] synthesized a 9-bar spatial mechanism, RRSSRRSSSRS, in which a 6R serial chain is constrained by three SS-Dyads to give a two degree of freedom system for the actuation of a prosthetic thumb, see Figure 1.4.

The multiloop spatial mechanisms above are examples of how to synthesize those specific mechanisms. This dissertation presents the first generalized method that allows for the synthesis of spatial six-bar mechanisms by defining a three link serial chain and constraining it with two SS dayds. This method is used to synthesize the RRR-2SS, RPR-2SS and PRP-

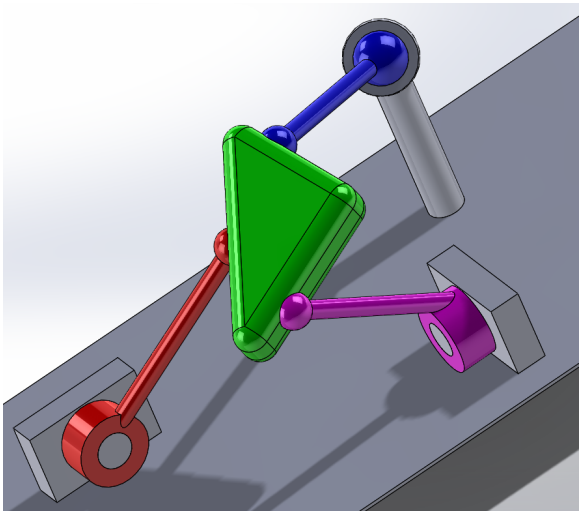


Figure 1.2: RSSR-SS from Sandor [72].

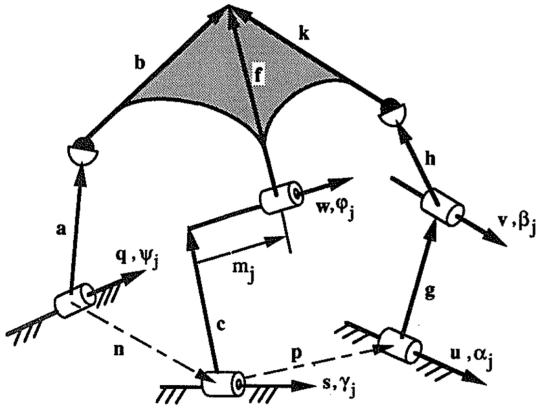


Figure 1.3: RSCC-RRS from Chaing [16].

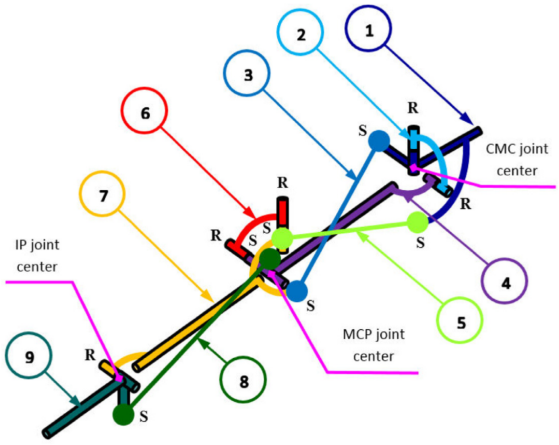


Figure 1.4: RRSSRRSSSRS from Avila [1].

2SS mechanisms. This methodology can be applied to generating other two loop spatial mechanisms of other compositions.

1.3 New Applications

1.3.1 Flapping Wing Mechanisms

Flapping Wing Micro Air Vehicles (FWMAV) are flying crafts that are smaller than 15cm and utilize flapping wings to achieve flight. Currently, the majority of FWMAV mimic insect's utilization of unconventional aerodynamics and compliant wings[32, 6]. The wings swing in a planar path without any out of plane motion[27, 58]. The pitch is adjusted passively utilizing the resistance with the air to position the wing. Control of the precise pitch angle has been not available due to the necessity of ultra low weight, thus prohibiting the additional mechanisms or servos necessary to manipulate the pitch of the wings[92, 27, 58, 74]. Recent research by Taha, et al has shown that the novel vibrational stabilization method developed may allow for improve flight and hovering stability[83, 84]. This method, however, requires a the wing pitch to be precisely positioned at every point in the wing's swing. Additional kinematic optimization studies have recommended that active pitch control be added to FWMAV[6, 94], but additional motors fail the weight requirement. So a flapping mechanism that can position both the swing and the pitch of the wing simultaneously needs to be developed[5]. Several FWMAV design have successfully flown such as the Purdue Mechanism[42], the Aerovironment Nano-Hummingbird[46], the Harvard Robo-Bee [51] and the DelFly mechanism[23]. The Nano-Hummingbird is shown in Figure 1.5. However, none of those designs have precise position control of the pitch wing with relation to the swing of the wing. Conn[19] and Balta [5] developed a mechanism that accomplishes the pitch control, but they are too heavy to fly. See Figures 1.6 and 1.7. Recently, the Robo Raven

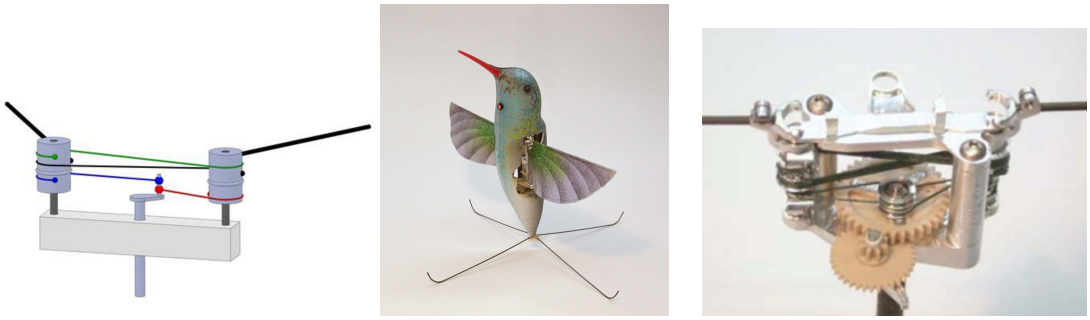


Figure 1.5: Nano Hummingbird by Aerovironmnet [46].

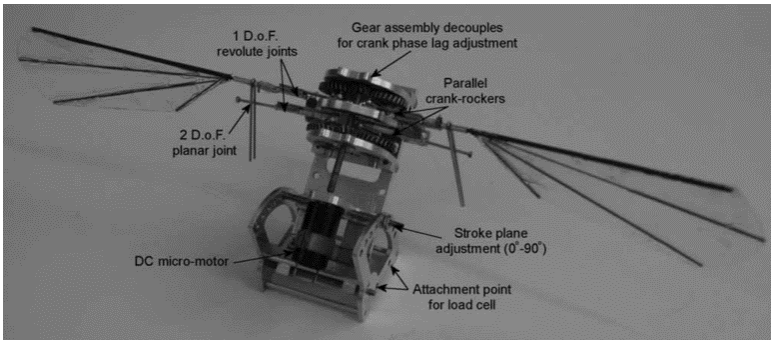


Figure 1.6: Flapping Mechanism with Pitch Control by Conn [19].

[38] and the Bat Bot [67] have utilized a different flying mode in which extreme deformation of large wings is used to achieve flight, see Figure 1.8.

Thus a novel mechanism is needed that can position both the swing angle and the pitch angle of the wing. The first iteration of this mechanism is shown in Wang [89], where a planar four bar linkage is combined with a spatial four bar linkage to create a spatial six bar linkage that controls both the wing swing and wing pitch, see Figure 1.9. The following

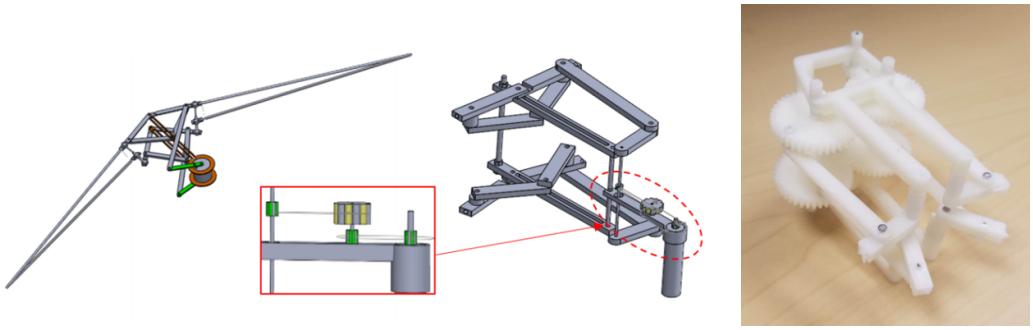
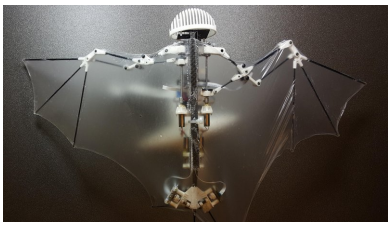


Figure 1.7: Flapping Wing Mechanism by Balta [5].



(a) Bat Bot [67]

(b) RoboRaven [38].

Figure 1.8: Flexible Flapping Wing Robots

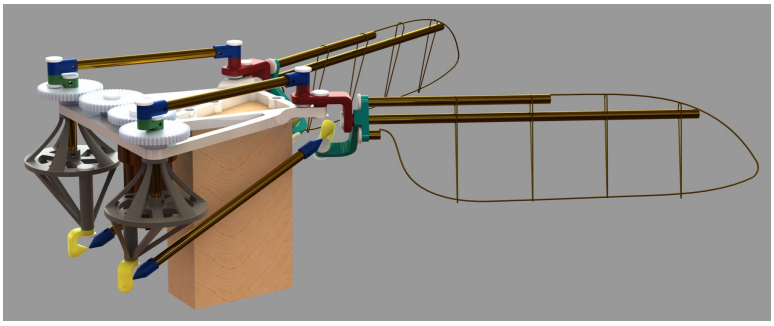


Figure 1.9: Planar-Spatial Flapping Wing Mechanism

design constrained a spatial RRR chain to improve packaging and control the input motor position, see Figure 1.10.

1.3.2 Soil Conditioning Injection Valve

Soil conditioning foam is used in tunnel boring machines (TBM) to adjust the properties of the excavated soil to reduce wear, and increase efficiency of the machines. [45] The soil conditioning foam lubricates the soil and can reduce its stickiness. The stickiness of the ground is caused by an electro-kinectic phenomenon [52] that is found in very fine soil particles, such as clays. This phenomenon can cause significant build up on the cutting tools and cutterhead of the TBM which can dramatically increase the cutterhead torque, see Figure 1.11. The increase in energy required can be significant enough to overwhelm a TBM. [70] The ability to get the foam to mix with the soil is critical to TBM performance.

The soil conditioning foam is injected into the soil through the cutter head from ports shown

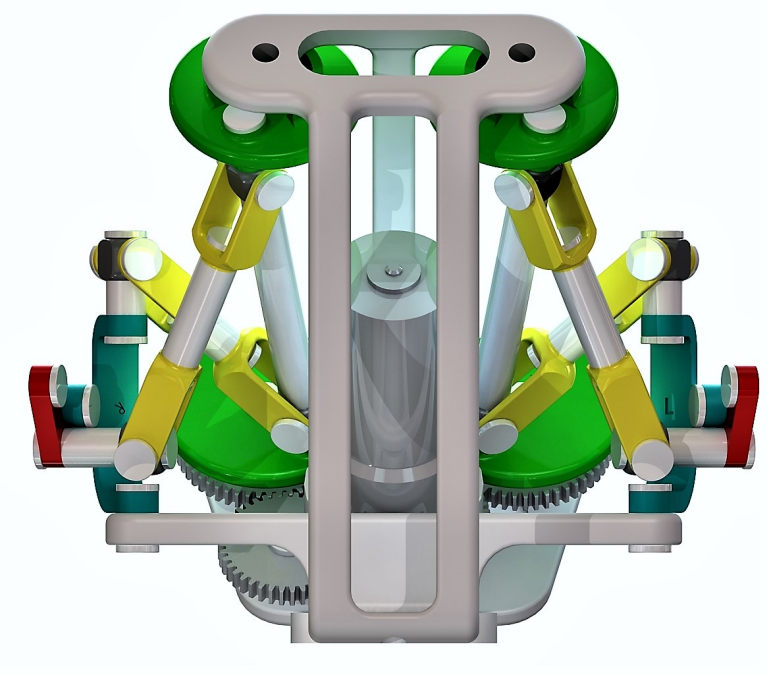


Figure 1.10: 3R-2SS Spatial Six-bar Flapping Wing Mechanism

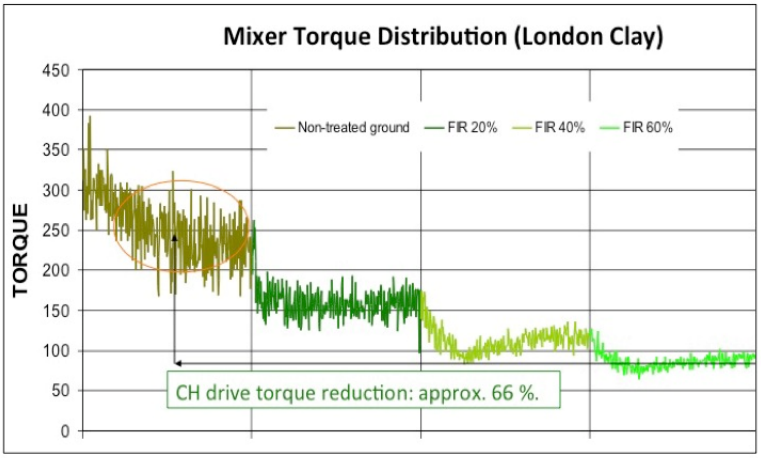


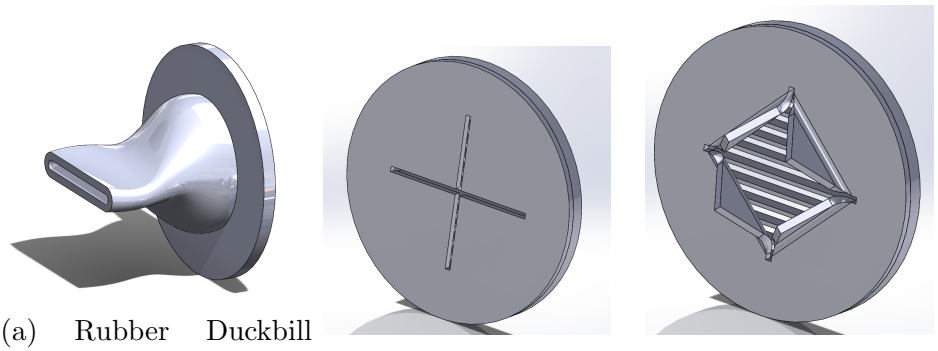
Figure 1.11: The effect of ground conditioner on TBM cutterhead torque.

Figure 1.12: TBM testing the foam injection ports on the surface.

in Figure 1.12 and 1.16. One of the major challenges resides in keeping these ports open and clear. Cohesive soil has a tendency not only to stick on flat metal surfaces but also to bridge and clog any small openings like foam the nozzle ports. It is therefore not unusual that thirty percent of the installed foam nozzle ports of a cutterhead will be clogged while mining through cohesive soils. [85]

Currently there are several check valve strategies used in TBM's foam injection system to prevent the soil from flowing back into the foam injection port. The most common systems utilize a deformable rubber check valve that only opens when flow of foam starts, see Figure 1.13. However, since these rubber valves must be protected inside a pipe to prevent damage to the rubber, the portion of exposed pipe will often become plugged. Then the injection ports are physically drilled out and the rubber check rendered useless thereafter. If they are not protected the rubber is subjected to excessive wear and will degrade rapidly.

Another methodology used is to have an injection port that is sealed with an O-Ring. When there is no foam flow the O-Ring seals tightly against the injection holes, see Figure 1.14. When foam is flowing the O-Ring will deform to allow the foam to escape, see Figure 1.15.



(a) Rubber Duckbill Closed (b) Rubber Flap Closed (c) Rubber Flap Open

Figure 1.13: Flexible Rubber Check Valves Used Inside of Pipe

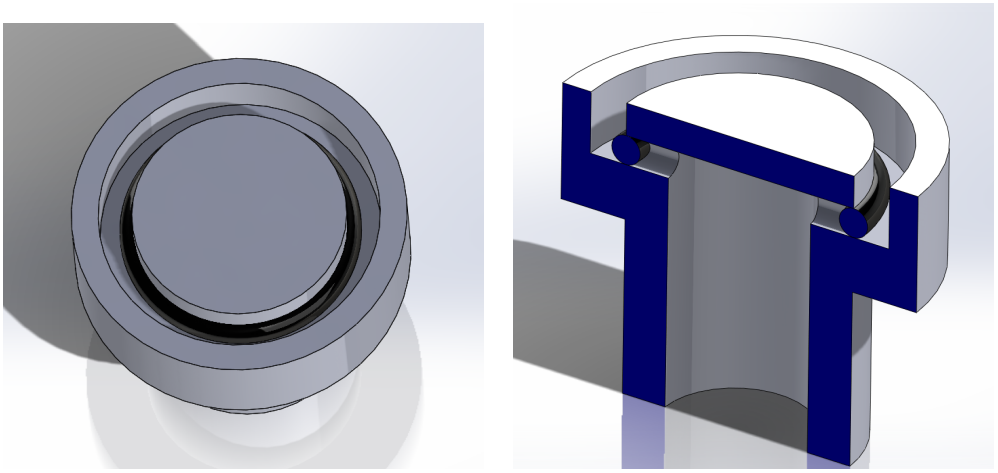


Figure 1.14: O-Ring Port Closed

Though, this addresses the issue of rubber wearing due to exposure to the mixing soil, and reduces the potential clogging area, there still exists the possibility to clog the ports and no way of cleaning them from inside the TBM once they are plugged. So like replacing the ports/valves the cleaning procedure for this type of valve must be performed from outside the TBM by sending person into the unstable opening in front of the TBM to perform the work.

So given that back flows and muck build up inside the ports is difficult to avoid in cohesive soils, a design for a self cleaning valve would help prevent the build up from reaching a point at which it would plug the port.

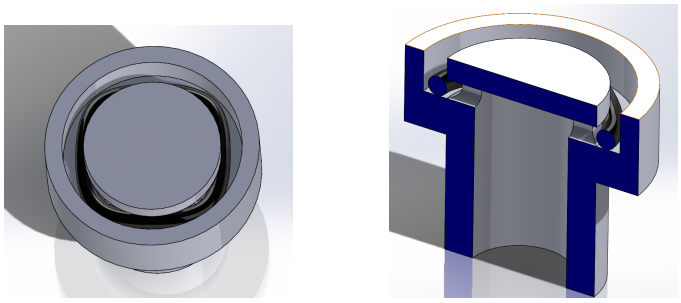


Figure 1.15: O-Ring Port Open

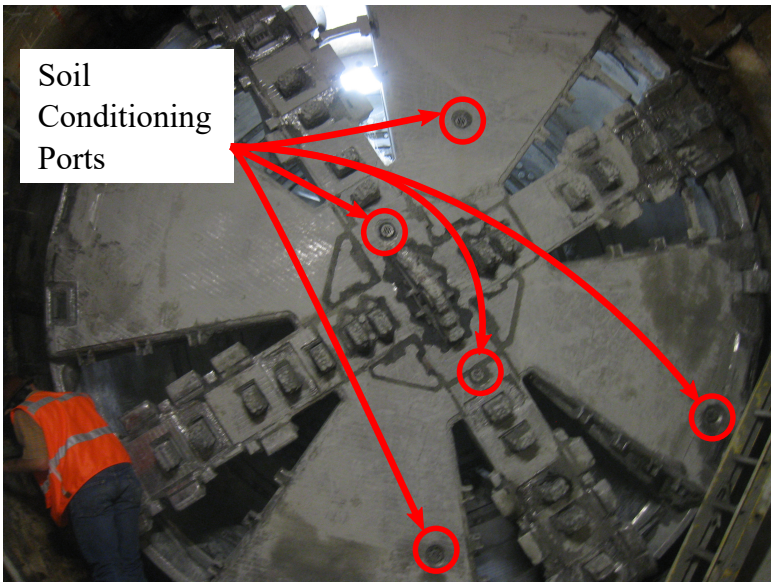


Figure 1.16: Photograph of a tunnel boring machine cutter with arrows identifying the soil conditioning ports.

The conceptual design of the self cleaning valve required a system driven by a check valve opening that both cleans the injection port and prevents back flow of soil into the port. An RPR serial chain satisfied these three criteria. The rotation, θ_1, of a pressure plate in the soil conditioning line is used to generate two output movements, (i) a slide d_2 of an array of studs that clean the port, and (ii) a rotation θ_3 that seals the port. This mechanism provides a regular cleaning operation to remove soil build-up from the soil conditioning ports when pressure is relieved during the ring build phase, or during any required down-time, which is a regular part of the tunneling process. Currently, ports are cleaned infrequently when there is a complete blockage. This regular cleaning operation provided by this mechanism

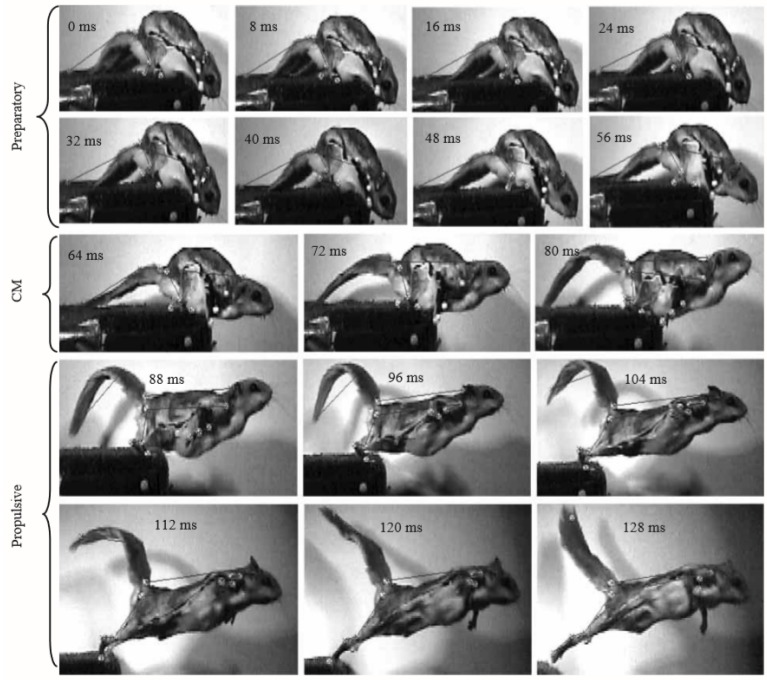


Figure 1.17: Essner High Speed Photos of Squirrel Jumping to Analyze the Kinematics [33].

can reduce wear and power consumption in mechanized tunneling [31, 85, 70, 45].

1.3.3 Jumping Gliding Mechanisms

Flying squirrels utilize mostly quadrapedal locomotion, however, they are best known for their ability to jump and glide [34]. During their leap the extend their limps stretching a skin flap that connects their forward and hind legs called the Patagium, see Figure 1.18.

Essner [33] describes three major forms of locomotion through the air that is performed by squirrels, leaping, parachuting and gliding. A high speed camera was used to evaluate the body kinematics of the squirrels, see Figure 1.17. Scheibe explored the kinematics of in the various stride phases and the dynamics of the jumping and gliding process for captive northern flying squirrels [73]. Youlantos [96] studied the locomotor and postural behavior of squirrels to understand when which types of locomotion were preferred.

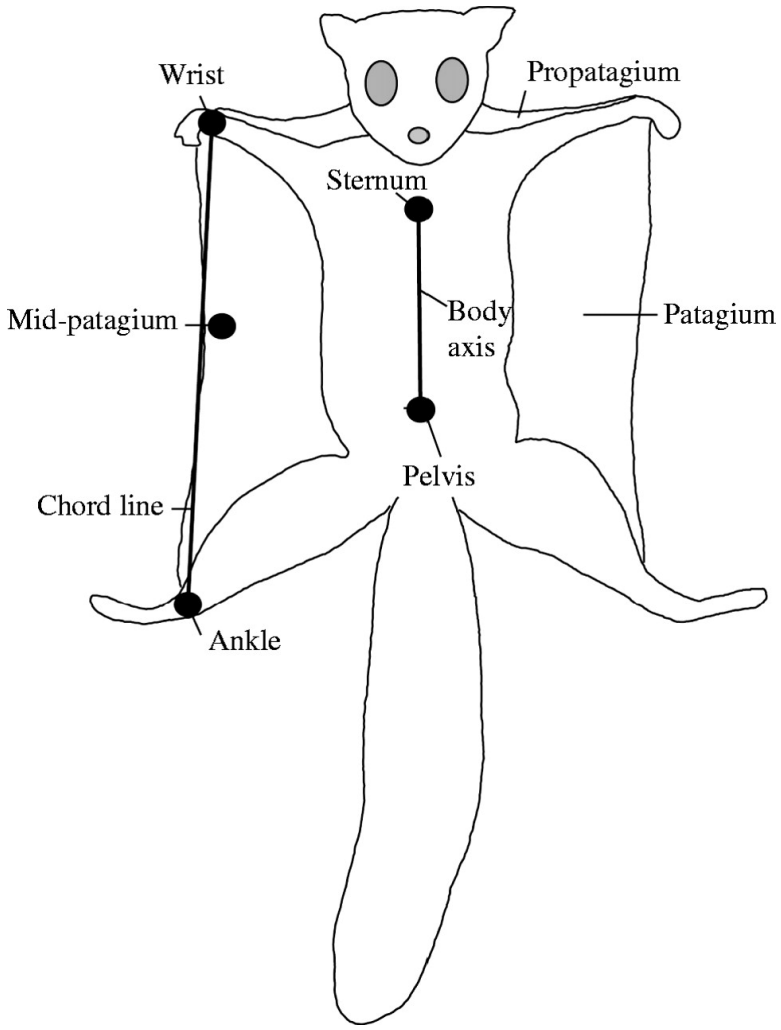


Figure 1.18: Anatomy of a Flying Squirrel [7].

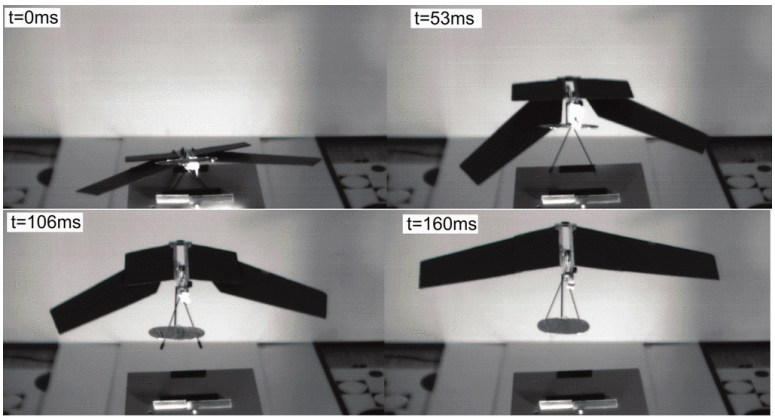


Figure 1.19: Kovac jump glider with unfolding wings. [47].

Bishop [7] studied the relationship between kinematics and gliding performance. Goldingay and Taylor studied the distance and height from which gliding squirrels jumped from in order to develop a standard for poles to allow squirrels to glide across roads [39]. Li [49] studied the aerodyanmic effect of manipulating a flexible membrane simulating a flying squirrel.

Kovac et al [47] created bat, butterfly and locust inspired jump gliding mechanisms. The locust inspired robot had wings that would unfold during the jump, see Figure 1.19. Desbiens [25] shows a jump glider that launches using a bent spring and has a wing that rotates backwards during the jumping phase, see Figure 1.20. Baek [2] demonstrates a jump-gliding mechanism which has the body and wings completely folded that deploy using an origami linkage, this allows for the mechanism to be compact while jumping then deploy the wings to glide to reduce air resistance during the jumping phase, see Figure 1.21. However Baek does not include a jumping mechanism as part of the design.

The flying squirrel mechanism developed takes advantage of a compact body shape while jumping to reduce the drag of the first jump then deploys the patagium towards the top of the jump, see Figure 1.22.

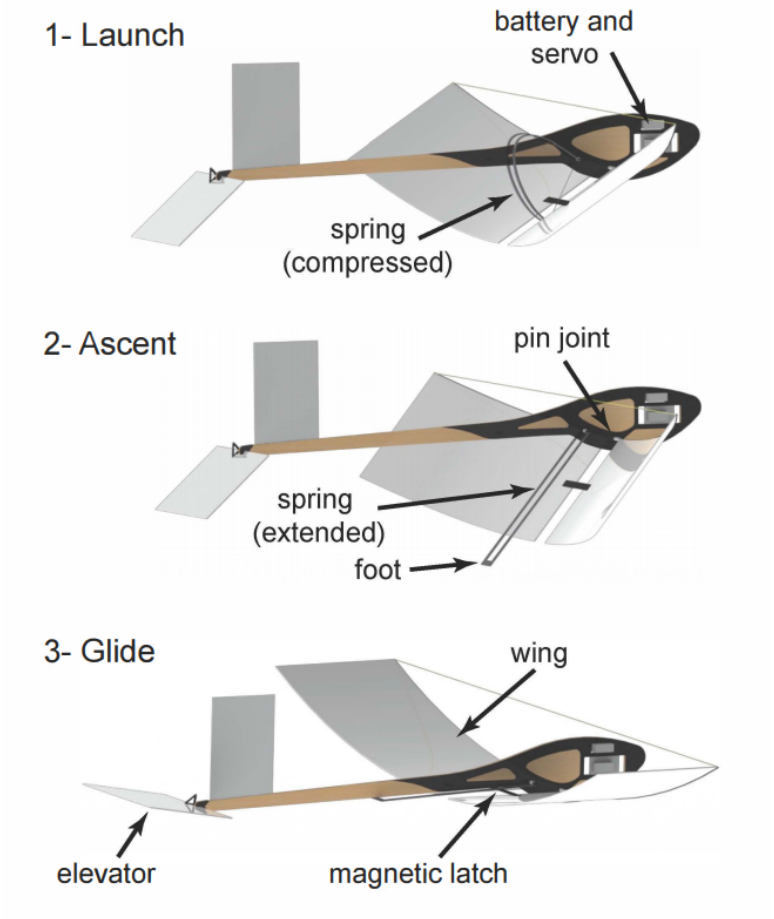


Figure 1.20: Desbiens jump glider. [25].

1.4 Contributions

This dissertation presents a new design methodology for the kinematic synthesis of spatial six bar linkages. The contributions are:

1. The first mathematical formulation of the kinematic synthesis of spatial six-bar function generators as three-link serial chains constrained by two SS dyads,

2. The first formulation of the synthesis of a spatial six-bar linkage by constraining a spatial RRR serial chain,

3. The first kinematic synthesis of a constrained RPR serial chain to obtain a spatial

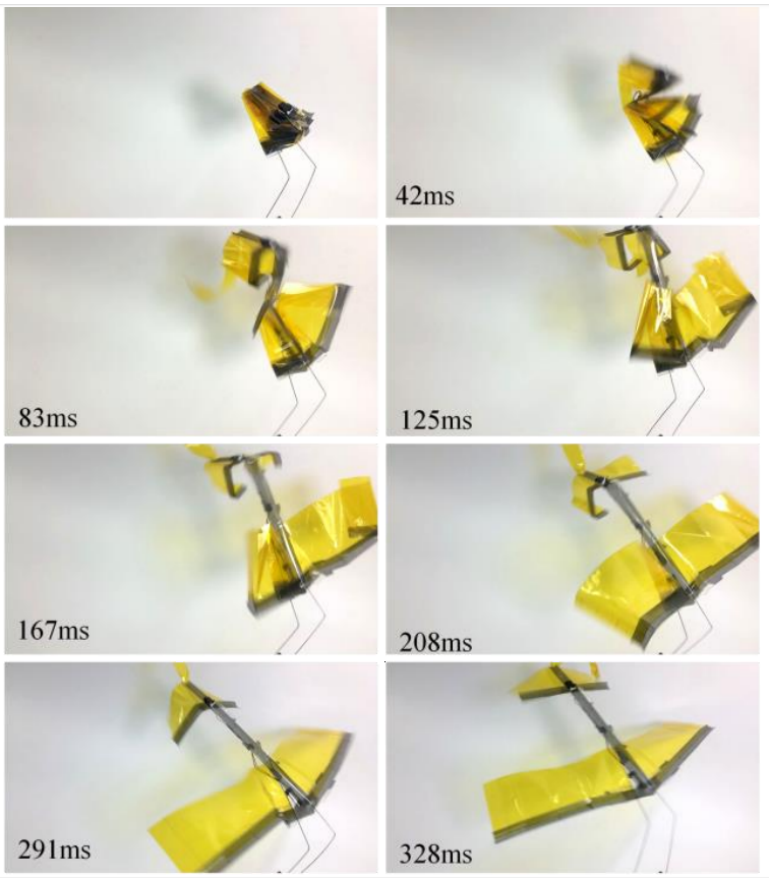


Figure 1.21: Baek origami jump glider. [2].

six-bar linkage,

4. The first kinematic synthesis of a constrained PRP serial chain,

5. The first demonstration of the synthesis of spatial six-bar linkages for a flapping wing mechanism, a valve closing mechanism, and biomimetic leg movement for jumping gliding.

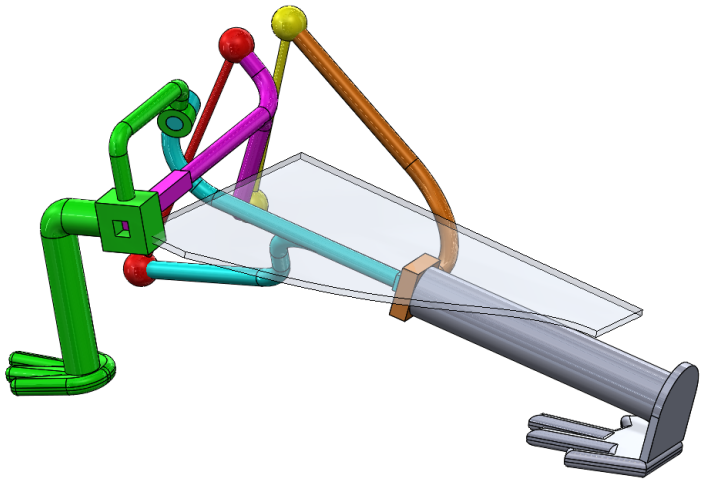


Figure 1.22: Left side of flying squirrel mechanism.

Chapter 2

Mathematical Background

This chapter discusses the mathematical formulations and background required to perform the synthesis of spatial six-bar linkages. The chapter covers formulation of the kinematic equations of the spatial serial chain, and the RR and SS dyads. The synthesis of planar four and six bar mechanisms are presented to provide a foundation for the synthesis of the spatial mechanisms. Finally the synthesis of the spatial six-bar mechanism is presented in detail.

2.1 Spatial Serial Chain

Spatial serial chains connect an end effector to a ground via a sequence of joints and links. The position of the end effector and each joint can be described using the Denevit-Hartenburg convention. First an axis is drawn along each of the joints, and the line which draws the shortest distance between the two sequential axes is found, this is also known as the common normal between the two axes. The rotation about and translation along axis i are θ_i and d_i, respectively. The angle between axis i and $i+1$ is α_i and the length of the common normal between the two axes is a_i.

By using the following coordinate screw matricies,

$$\mathbf{Z}(\theta_i, d_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{X}(\alpha_i, a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad i = 1, 2, 3 \tag{2.1}$$

the kinematic equations of a five link spatial serial chain can be described by

$$K = \mathbf{Z}(\theta_1, d_1)\mathbf{X}(\alpha_1, a_1)\mathbf{Z}(\theta_2, d_2)\mathbf{X}(\alpha_2, a_2)\mathbf{Z}(\theta_3, d_3)\mathbf{X}(\alpha_3, a_3)fZ(\theta_4, d_4)\mathbf{X}(\alpha_4, a_4)fZ(\theta_5, d_5) \tag{2.2}$$

The variables θ_i, d_i, α_i, and a_i can be arranged into a table called a Denavit-Hartenberg table.

Table 2.1: Denavit-Hartenberg table for the a serial chain.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	d_1	α_1	a_1
2	θ_2	d_2	α_2	a_2
3	θ_3	d_3	α_3	a_3
4	θ_4	d_4	α_4	a_4
5	θ_5	d_5	-	-

2.2 Planar Mechanism Synthesis

Planar mechanisms are mechanisms whose links and joints all remain within a single plane through out its movement. This forces the axis of rotation of the revolute joints to be perpendicular to the plane of movement and the axis of slide of prismatic joints to lie on the plane.

2.2.1 RR Dyad Algebraic Synthesis

An RR dyad is a rigid link with revolute joints at both ends—R denotes a revolute, or hinged, joint. To synthesize an RR dyad the ground frame G_1 and the moving frame M_1 where the two frames are connected by the RR dyad. The initial coordinates of the fixed and moving pivots are given by $\mathbf{A} = (A_x, A_y)$ and $\mathbf{B} = (B_x, B_y)$, respectively. The five positions of the moving frame M_1 are given by the the transformation matrix $[T_i], i = 1, .., 5$. The five coordinates of the moving pivot, $\mathbf{B}^i, i, = 1, \ldots, 7$ in each position of the moving frame is given by

$$\mathbf{B}^i = [T_i][T_i^-1]B^1, i = 1, \ldots, 5 \tag{2.3}$$

Now, knowing that **AB** is a fixed length the following constraint equations are formed,

$$(\mathbf{B}^i - \mathbf{A}) \cdot (\mathbf{B}^i - \mathbf{A}) = R^2, i = 1, \ldots, 5 \tag{2.4}$$

.

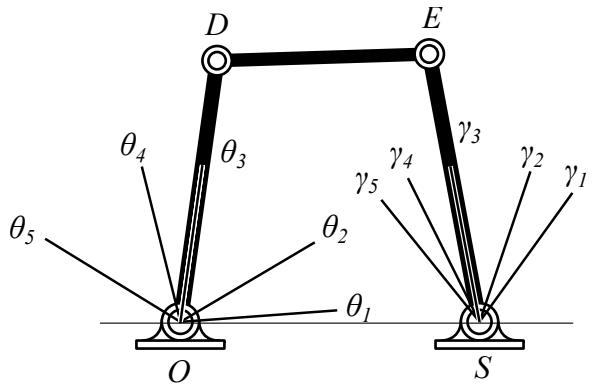


Figure 2.1: The RRRR linkage is shown with the two revolute joints connected to the ground link **OC** with **AB** and **BC** moving.

The equations can be simplified by subtracting the first equation in the set from the remaining four equations. This cancels the constant R^2 and the squared terms for the four coordinates of **AB**. The result is a set of design equations

$$(\mathbf{B}^i - \mathbf{A}) \cdot (\mathbf{B}^i - \mathbf{A}) - (\mathbf{B}^1 - \mathbf{A}) \cdot (\mathbf{B}^1 - \mathbf{A}) = 0, i = 1, \ldots, 5 \tag{2.5}$$

The four equations are solved simultaneously to give a maximum of four pairs of coordinates, $\mathbf{B}^1$ and **A**. The four equations can be put in the form a single univariate polynomial of the sixth order. However, two of the coefficients go to zero due to the circular cubic structure leaving a quartic polynomial, which yields a maximum of four solutions. [54]

2.2.2 Planar RRRR Function Generator

The RRRR function generator produces a given output angle γ for a given input angle θ, seen in Figure 2.1. The input link and the output link are connected by RR-Dyad **AB**. To synthesize the RSSR function generator five pairs of input and output angles must be selected.

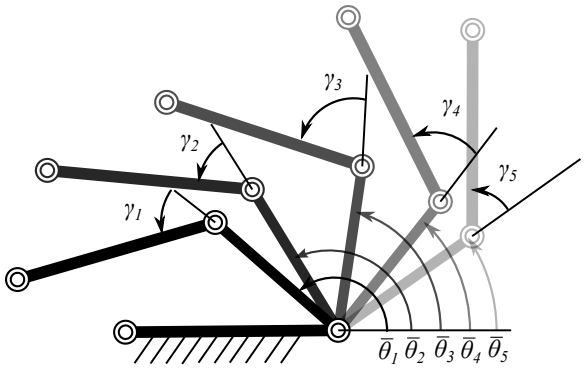


Figure 2.2: The RRRR linkage shown has link **OA** connected to ground with links **OC** and **BC** moving.

The linkage is then inverted as shown in Figure 2.2. Frame **OA** is fixed to ground. While frame **OC** moves by $\bar{\theta} = \pi - \theta$ and Frame **BC** continues to move by γ.

Now joint **A** is in the ground frame and joint **B** is in the moving frame. This is the configuration needed for RR-Dyad synthesis. The transformations from joint **A** to joint **B** are given by

$$T(\theta^i, \gamma^i) = Z(\theta^i, d_1)X(\alpha, a)Z(\gamma^i, d_2), i = 1, \ldots, 5. \tag{2.6}$$

The RR Dyad synthesis can now be solved by substituting the transformation into equations 2.17 and 2.19. Up to 4 sets of initial coordinates for joint **A** to joint **B** can be found and used to assemble the RRRR linkage.

2.2.3 Planar Six-bar Mechanism Synthesis

Planar six bar mechanisms come in two topologies: The Watt topology has the ternary links connected see Figure 2.3, The Stephenson topology has the ternary links connected by a binary link, see Figure 2.4.

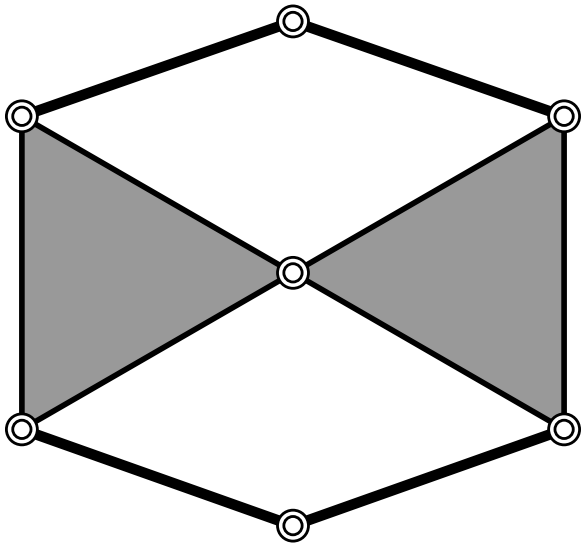


Figure 2.3: Planar Six-bar Watt Topology

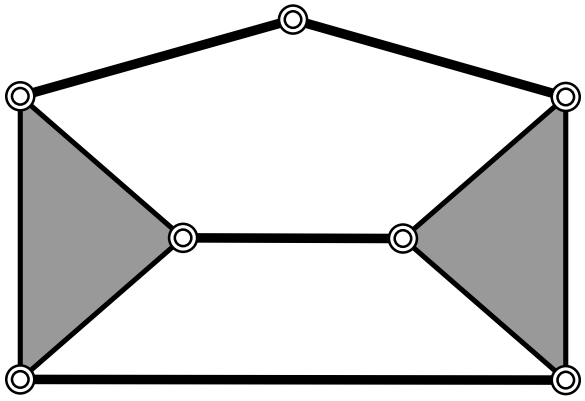


Figure 2.4: Planar Six-bar Stephenson Topology

The synthesis technique used in this dissertation is based off of the the planar six bar synthesis method that comes from Soh and McCarthy [78]. The synthesis technique first defines a planar RRR chain in five positions which is then constrained by two RR links. It was found that every inversion of the two planar six bar topologies, Watt and Stephenson, except Watt II, have a RRR backbone chain. This technique was expanded upon by Soh and Ying [79] by adding prismatic joints and expanding to eight bar linkages.

2.2.3.1 Planar RRR Chain

First a planar RRR chain must be found that can reach five task positions. The kinematic equations of the RRR chain are

$$D = \mathbf{BZ}(\theta_1, 0)\mathbf{X}(0, a_1)\mathbf{Z}(\theta_2, 0)\mathbf{X}(0, a_2)\mathbf{Z}(\theta_3, 0)\mathbf{X}(0, a_3)\mathbf{H} \tag{2.7}$$

where $\mathbf{B}$ and $\mathbf{H}$ are the distance of the base link to the world frame, and the task frame relative to the end effector fame, respectively. The set of five task positions are given as $[T_j], j = 1, \ldots, 5$. The inverse kinematics can be solved for the joint parameters $\mathbf{q_j} = (\theta_{1j}, \theta_{2j}, \theta_{3j}), j = 1, \ldots, 5$. This gives the RRR chain that can achieve the five given task positions.

2.2.3.2 Synthesis Equations

The RR dyads are then used to constrain the frames of the RRR chain, where the frames are

$$
\begin{aligned}
L_0 =& \mathbf{B} \\
L_1 =& \mathbf{BZ}(\theta_1, 0)\mathbf{X}(0, a_1) \\
L_2 =& \mathbf{BZ}(\theta_1, 0)\mathbf{X}(0, a_1)\mathbf{Z}(\theta_2, 0)\mathbf{X}(0, a_2) \\
L_3 =& \mathbf{BZ}(\theta_1, 0)\mathbf{X}(0, a_1)\mathbf{Z}(\theta_2, 0)\mathbf{X}(0, a_2)\mathbf{Z}(\theta_3, 0)\mathbf{X}(0, a_3)
\end{aligned} \tag{2.8}
$$

Knowing that two consecutive links cannot be constrained, the first RR dyad is chosen by connecting L_0L_2, L_0L_3, or L_1L_3. The second RR dyad constrain can be added to either connect a link to ground as listed previously or to connect the new RR dyad link L_4 to the original RRR chain. For example, the set of RR dyads (L_0L_2, L_0L_3) is the Stephenson IIIa. The transformation matrices used for the RR dyad synthesis would be L_2 and L_3 for the first and second RR dyads, respectively. These are used because they already represent the transformation from the ground frame to frames L_2 and L_3. Other combinations would require the transformation between each of the frames to be calculated.

See Ref [78] for a detailed discussion on the synthesis.

Continuing from the Stephenson IIIa example, the transformation matrices for the RR dyads are $T_2 = L_2$ and $T_3 = L_3$ and the three joint trajectories are defined by $\mathbf{q_j} = (\theta_{1j}, \theta_{2j}, \theta_{3j}), j = 1, \ldots, 5$.

The relative transformations of the second and third frames are given by,

$$
\mathbf{R}_{1j} = \mathbf{T}_2(\mathbf{q}_j)\mathbf{T}_2(\mathbf{q}_1)^{-1} \text{ and } \mathbf{S}_{1j} = \mathbf{T}_3(\mathbf{q}_j)\mathbf{T}_3(\mathbf{q}_1)^{-1}, \tag{2.9}
$$

which yields,

$$\mathbf{B}^j = \mathbf{R}_{1j}\mathbf{B}^1 \text{ and } \mathbf{E}^j = \mathbf{S}_{1j}\mathbf{E}^1. \tag{2.10}$$

The RR dyad **BC** has coordinates that must satisfy the constraint equations,

$$(\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) = h^2, \quad j = 1, \ldots, 5. \tag{2.11}$$

where

$$\mathbf{B}^1 = (x, y, 0), \quad \mathbf{C} = (u, v, 0). \tag{2.12}$$

Similarly, the SS dyad **EF** has coordinates which must satisfy

$$(\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) = k^2, \quad j = 1, \ldots, 5, \tag{2.13}$$

where

$$\mathbf{E}^1 = (m, n, 0), \quad \mathbf{F} = (p, q, 0). \tag{2.14}$$

where the lengths of **BC** and **EF** are h and k, respectively. Both sets of equations (2.11) and (2.13) can be simplified by subtracting the first equation in the set from the remaining five equations. This removes the squared terms for all 8 coordinates of **BC** and **EF** and the constants h^2 and k^2. The result is the two sets of design equations

$$\begin{aligned} \mathcal{A}_j : (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) - (\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{B}^1 - \mathbf{C}) = 0, \\ j = 2, \ldots, 5, \end{aligned} \tag{2.15}$$

and

$$\mathcal{B}_j : (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) - (\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{E}^1 - \mathbf{F}) = 0,$$
$$j = 2, \ldots, 5, \tag{2.16}$$

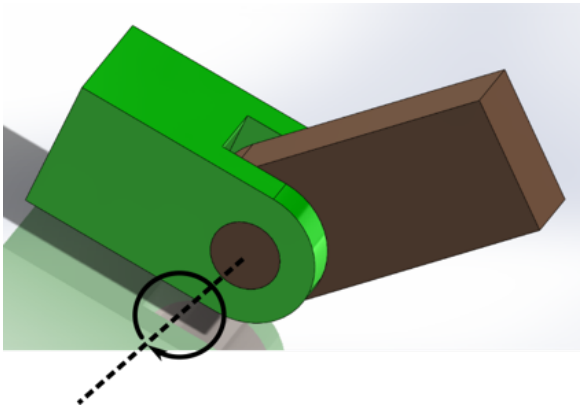
The design equations are then solved which produce the coordinates of the RR dyads in the initial position of the linkage.

2.3 Spatial Mechanism Synthesis

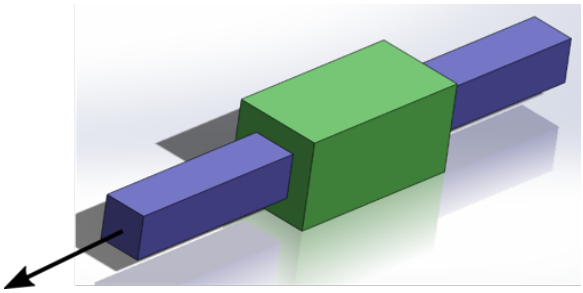
Spatial mechanisms are characterized by at least one link moving between two general positions in space, which must be specified using three coordinates for position and three rotations for orientation. [54] Spatial mechanisms have no restrictions on the relative movements of link. Planar and spherical mechanisms are actually subsets of spatial mechanisms. [88] Additionally, spatial mechanisms have a greater variety of joints that can be used, Figure 2.5 shows a few examples.

2.3.1 SS Dyad Synthesis

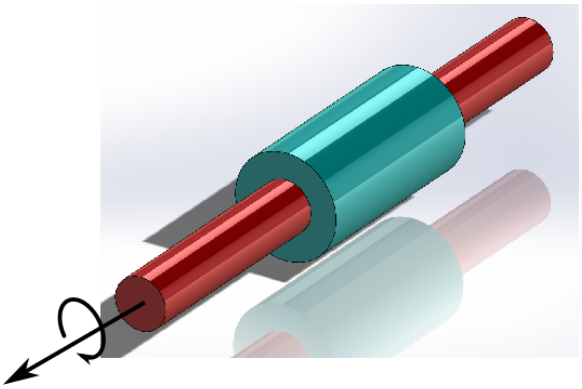
An SS Dyad is composed of two spherical joints connected by a rigid link. To synthesize an ss dyad, first a ground frame G_1 and a moving frame M_1 are defined, where the SS dyad will connect the ground frame to the moving frame. The coordinates of the spherical joints in the first position are given by $\mathbf{A} = (A_x, A_y, A_z)^T$ and $\mathbf{B} = (B_x, B_y, B_z)^T$. Seven positions of the moving frame M_1 are given by the transformation matrix $[T_i], i = 1, \ldots, 7$.



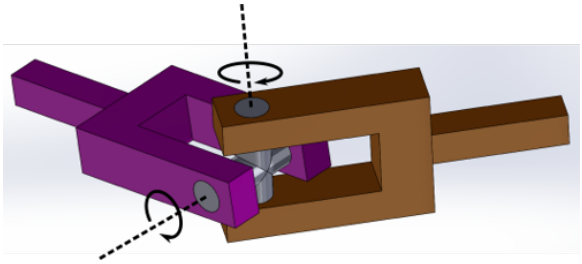
(a) Revolute Joint (1 DOF)



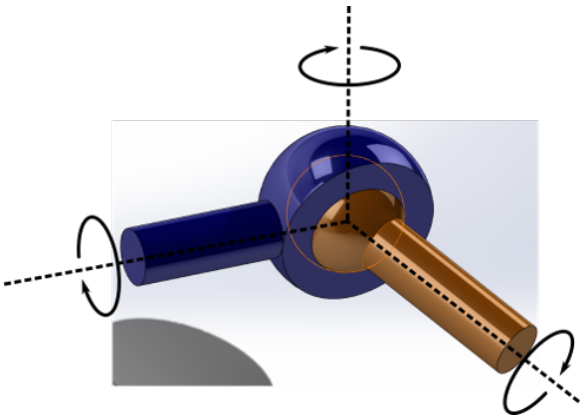
(b) Prismatic Joint (1 DOF)



(c) Cylindrical Joint (2 DOF)



(d) Universal Joint (2 DOF)



(e) Spherical Joint (3 DOF)

Figure 2.5: Examples of Spatial Joints

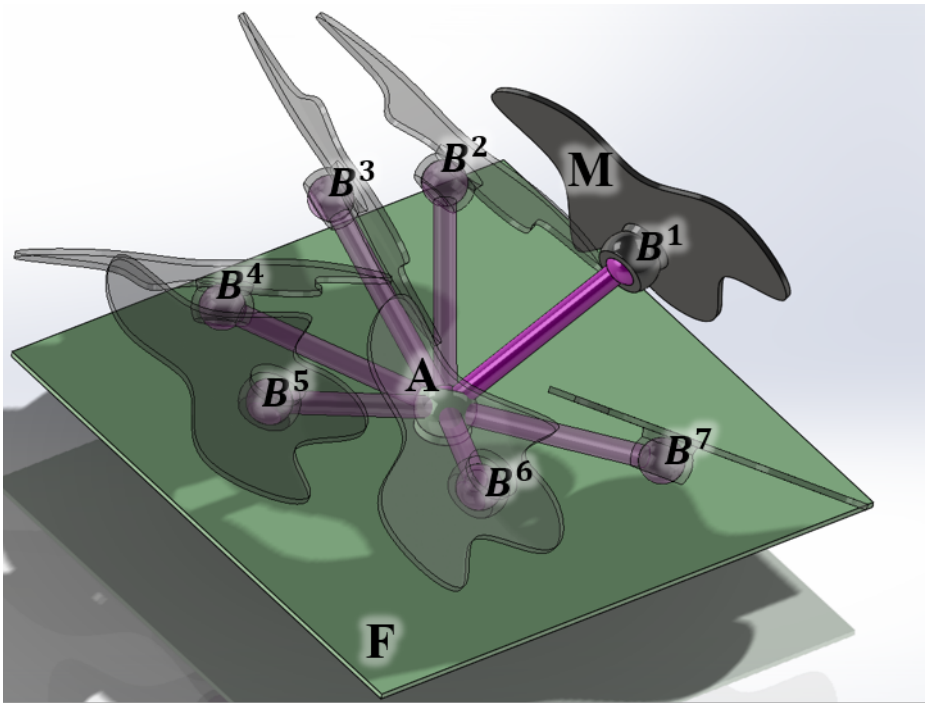


Figure 2.6: The SS Dyad shown has a fixed pivot $\mathbf{A} = (A_x, A_y, A_z)^T$ and a moving pivot $\mathbf{B} = (B_x, B_y, B_z)^T$ in fixed frame $\mathbf{F}$.

The coordinates of $\mathbf{B}^i, i = 1, \ldots, 7$ can be found in terms of

$$\mathbf{B}^i = [T_i][T_1^{-1}]\mathbf{B}^1, i = 1, \ldots, 7 \tag{2.17}$$

Now, knowing that $\mathbf{AB}$ is a fixed length the following constraint equations are formed,

$$(\mathbf{B}^i - \mathbf{A}) \cdot (\mathbf{B}^i - \mathbf{A}) = R^2, i = 1, \ldots, 7 \tag{2.18}$$

.

The equations can be simplified by subtracting the first equation in the set from the remaining

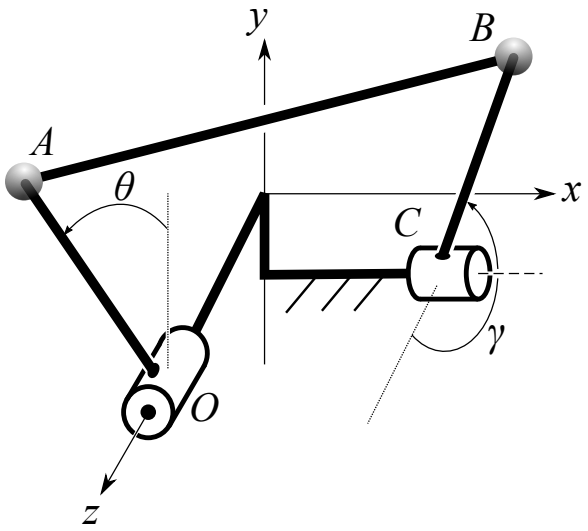


Figure 2.7: The RSSR linkage is shown with the two revolute joints connected to the ground link **OC** with **AB** and **BC** moving.

six equations. This cancels the constant R^2 and the squared terms for the six coordinates of **AB**. The result is a set of design equations

$$(\mathbf{B}^i - \mathbf{A}) \cdot (\mathbf{B}^i - \mathbf{A}) - (\mathbf{B}^1 - \mathbf{A}) \cdot (\mathbf{B}^1 - \mathbf{A}) = 0, i = 1, \ldots, 7 \tag{2.19}$$

The six equations are solved simultaneously to give a maximum of 20 pairs of coordinates, $\mathbf{B}^1$ and $\mathbf{A}$. The six equations can be put in the form a single univariate polynomial of the 20th order.

2.3.2 RSSR Function Generator Synthesis

The RSSR function generator produces a given output angle γ for a given input angle θ, seen in Figure 2.7. The input link and the output link are connected by SS-Dyad **AB**. To synthesize the RSSR function generator seven pairs of input and output angles must be selected.

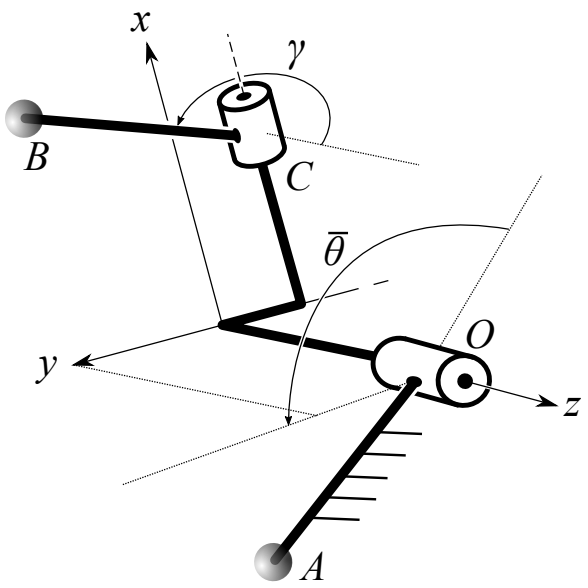


Figure 2.8: The RSSR linkage shown has link **OA** connected to ground with links **OC** and **BC** moving.

The linkage is then inverted as shown in Figure 2.8. Frame **OA** is fixed to ground. While frame **OC** moves by $\bar{\theta} = \pi - \theta$ and Frame **BC** continues to move by γ.

Now joint **A** is in the ground frame and joint **B** is in the moving frame. This is the configuration needed for SS-Dyad synthesis. The transformations from joint **A** to joint **B** are given by

$$T(\theta^i, \gamma^i) = Z(\theta^i, d_1)X(\alpha, a)Z(\gamma^i, d_2), i = 1, \ldots, 7. \tag{2.20}$$

The SS Dyad synthesis can now be solved by substituting the transformation into equations 2.17 and 2.19. Up to 20 sets of initial coordinates for joint **A** to joint **B** can be found and used to assemble the RSSR linkage.

2.4 Forming Spatial Loops

Spatial loops are formed by connecting links, which may be dyads, triads, etc..., together to form a closed loop. Chen and Roth [14, 15] show the number of points that satisfy a given link constraint for a specified number of design positions for both dyads and single loop linkages. The points that satisfy the single loop equations are shown to be the intersection of two surfaces associated with the links. For example the RPSS linkage is created by combining the RPS and SS Dyads togther. The Locus of points is the intersection of a hyperboloid and sphere. This combination has a maximum of five specified positions and gives a 20th order space curve as that satisfies the linkage constraint, meaning that there is an infinite number of points that satisfies the linkage constraint. So an additional constraining plane can be added to obtain finite solutions.

2.5 Spatial Six Bar Synthesis

The synthesis procedures of the spatial six-bar mechanism is the foundation of the research presented in this dissertation. A systematic method to synthesizing a spatial six-bar mechanism is provided that can be applied to a variety of different mechanisms. The synthesis begins with the specification of a spatial three link serial chain. The spatial three link chain is defined in terms of the link lengths and joint positions at each of the specified precision points. Each precision point consists of the joint angles or slides at each position of the three link serial chain. After the serial chain and the desired precision points are defined then SS dyads can be found that will constrain the second and third joints to move through the desired precision points. The constrained linkage is a single degree of freedom mechanism.

2.5.1 Three Jointed Spatial Serial Chain

For serial chains with three joints, let $L_i, i = 1, 2, 3$ be the coordinate frames of each link with the z-axis along the ith joint and x-axis along the common normal to the next joint axis. The position of each link relative to the ground frame is defined by the kinematics equations,

$$\begin{aligned}
\mathbf{T}_1 =& \mathbf{Z}(\theta_1, d_1), \\
\mathbf{T}_2 =& \mathbf{Z}(\theta_1, d_1)\mathbf{X}(\alpha_1, a_1)\mathbf{Z}(\theta_2, d_2), \\
\mathbf{T}_3 =& \mathbf{Z}(\theta_1, d_1)\mathbf{X}(\alpha_1, a_1)\mathbf{Z}(\theta_2, d_2)\mathbf{X}(\alpha_2, a_2)\mathbf{Z}(\theta_3, d_3).
\end{aligned} \tag{2.21}$$

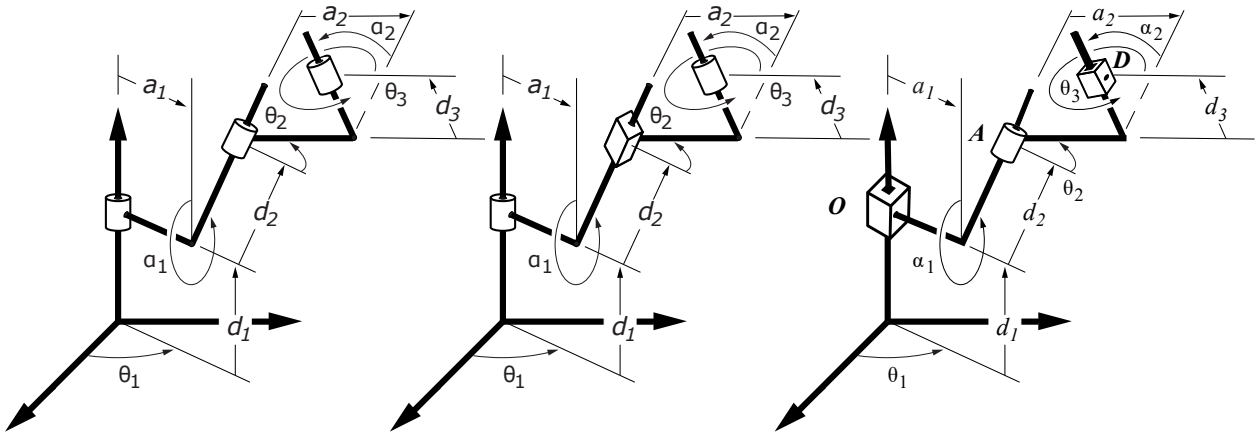
The joint parameters θ and d and the link parameters α and a are found the DH Table, Table 2.2. The RRR, RPR and PRP serial chains are shown in Figure 2.9 and their respective variable parameters and constants are shown.

Table 2.2: Denavit-Hartenberg table for the a serial chain.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	d_1	α_1	a_1
2	θ_2	d_2	α_2	a_2
3	θ_3	d_3	-	-

2.5.2 Task Requirements

The task requirements are sets of joint angles and slides in each position of the three jointed serial chain. Seven sets of joint angles need to be specified when constraining an RRR serial chain and six sets of joint angles are needed to contraining an RPR and PRP serial chains. The RRR task requirements are seven sets of (θ_1,θ_2,θ_3), Figure 2.9a. The RPR task requirements are six sets of (θ_1,d_2,θ_3), Figure 2.9b. The RRR task requirements are six sets of (d_1,θ_2,d_3), Figure 2.9c.



(a) RRR Serial Chain
Variables: θ_1,θ_2,θ_3
Constants:
d_1,d_2,d_3,α_1,α_2,a_1,a_2

(b) RPR Serial Chain
Variables: θ_1,d_2,θ_3
Constants:
d_1,θ_2,d_3,α_1,α_2,a_1,a_2

(c) PRP Serial Chain
Variables: d_1,θ_2,d_3
Constants:
θ_1,d_2,θ_3,α_1,α_2,a_1,a_2

Figure 2.9: The RRR, RPR, and PRP serial chains and their respective variable parameters and constants.

The joint angles and slides can either be directly specified or calculated from a set of end effector positions. To calculate the joint angles and slides from the end effector position, the end effector must be specified in the form of $\mathbf{T}_{ef} = [A, \mathbf{d}]$ where A is a rotation matrix and d is the coordinate vector of the end effector. The end effector position $\mathbf{T}_{ef}$ is set equal to $\mathbf{T}_3$. Each serial chain will have a different end-effector transformation (2.21) depending on the joint parameters are variable, that is

$$\mathbf{E}_{RRR} = \mathbf{T}_3(\theta_1, \theta_2, \theta_3), \quad \mathbf{E}_{RPR} = \mathbf{T}_3(\theta_1, d_2, \theta_3), \quad \mathbf{E}_{PRP} = \mathbf{T}_3(d_1, \theta_2, d_3). \tag{2.22}$$

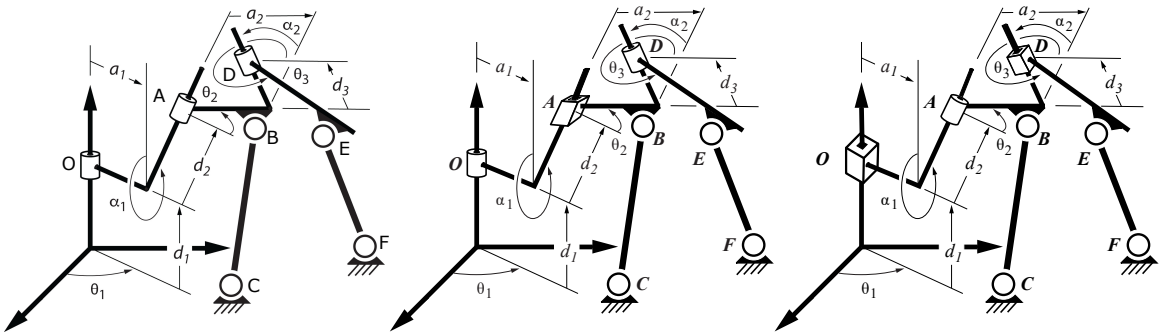
Given the end-effector location, the kinematics equations are solved to determine the joint parameters of the specified serial chain. This process is repeated for each of the task positions.

The task requirements are for the three different serial chains are:

1. RRR Serial Chain: $(\theta_{1i},\theta_{2i},\theta_{3i})$, $i = 1, \ldots, 7$;

2. RPR Serial Chain: $(\theta_{1i},d_{2i},\theta_{3i})$, $i = 1, \ldots, 6$;

3. PRP Serial Chain: $(d_{1i},\theta_{2i},d_{3i})$, $i = 1, \ldots, 6$.



(a) RRR-2SS Spatial Six-Bar (b) RPR-2SS Spatial Six-Bar (c) PRP-2SS Spatial Six-Bar

Figure 2.10: The RRR-2SS, RPR-2SS, and PRP-2SS Spatial Six-Bar Mechanisms.

2.5.3 Synthesis Equations

The spatial six-bar mechanisms are shown in Figure 2.10. This figure shows the three serial chains from Figure 2.9 each constrained by two SS Dyads.

In order to constrain the three link spatial chain to one degree-of-freedom, we introduce an SS dyad **BC** that connects L_2 to the ground frame F, and another SS dyad **EF** that connects L_3 to the ground frame. Let $\mathbf{B}^j$ denote the coordinates of the moving pivot attached to L_2 measured in F, when the three link chain is in the configuration defined by $\mathbf{q}_j$. Similarly, let $\mathbf{E}^j$ be the coordinates of the moving pivot attached to L_3 for each of the precision positions $\mathbf{q}_j$.

Define the coordinates of the spherical joints as,

$$\mathbf{B}^1 = (x, y, z), \quad \mathbf{C} = (u, v, w), \quad \mathbf{E}^1 = (m, n, o), \quad \mathbf{F} = (p, q, r). \tag{2.23}$$

where B^1 and E^1 are the initial positions of the moving pivots.

Introduce the relative displacements that give the displacement relative to the initial position

of the mechanism,

$$\mathbf{R}_{1j} = \mathbf{T}_2(\mathbf{q}_j)\mathbf{T}_2(\mathbf{q}_1)^{-1} \text{ and } \mathbf{S}_{1j} = \mathbf{T}_3(\mathbf{q}_j)\mathbf{T}_3(\mathbf{q}_1)^{-1}, \tag{2.24}$$

so we have the positions of joints **B** and **E** defined in terms of their respective initial positions $\mathbf{B}^1$ and $\mathbf{E}^1$,

$$\mathbf{B}^j = \mathbf{R}_{1j}\mathbf{B}^1 \text{ and } \mathbf{E}^j = \mathbf{S}_{1j}\mathbf{E}^1. \tag{2.25}$$

The coordinates for the SS dyad **BC** must satisfy the constraint equations,

$$(\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) = h^2, \quad j = 1, \ldots, 7. \tag{2.26}$$

Similarly, the coordinates for the SS dyad **EF** must satisfy

$$(\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) = k^2, \quad j = 1, \ldots, 7, \tag{2.27}$$

where h and k are the lengths of **BC** and **EF**, respectively.

Both equations (2.26) and (2.27) can be simplified by subtracting the first equation in the set from the remaining six equations. This cancels the constants h^2 and k^2 as well as the squared terms for all 12 coordinates of **BC** and **EF**. The result is the two sets of design equations

$$\begin{aligned} \mathcal{A}_j : (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) - (\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{B}^1 - \mathbf{C}) = 0, \\ j = 2, \ldots, 7, \end{aligned} \tag{2.28}$$

and

$$\mathcal{B}_j : (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) - (\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{E}^1 - \mathbf{F}) = 0,$$
$$j = 2, \ldots, 7, \tag{2.29}$$

The equations $\mathcal{A}_j$ and $\mathcal{B}_j$ are each bilinear in their respective six unknown coordinates for **BC** and **EF**. They can be solved independently to determine as many as 20 SS dyads, which is as many as 400 pairs of SS dyads that guide the three link serial chain through the seven precision points, $\mathbf{q}_j, j = 1, \ldots, 7$, [44, 54].

2.5.4 Performance Evaluation

The spatial six-bar mechanism consists of two loops: i) one formed by **OABC** that forms a spatial four-bar closed chain, ii) the other formed by **OADEF** is a spatial five-bar closed chain. Once the two SS dyads are determined, then the coordinates of $\mathbf{B}^1\mathbf{C}$ and $\mathbf{E}^1\mathbf{F}$ are known, as are the lengths $h = |\mathbf{B}^1\mathbf{C}|$ and $k = |\mathbf{E}^1\mathbf{F}|$. The movement of the system is determined by specifying the input angle θ_1 or slide d_1 and solving the loop constraint equations for the two outputs, θ_2 or d_2 and θ_3 or d_3.

2.5.4.1 RRR-2SS Analysis Equations

The constraint equation for the loop **OABC** is given by

$$(\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) = h^2, \tag{2.30}$$

where

$$\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2) = \mathbf{T}_2(\theta_1, \theta_2)\mathbf{T}_2(\theta_{11}, \theta_{21})^{-1}, \tag{2.31}$$

and $\Delta\theta_1 = \theta_1 - \theta_{11}$ and $\Delta\theta_2 = \theta_2 - \theta_{21}$.

Equation 2.31 should then be expanded and solved for θ_2 in terms of θ_1.

The constraint equation for the loop **OADEF** is given by

$$(\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) = k^2, \tag{2.32}$$

where

$$\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3) = \mathbf{T}_3(\theta_1, \theta_2, \theta_3)\mathbf{T}_3(\theta_{11}, \theta_{21}, \theta_{31})^{-1}, \tag{2.33}$$

and $\Delta\theta_3 = \theta_3 - \theta_{31}$. Expand equation 2.33 and solve for θ_3 in terms of θ_1 and θ_2.

The expanded form of the equations take the following form,

$$A(\Delta\theta_1)\cos\Delta\theta_2 + B(\Delta\theta_1)\sin\Delta\theta_2 = C(\Delta\theta_1), \tag{2.34}$$

and

$$D(\Delta\theta_1, \Delta\theta_2)\cos\Delta\theta_3 + E(\Delta\theta_1, \Delta\theta_2)\sin\Delta\theta_3 = F(\Delta\theta_1, \Delta\theta_2), \tag{2.35}$$

where the solutions are,

$$\Delta\theta_2 = \arctan\frac{B}{A} \pm \arccos\frac{C}{\sqrt{A^2 + B^2}}, \tag{2.36}$$

and

$$\Delta\theta_3 = \arctan\frac{E}{D} \pm \arccos\frac{F}{\sqrt{D^2 + E^2}}. \tag{2.37}$$

For each value of the input joint, the solution to the analysis equations yields two values for

θ_2, which we denote as θ_2^+ and θ_2^-. Similarly, the second equation yields two values θ_3^+ and θ_3^-.

The result is four joint trajectories,

$$\begin{aligned}\mathbf{q}^1 =&(\theta_1, \theta_2^+, \theta_3^+), \mathbf{q}^2 = (\theta_1, \theta_2^+, \theta_3^-),\\ \mathbf{q}^3 =&(\theta_1, \theta_2^-, \theta_3^+), \text{ and } \mathbf{q}^4 = (\theta_1, \theta_2^-, \theta_3^-).\end{aligned} \tag{2.38}$$

We compare these four trajectories to the required task to determine if the mechanism moves smoothly through the required configurations. If the task positions do not lie on the same trajectory the mechanism design is discarded.

2.5.4.2 RPR-2SS Analysis Equations

The constraint equation for the loop **OABC** is given by

$$(\mathbf{R}_{1q}(\Delta\theta_1, \Delta d_2)\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1q}(\Delta\theta_1, \Delta d_2)\mathbf{B}^1 - \mathbf{C}) = h^2, \tag{2.39}$$

where

$$\mathbf{R}_{1q}(\Delta\theta_1, \Delta d_2) = \mathbf{T}_2(\theta_1, d_2)\mathbf{T}_2(\theta_{11}, d_{21})^{-1}, \tag{2.40}$$

and $\Delta\theta_1 = \theta_1 - \theta_{11}$ and $\Delta\theta_2 = \theta_2 - \theta_{21}$.

Equation 2.40 should then be expanded and solved for d_2 in terms of θ_1.

The constraint equation for the loop **OADEF** is given by

$$(\mathbf{S}_{1q}(\Delta\theta_1, \Delta d_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1q}(\Delta\theta_1, \Delta d_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) = k^2, \tag{2.41}$$

where

$$\mathbf{S}_{1q}(\Delta\theta_1, \Delta d_2, \Delta\theta_3) = \mathbf{T}_3(\theta_1, d_2, \theta_3)\mathbf{T}_3(\theta_{11}, d_{21}, \theta_{31})^{-1}, \tag{2.42}$$

and $\Delta\theta_3 = \theta_3 - \theta_{31}$. Expand equation 2.42 and solve for θ_3 in terms of θ_1 and d_2.

The expanded form of the equations respectively take the following form,

$$A(\Delta\theta_1)\Delta d_2^2 + B(\Delta\theta_1)\Delta d_2 + C(\Delta\theta_1) = 0, \tag{2.43}$$

and

$$D(\Delta\theta_1, \Delta\theta_2)\cos\Delta\theta_3 + E(\Delta\theta_1, \Delta\theta_2)\sin\Delta\theta_3 = F(\Delta\theta_1, \Delta\theta_2), \tag{2.44}$$

where the solutions are,

$$\Delta d_2 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}, \tag{2.45}$$

and

$$\Delta\theta_3 = \arctan\frac{E}{D} \pm \arccos\frac{F}{\sqrt{D^2 + E^2}}. \tag{2.46}$$

For each value of the input joint, the solution to the analysis equations yields two values for d_2, which we denote as d_2^+ and d_2^-. Similarly, the second equation yields two values θ_3^+ and θ_3^-.

The result is four joint trajectories,

$$\begin{aligned}
&\mathbf{q}^1 = (\theta_1, d_2^+, \theta_3^+), \mathbf{q}^2 = (\theta_1, d_2^+, \theta_3^-), \\
&\mathbf{q}^3 = (\theta_1, d_2^-, \theta_3^+), \text{ and } \mathbf{q}^4 = (\theta_1, d_2^-, \theta_3^-).
\end{aligned} \tag{2.47}$$

We compare these four trajectories to the required task to determine if the mechanism moves smoothly through the required configurations. If the task positions do not lie on the same trajectory the mechanism design is discarded.

2.5.4.3 PRP-2SS Analysis Equations

The constraint equation for the loop **OABC** is given by

$$(\mathbf{R}_{1q}(\Delta d_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1q}(\Delta d_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) = h^2, \tag{2.48}$$

where

$$\mathbf{R}_{1q}(\Delta d_1, \Delta\theta_2) = \mathbf{T}_2(d_1, \theta_2)\mathbf{T}_2(d_{11}, \theta_{21})^{-1}, \tag{2.49}$$

and $\Delta d_1 = d_1 - d_{11}$ and $\Delta\theta_2 = \theta_2 - \theta_{21}$.

Equation 2.49 should then be expanded and solved for θ_2 in terms of d_1.

The constraint equation for the loop **OADEF** is given by

$$(\mathbf{S}_{1q}(\Delta d_1, \Delta\theta_2, \Delta d_3)\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1q}(\Delta d_1, \Delta\theta_2, \Delta d_3)\mathbf{E}^1 - \mathbf{F}) = k^2, \tag{2.50}$$

where

$$\mathbf{S}_{1q}(\Delta d_1, \Delta\theta_2, \Delta d_3) = \mathbf{T}_3(d_1, \theta_2, d_3)\mathbf{T}_3(d_{11}, \theta_{21}, d_{31})^{-1}, \tag{2.51}$$

and $\Delta d_3 = d_3 - d_{31}$. Expand equation 2.51 and solve for d_3 in terms of d_1 and θ_2.

The expanded form of the equations respectively take the following form,

$$A(\Delta\theta_1)\cos\Delta\theta_2 + B(\Delta\theta_1)\sin\Delta\theta_2 = C(\Delta\theta_1), \tag{2.52}$$

and

$$D(\Delta d_1, \Delta\theta_2)\Delta d_3^2 + E(\Delta d_1, \Delta\theta_2)\Delta d_3 + F(\Delta d_1, \Delta\theta_2) = 0 \tag{2.53}$$

Where the solutions are,

$$\Delta\theta_2 = \arctan\frac{B}{A} \pm \arccos\frac{C}{\sqrt{A^2+B^2}}, \tag{2.54}$$

and

$$\Delta d_3 = \frac{-E \pm \sqrt{E^2 - 4DF}}{2D}. \tag{2.55}$$

For each value of the input joint, the solution to the analysis equations yields two values for θ_2, which we denote as θ_2^+ and θ_2^-. Similarly, the second equation yields two values d_3^+ and d_3^-.

The result is four joint trajectories,

$$\begin{aligned}
&\mathbf{q}^1 = (d_1, \theta_2^+, d_3^+), \mathbf{q}^2 = (d_1, \theta_2^+, d_3^-), \\
&\mathbf{q}^3 = (d_1, \theta_2^-, d_3^+), \text{ and } \mathbf{q}^4 = (d_1, \theta_2^-, d_3^-).
\end{aligned} \tag{2.56}$$

We compare these four trajectories to the required task to determine if the mechanism moves smoothly through the required configurations. If the task positions do not lie on the same trajectory the mechanism design is discarded.

2.6 Summary

The mathematical foundation laid by this chapter provides the necessary tools to synthesize spatial six-bar linkages through constraining a spatial serial chain. These tools can be used to synthesize linkages beyond the specific configurations described in this dissertation.

Chapter 3

Kinematic Synthesis

This chapter covers the kinematic synthesis procedures for the RRR-RR-SS, RRR-2SS, RPR-2SS and PRP-2SS mechanisms following from the spatial six bar linkage procedure described previously. The RRR-RR-SS follows a different synthesis procedure since the first link of the RRR spatial chain is constrained by a planar RR dyad that was calculated using a two position synthesis method. The RRR-2SS, RPR-2SS and the PRP-2SS follow the spatial six-bar synthesis procedure. First the serial chain is defined, and the unique aspects of solving the constraints for each linkage are described. Each section gives an example input function as well as the resulting design equations. The forward kinematic equations for each linkage are derived for use in the analysis routine to check for branching.

The term task point or task position will be used to describe the set three joint positions for each position of the spatial three link chain.

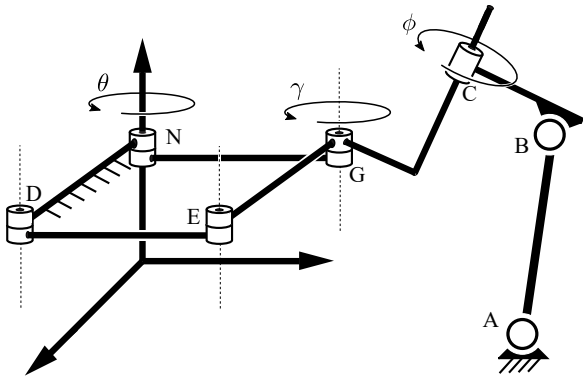


Figure 3.1: The diagram shows the structure of the six-bar mechanism

3.1 RRR-RR-SS Mechanism

The mechanism combines a planar four-bar linkage with a spatial four-bar linkage attached to the input and output links forming a six-bar linkage. The result is a six-bar mechanism, Figure 3.1, that uses a single input to coordinate two outputs. This mechanism was inspired by the need to control the swing and pitch of a wing with a single motor. The planar four-bar linkage was designed to control the wing swing trajectory profile and the spatial four-bar linkage was designed to coordinate the pitch of the wing to the swing movement.

3.1.1 Example Input Function

Yan and Taha [94] give the wing swing and wing pitch functions, $q(\theta)$ and $\phi(\theta)$,

$$q(\theta) = \frac{\pi}{2} - \frac{\pi}{3}\cos\theta, \quad \phi(\theta) = \frac{\pi}{2} - \frac{\pi}{3}\sin\theta, \tag{3.1}$$

where the driving angle is $\theta = \omega t$, and ω is the flapping frequency, Figure 3.2.

These functions are used to demonstrate the synthesis technique for the planar spatial six bar mechanism.

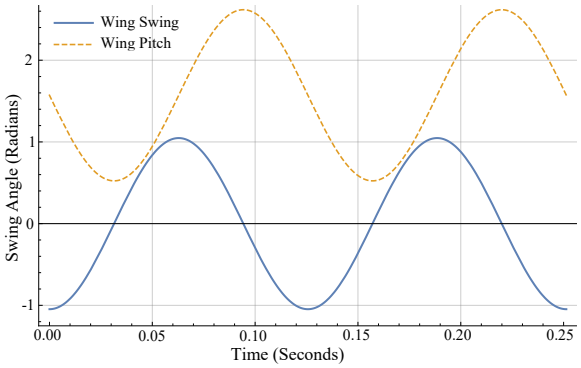


Figure 3.2: The wing swing and wing pitch functions.

3.1.2 Synthesis of the Planar Mechanism for Wing Swing

In order to design the wing swing mechanism, we follow Brodell and Soni [8] to obtain a four-bar linkage that has a time ratio of one. This ensures the speed of the flapping movement is the same in both the forward and back strokes.

Brodell and Soni provide formulas for the link lengths parameterized by the desired wing swing angle σ and transmission angle λ,

$$\frac{r_3}{r_1} = \sqrt{\frac{1-\cos\sigma}{2\cos^2\lambda}}, \quad \frac{r_4}{r_1} = \sqrt{\frac{1-(r_3/r_1)^2}{1-(r_3/r_1)^2\cos^2\lambda}},$$
$$\frac{r_2}{r_1} = \sqrt{\left(\frac{r_3}{r_1}\right)^2 + \left(\frac{r_4}{r_1}\right)^2 - 1}, \tag{3.2}$$

where r_1, r_2, r_3, and r_4 are the lengths of the ground link, the input crank, the coupler link, and the follower link, respectively.

Given the link lengths of the crank-rocker, the wing swing function $\gamma(\Delta\theta)$ where $\Delta\theta$ is measured from the line **NG** of the linkage, McCarthy [54],

$$A(\Delta\theta)\cos\gamma + B(\Delta\theta)\sin\gamma = C(\Delta\theta), \tag{3.3}$$

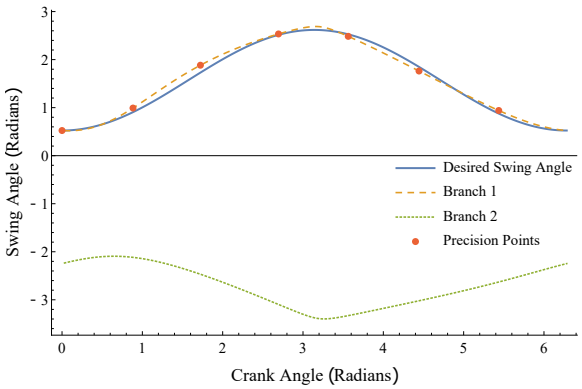


Figure 3.3: Comparison of the task swing function $q(\theta)$ and the output of the swing mechanism $\bar{\gamma}(\theta)$.

where

$$A(\Delta\theta) = 2r_2r_4\cos\Delta\theta + 2r_1r_4, \quad B(\Delta\theta) = 2r_2r_4\sin\Delta\theta,$$
$$C(\Delta\theta) = r_1^2 + r_2^2 + r_4^2 - r_3^2 + 2r_1r_2\cos\Delta\theta. \tag{3.4}$$

This has the solution,

$$\gamma(\Delta\theta) = \arctan\left(\frac{B}{A}\right) \pm \arccos\left(\frac{C}{\sqrt{A^2+B^2}}\right). \tag{3.5}$$

Our goal is to achieve the four-bar linkage output $\bar{\gamma}(\theta) = \gamma(\theta_0 + \Delta\theta) + k_0$ that matches the wing swing function $q(\theta)$ at $\theta = 0$ and where $q(0) = \pi/6$ and the minimum of γ is at $\gamma(0.045) = 0.568$. Thus, $0 = \theta_0 + 0.045$ and $\pi/6 = 0.568 + k_0$. The result is θ_0 = -0.045, or -2.6 degrees and k_0 = -0.045 or -2.6 degrees, as seen in Figure 3.3.

Thus, we obtain

$$\bar{\gamma} = \gamma(\Delta\theta - 0.045) - 0.045 \tag{3.6}$$

3.1.3 Synthesis of the Pitch Mechanism

In order to control the pitch of the flapping wing, we introduce an RSSR linkage connecting the input and output links of the swing mechanism that will orient the pitch of the wing during the flapping movement.

The planar wing swing mechanism **NDEG**, shown in Figure 3.4, is positioned in the ground frame W such that the fixed pivot **G** of the link **EGC** is on the z-axis of W. The axes of the four joints of the planar four-bar **NDEG** are also directed along the z-axis of W. The rotation γ of **EGC** provides the swing of the wing. The output crank **CB** of the RSSR linkage **NABC** controls the pitch ϕ of the wing around the axis of **C**. The output of the wing swing mechanism **NDEG** is an input to the wing pitch mechanism **NABC**. The dimension g is selected by the designer.

The location of the fixed pivot $\mathbf{N} = (0, t, 0)$ of input crank **DNA** is selected by the designer so that the ground link **GN** has the length r_1. The input rotation of **DNA** about the axis of **N** drives both the planar wing swing mechanism and spatial wing pitch mechanism.

Our goal is to determine the dimensions of the linkage by solving the SS constraint equations for seven values of the wing pitch function, $\phi_i = \phi(\theta_i)$, $i = 1, \ldots, 7$, see [54]. The values $\phi(\theta_i)$, which are chosen using Chebyshev spacing along the wing pitch function[41].

The synthesis equations coordinate the rotation of the link **DNA**, **EGC** and **BC** links so that they simultaneously satisfy, $\bar{\gamma}_i = \bar{\gamma}(\theta_i)$ and $\phi_i = \phi(\theta_i)$, $i = 1, \ldots, 7$.

The homogeneous transforms **Z** and **X** that define coordinate screw displacement about the

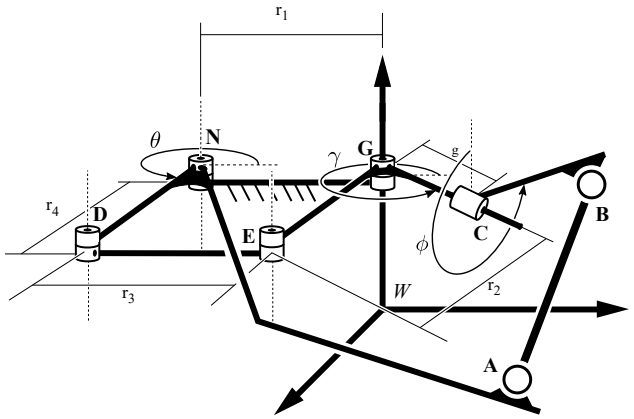


Figure 3.4: The diagram shows the RSSR and Planar Four-Bar mechanism and the frames of reference used to define the rotation axes.

x and z axes, given by,

$$\mathbf{Z}(\theta, d) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{X}(\alpha, a) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

are used to locate the axis of rotation of **N** for the input to the RSSR linkage and the axis of **C** for the output, Figure 3.4.

The rotation axis of **N** of the input crank **DNA** is defined by the sequence of transformations,

$$\mathbf{H}(\theta) = \mathbf{X}(0, t)\mathbf{Z}(\theta, 0), \tag{3.8}$$

where the parameter θ defines the rotation of the link **DNA** and $(0, t)$ define its position relative to the ground frame, see Figure 3.4. Let $\mathbf{H}_{i1}$ be the transformation relative to the initial configuration of the RSSR chain evaluated at seven task angles θ_i, $i = 1, \ldots, 7$, so we

have,

$$\mathbf{H}_{i1} = \mathbf{H}_i \mathbf{H}_1^{-1}, \quad i = 1, \ldots, 7. \tag{3.9}$$

The rotation axis of **C** of the output crank **CB** is defined by the sequence of transformations,

$$\mathbf{J}(\phi, \bar{\gamma}) = \mathbf{Z}(\bar{\gamma}, 0)\mathbf{X}(\pi/2, 0)\mathbf{Z}(\phi, g), \tag{3.10}$$

where $\bar{\gamma}$ and g defines the position relative to the ground frame, and ϕ defines the rotation of the output crank. Let $\mathbf{J}_{i1}$ denote the transformation relative to the initial configuration of the RSSR chain evaluated at seven task angles ϕ_i, $i = 1, \ldots, 7$,

$$\mathbf{J}_{i1} = \mathbf{J}_i \cdot \mathbf{J}_1^{-1} \quad i = 1, \ldots, 7. \tag{3.11}$$

The design equations for wing pitch mechanism are obtained as the constraint that the length of the SS link be constant in each of the task positions specified by $\phi_i = \phi(\theta_i)$, $i = 1, \ldots, 7$. Let the coordinates of the centers **A** and **B** of the S-joints in the first task position be given by

$$\mathbf{A}^1 = (u, v, w), \quad \mathbf{B}^1 = (x, y, z). \tag{3.12}$$

Then the constraint that the coupler link **AB** to be of constant length $b = |\mathbf{AB}|$ in each of the task positions yields the equations,

$$|\mathbf{A}^i - \mathbf{B}^i|^2 = |([\mathbf{H}_{i1}]\mathbf{A}^1 - [\mathbf{J}_{i1}]\mathbf{B}^1)|^2 = b^2, \quad i = 1, \ldots, 7. \tag{3.13}$$

These equations can be reduced to a degree 20 polynomial in terms of one of the following u, v, w, x, y or z, [44, 54], and thus the system of equations has a maximum of 20 unique solutions. This set of equations was solved using Mathematica which yielded values for the points $\mathbf{A}^1 = (u, v, w)$ and $\mathbf{B}^1 = (x, y, z)$. The solutions found are all possible solutions for the set of specified dimensions and angles.

3.1.4 Analysis of the Wing Pitch Mechanism

In order to evaluate the linkage obtained from the synthesis routine, we analyze RSSR Wing pitch mechanism at each input crank position. The goal is to verify that the mechanism moves smoothly through the specified wing swing and wing pitch movements. Every solution from the synthesis is analyzed.

The constraint equation of the RSSR chain defines the length of the link **AB** in terms of $\Delta\theta$, $\bar{\gamma}$ and ϕ given by,

$$|[\mathbf{H}(\Delta\theta)]\mathbf{a} - [\mathbf{J}(\phi, \bar{\gamma})]\mathbf{b}|^2 = b^2, \tag{3.14}$$

where $\mathbf{H}$ and $\mathbf{J}$ are given in (3.8) and (3.10) and $\mathbf{a}$ and $\mathbf{b}$ are the coordinates of $\mathbf{A}^1$ and $\mathbf{B}^1$ in the frame of the first task position given by,

$$\mathbf{a} = [\mathbf{H}_1]^{-1}\mathbf{A}^1 = (\bar{u}, \bar{v}, \bar{w}), \quad \mathbf{b} = [\mathbf{J}_1]^{-1}\mathbf{B}^1 = (\bar{x}, \bar{y}, \bar{z}). \tag{3.15}$$

For each value of $\Delta\theta$, we obtain $\bar{\gamma}$, and obtain an equation of the form,

$$A(\Delta\theta)\cos\phi + B(\Delta\theta)\sin\phi = C(\Delta\theta), \tag{3.16}$$

where

$$
\begin{aligned}
A(\Delta\theta) = &- 2t\bar{x}\cos(\gamma) - 2\bar{u}\bar{x}\cos(\gamma - \Delta\theta) \\
&- 2\bar{v}\bar{x}\sin(\gamma - \Delta\theta) - 2\bar{w}\bar{y} \\
B(\Delta\theta) = &+ 2t\bar{y}\cos(\gamma) + 2\bar{u}\bar{y}\cos(\gamma - \Delta\theta) \\
&+ 2\bar{v}\bar{y}\sin(\gamma - \Delta\theta) - 2\bar{w}\bar{x} \\
C(\Delta\theta) = &- b^2 + g^2 + t^2 + \bar{x}^2 + \bar{y}^2 + \bar{z}^2 + \bar{u}^2 + \bar{v}^2 + \bar{w}^2 \\
&+ 2g\bar{z} - 2gt\sin(\gamma) - 2g\bar{u}\sin(\gamma - \Delta\theta) \\
&+ 2g\bar{v}\cos(\gamma - \Delta\theta) + 2t\bar{u}\cos(\Delta\theta) - 2t\bar{v}\sin(\Delta\theta) \\
&- 2t\bar{z}\sin(\gamma) - 2\bar{u}\bar{z}\sin(\gamma - \Delta\theta) + 2\bar{v}\bar{z}\cos(\gamma - \Delta\theta)
\end{aligned}
\tag{3.17}
$$

where g and t are determined by the designer.

Solve for $\phi_i(\Delta\theta_i) \quad i = 1, ..., 7$ using Eq 3.5.

The solutions that pass through all task points on a single branch pass the branching analysis. The passing solutions are then checked for continuity. The range of the input crank angles is divided into two hundred intermediate angles and the links are checked to ensure that they do not change length at the intermediary positions. The solutions are plotted for the entire range of motion of the input crank. An example of a solution that fails in continuity, but passes branching is shown in Figure 3.5.

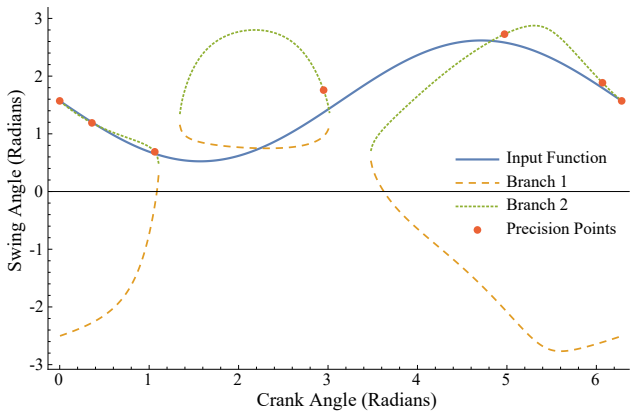


Figure 3.5: The plot of a non-continuous solution.

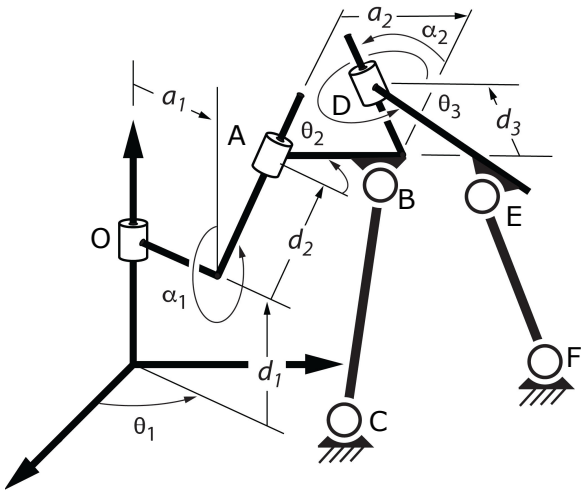


Figure 3.6: The spatial RRR-2SS linkage constructed by constraining a spatial RRR serial chain using two SS dyads that connect the second and third links to the ground frame.

3.2 RRR-2SS Mechanism

3.2.1 Motivation for Spatial Linkage

Creating a spatial RRR serial chain constrained by two SS dayds was motivated by packaging. The specific example which inspired this design was the need to move the location of the input crank from the previous RRRR-RR-SS linkage. The position of the input crank would require either a significantly more complex series of planer linkages or a spatial linkage. It was found that the usage of spatial linkage allowed for the placement of the input crank in any position that was tried.

3.2.2 Spatial Serial Chain

Table 3.1: Denavit-Hartenberg table for the RRR serial chain. $\theta_i, i = 1, 2, 3$ are joint variables. The remaining parameters are selected by the designer.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	d_1	α_1	a_1
2	θ_2	d_2	α_2	a_2
3	θ_3	d_3	–	–

3.2.2.1 The Spatial RRR Chain

The design of the RRR-2SS spatial six-bar linkage begins with the selection of a spatial RRR serial chain. The Denavit Hartenberg frames are shown in Figure 3.6 and the DH parameters are shown in Table 3.1, where $\theta_i, i = 1, 2, 3$ are joint variables and $d_i, i = 1, 2, 3$, $\alpha_i, i = 1, 2$ and $a_i, i = 1, 2$ are specified by the designer.

Let $L_i, i = 1, 2, 3$ be the coordinate frames with the z-axis along the ith joint and x-axis along the common normal to the next joint axis. The position of these links relative to the

ground frame are defined by the kinematics equations,

$$\begin{aligned}
\mathbf{T}_1 =& \mathbf{Z}(\theta_1, d_1), \\
\mathbf{T}_2 =& \mathbf{Z}(\theta_1, d_1)\mathbf{X}(\alpha_1, a_1)\mathbf{Z}(\theta_2, d_2), \\
\mathbf{T}_3 =& \mathbf{Z}(\theta_1, d_1)\mathbf{X}(\alpha_1, a_1)\mathbf{Z}(\theta_2, d_2)\mathbf{X}(\alpha_2, a_2)\mathbf{Z}(\theta_3, d_3),
\end{aligned} \tag{3.18}$$

where, $\mathbf{Z}(\theta_i, d_i)$ and $\mathbf{X}(\alpha_i, a_i)$ are the 4×4 homogeneous transforms,

$$\mathbf{Z}(\theta_i, d_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{X}(\alpha_i, a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad i = 1, 2, 3. \tag{3.19}$$

These kinematics equations are used in the design procedure.

3.2.3 SS Constraint Synthesis

Let the desired movement of the RRR serial chain be defined by the joint trajectories $\theta_1(t)$, $\theta_2(t)$ and $\theta_3(t)$, which are known functions of a parameter t. In order to design a one degree-of-freedom system that approximates this movement select seven task points from these joint trajectories, and denote them as $\mathbf{q}_i = (\theta_{1j}, \theta_{2j}, \theta_{3j})$, $j = 1, \ldots, 7$. Two SS constraints can be calculated that ensure the system reaches these seven task points.

In order to constrain the RRR spatial chain to one degree-of-freedom, we introduce an SS

dyad **BC** that connects L_2 to the ground frame F, and another SS dyad **EF** that connects L_3 to the ground frame.

Let $\mathbf{B}^j$ denote the coordinates of the moving pivot attached to L_2 measured in F, when the RRR chain is in the configuration defined by $\mathbf{q}_j$. Similarly, let $\mathbf{E}^j$ be the coordinates of the moving pivot attached to L_3 for each of the task positions $\mathbf{q}_j$. Introduce the relative displacements,

$$\mathbf{R}_{1j} = \mathbf{T}_2(\mathbf{q}_j)\mathbf{T}_2(\mathbf{q}_1)^{-1} \text{ and } \mathbf{S}_{1j} = \mathbf{T}_3(\mathbf{q}_j)\mathbf{T}_3(\mathbf{q}_1)^{-1}, \tag{3.20}$$

so we have

$$\mathbf{B}^j = \mathbf{R}_{1j}\mathbf{B}^1 \text{ and } \mathbf{E}^j = \mathbf{S}_{1j}\mathbf{E}^1. \tag{3.21}$$

The coordinates for the SS dyad **BC** must satisfy the constraint equations,

$$(\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) = h^2, \quad j = 1, \ldots, 7. \tag{3.22}$$

where

$$\mathbf{B}^1 = (x, y, z), \quad \mathbf{C} = (u, v, w). \tag{3.23}$$

Similarly, the coordinates for the SS dyad **EF** must satisfy

$$(\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) = k^2, \quad j = 1, \ldots, 7, \tag{3.24}$$

where

$$\mathbf{E}^1 = (m, n, o), \quad \mathbf{F} = (p, q, r). \tag{3.25}$$

where h and k are the lengths of **BC** and **EF**, respectively.

Both equations (3.22) and (3.24) can be simplified by subtracting the first equation in the set from the remaining six equations. This cancels the constants h^2 and k^2 as well as the squared terms for all 12 coordinates of **BC** and **EF**. The result is the two sets of design equations

$$\mathcal{A}_j : (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1j}\mathbf{B}^1 - \mathbf{C}) - (\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{B}^1 - \mathbf{C}) = 0, \quad j = 2, \ldots, 7, \tag{3.26}$$

and

$$\mathcal{B}_j : (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1j}\mathbf{E}^1 - \mathbf{F}) - (\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{E}^1 - \mathbf{F}) = 0, \quad j = 2, \ldots, 7, \tag{3.27}$$

The equations $\mathcal{A}_j$ and $\mathcal{B}_j$ are each bilinear in their respective six unknown coordinates for **BC** and **EF**. They can be solved independently to determine as many as 20 SS dyads, which is as many as 400 pairs of SS dyads that guide the RRR serial chain through the seven task points, $\mathbf{q}_j, j = 1, \ldots, 7$, [44, 54].

3.2.4 RRR-2SS Synthesis Example

Table 3.2: Denavit-Hartenberg table for the RRR serial chain of the Flapping Wing Mechanism.

Joint	θ_i	d_i	α_i	a_i
1	θ_1	7.07 cm	45°	-3.00 cm
2	θ_2	-5.00 cm	90°	-1.00 cm
3	θ_3	2.00 cm	–	–

The joint trajectories for the RRR chain that move this system as recommended by Yan and

Table 3.3: Seven task points (radians) selected from the joint trajectories of the RRR spatial chain to design the SS dyads

Point $\mathbf{q}_j$	θ_1	θ_2	θ_3
1	0.	0.52	1.57
2	0.88	0.99	0.81
3	1.72	1.88	0.56
4	2.69	2.53	1.04
5	3.56	2.48	2.03
6	4.44	1.76	2.58
7	5.44	0.94	2.32

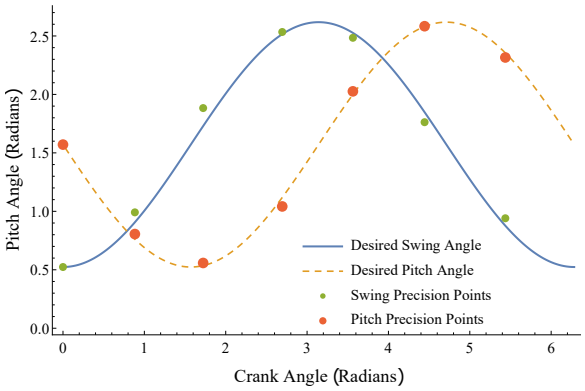


Figure 3.7: The task points selected for the synthesis of the swing and pitch linkages. The task points are shifted slightly from the required curves in the design process.

Taha [94] are giving by

$$\begin{aligned}\theta_1 =& t,\\ \theta_2 =& \frac{\pi}{2} - \frac{\pi}{3}\cos t,\\ \theta_3 =& \frac{\pi}{2} - \frac{\pi}{3}\sin t.\end{aligned} \tag{3.28}$$

as shown in Figure 3.2. Seven task positions selected from these trajectory curves are given in Table 3.3.

For this example the RRR spatial serial chain is given in Table 3.2 and the selected task points are given in Table 3.3.

Substitute the task points into the design equations $\mathcal{A}_j, j = 2, \ldots, 7$ for the link **BC**. The result is the following set of design equations,

$$
\begin{aligned}
\mathcal{A}_2: \quad & 1.36ux + 1.87uy + 0.32uz - 0.39u - 1.78vx + 1.29vy \\
& + 0.56vz - 4.68v - 0.64wx + 0.11wy + 0.11wz - 2.67w \\
& + 5.15x + 1.15y + 1.15z + 7.30 = 0 \\
\mathcal{A}_3: \quad & 3.43ux + 0.99uy - 0.99uz + 10.39u - 0.21vx + 3.55vy \\
& + 1.25vz - 3.5v - 1.38wx + 0.79wy + 0.79wz - 9.74w \\
& + 14.52x - 1.44y - 1.44z + 53.74 = 0 \\
\mathcal{A}_4: \quad & 1.79ux - 0.91uy - 1.77uz + 6.46u + 1.52vx + 3.07vy \\
& - 0.73vz + 12.34v - 1.28wx + 1.43wy + 1.43wz - 13.92w \\
& + 17.6x - 6.24y - 6.24z + 96.94 = 0 \\
\mathcal{A}_5: \quad & 0.77ux - 1.45uy - 0.63uz - 4.72u + 0.88vx + 2.03vy \\
& - 1.79vz + 12.88v - 1.31wx + 1.38wy + 1.38wz - 13.68w \\
& + 17.53x - 5.83y - 5.83z + 93.85 = 0 \\
\mathcal{A}_6: \quad & 0.88ux - 1.63uy + 0.3uz - 7.03u + 0.98vx + 1.06vy \\
& - 1.47vz + 7.54v - 1.34wx + 0.67wy + 0.67wz - 8.78w \\
& + 13.5x - 0.76y - 0.76z + 45.84 = 0 \\
\mathcal{A}_7: \quad & 0.36ux - 1.05uy + 0.44uz - 4.08u + 0.99vx + 0.3vy \\
& - 0.37vz + 1.11v - 0.57wx + 0.09wy + 0.09wz - 2.32w \\
& + 4.56x + 1.11y + 1.11z + 5.83 = 0. \qquad (3.29)
\end{aligned}
$$

The solution of these equations yield four real sets of coordinates for $\mathbf{B}^1 = (x, y, z)$ and $\mathbf{C} = (u, v, w)$, listed in Table 3.4.

Substitute the task points into the design equations $\mathcal{B}_i, j = 2, \ldots, 7$ for the link **EF**. The result is the following set of design equations,

$$
\begin{aligned}
\mathcal{B}_2 :& 1.96mp - 1.94mq + 0.5mr + 4.41m + 1.79np + 2.19nq \\
&+ 0.87nr - 0.66n + 0.89op - 0.46oq + 0.27or + 2.36o \\
&+ 0.38p - 4.87q - 1.23r + 6.36 = 0 \\
\mathcal{B}_3 :& -0.0072np + 3.95mp - 0.38mq + 0.2mr + 10.71m \\
&+ 2.92nq + 1.78nr - 8.61n + 0.43op + 1.73oq + 1.1or \\
&+ 2.62o + 11.05p - 3.72q - 7.72r + 48.91 = 0 \\
\mathcal{B}_4 :& 2.19mp + 1.91mq - 0.57mr + 16.58m - 0.4np + 2.59nq \\
&+ 1.87nr - 9.74n - 1.95op + 0.06oq + 1.57or - 3.58o \\
&+ 6.97p + 12.83q - 13.01r + 95.64 = 0 \\
\mathcal{B}_5 :& 1.15mp + 0.38mq - 1.77mr + 16.75m - 1.81np + 1.78nq \\
&+ 0.82nr - 3.4n + 0.04op - 1.95oq + 1.56or - 8.91o \\
&- 4.23p + 12.24q - 14.27r + 92.86 = 0 \\
\mathcal{B}_6 :& 2.47mp + 0.71mq - 1.81mr + 9.85m - 1.39np + 0.83nq \\
&- 0.82nr + 3.13n + 1.35op - 1.46oq + 1.78or - 7.62o \\
&- 5.02p + 7.19q - 9.38r + 41.21 = 0 \\
\mathcal{B}_7 :& 1.36mp + 1.51mq - 1.14mr + 3.98m - 1.23np + 0.75nq \\
&- 0.96nr + 2.09n + 1.44op - 0.4oq + 0.67or - 0.35o \\
&- 2.82p + 1.77q - 3.05r + 5.09 = 0.
\end{aligned}
\tag{3.30}
$$

The solution to these equations yield four sets of coordinates for $\mathbf{E} = (m, n, o)$ and $\mathbf{F} = (p, q, r)$ listed in Table 3.5.

Table 3.4: Real-valued solutions to design equations $\mathcal{A}_j$ for **BC**

	u	v	w	x	y	z
1	40.08	-50.05	298.50	-4.94	7.53	2.25
2	0.00.	0.00	234.83	-3.00	93.91	-86.83
3	-1.54	0.06	2.65	-4.47	0.53	5.99
4	0.00.	0.00	3.70	-3.00	9.29	-2.22

Table 3.5: Real-valued solutions to design equations $\mathcal{B}_j$ for **EF**

	p	q	r	m	n	o
1	2.80	-3.09	26.22	-4.94	4.76	4.15
2	-1.68	-0.48	3.65	-3.50	-41.90	-26.31
3	3.02	-1.06	-6.87	-2.05	3.39	2.53
4	-3.15	2.43	-10.00	-4.63	4.19	4.25

The two sets of four solutions to the design equations can be combined to define 16 candidate linkages that are analyzed to verify performance.

3.2.5 Analysis of Spatial Six-bar Mechanism

The Flapping Wing Mechanism consists of two loops: i) one formed by **OABC** that forms an RRSS closed chain, ii) the other formed by **OADEF** is an RRRSS closed chain. Once the two SS dyads are determined, then the coordinates of $\mathbf{B}^1\mathbf{C}$ and $\mathbf{E}^1\mathbf{F}$ are known, as are the lengths $h = |\mathbf{B}^1\mathbf{C}|$ and $k = |\mathbf{E}^1\mathbf{F}|$. The movement of the system is determined by specifying the input angle θ_1 and solving the loop constraint equations for θ_2 and θ_3.

The constraint equation for the loop **OABC** is given by

$$(\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) \cdot (\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2)\mathbf{B}^1 - \mathbf{C}) = h^2, \tag{3.31}$$

where

$$\mathbf{R}_{1q}(\Delta\theta_1, \Delta\theta_2) = \mathbf{T}_2(\theta_1, \theta_2)\mathbf{T}_2(\theta_{11}, \theta_{21})^{-1}, \tag{3.32}$$

and $\Delta\theta_1 = \theta_1 - \theta_{11}$ and $\Delta\theta_2 = \theta_2 - \theta_{21}$. Expand this equation to obtain

$$A(\Delta\theta_1)\cos\Delta\theta_2 + B(\Delta\theta_1)\sin\Delta\theta_2 = C(\Delta\theta_1), \tag{3.33}$$

where the coefficients are given by listed in Appendix Eq 3.37. The solution to this equation is given by

$$\Delta\theta_2 = \arctan\frac{B}{A} \pm \arccos\frac{C}{\sqrt{A^2 + B^2}}. \tag{3.34}$$

The constraint equation for the loop **OADEF** is given by

$$(\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) \cdot (\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)\mathbf{E}^1 - \mathbf{F}) = k^2, \tag{3.35}$$

where

$$\mathbf{S}_{1q}(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3) = \mathbf{T}_3(\theta_1, \theta_2, \theta_3)\mathbf{T}_3(\theta_{11}.\theta_{21}, \theta_{31})^{-1}, \tag{3.36}$$

and $\Delta\theta_3 = \theta_3 - \theta_{31}$. Expand this equation to obtain

$$D(\Delta\theta_1, \Delta\theta_2)\cos\Delta\theta_3 + E(\Delta\theta_1, \Delta\theta_2)\sin\Delta\theta_3 = F(\Delta\theta_1, \Delta\theta_2), \tag{3.37}$$

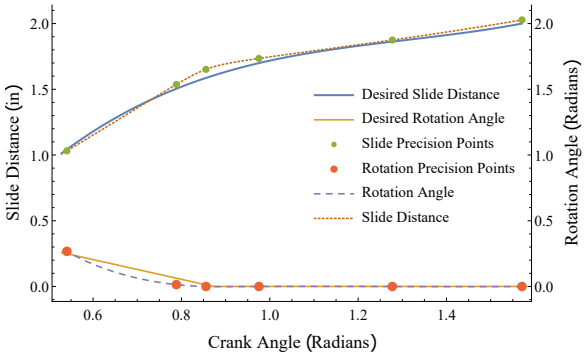


Figure 3.8: The movement of the plunger slide with link **BC** and sealing rotation with link **EF**.

where the coefficients are given in the Appendix Eq 3.58. The solution to this equation is

$$\Delta\theta_3 = \arctan\frac{E}{D} \pm \arccos\frac{F}{\sqrt{D^2+E^2}}. \tag{3.38}$$

For each value of θ_1, equation (3.34) yields two values for θ_2, which we denote as θ_2^+ and θ_2^-. Similarly, equation (3.38) yields two values θ_3^+ and θ_3^-. Therefore, we obtain four joint trajectories,

$$\begin{aligned}\mathbf{q}^1 =&(\theta_1, \theta_2^+, \theta_3^+), \mathbf{q}^2 = (\theta_1, \theta_2^+, \theta_3^-),\\ \mathbf{q}^3 =&(\theta_1, \theta_2^-, \theta_3^+), \text{ and } \mathbf{q}^4 = (\theta_1, \theta_2^-, \theta_3^-).\end{aligned} \tag{3.39}$$

Each of these trajectories represent one branch of the linkage. The selected task points are compared to the four branches to verify they lie on a single branch. The branch is then checked for continuity between the task points, which will allow the linkage to smoothly move through all the task points, thus avoiding branch defects. Figure 3.8 shows an example of an RPR-2SS mechanism that successfully passed the analysis.

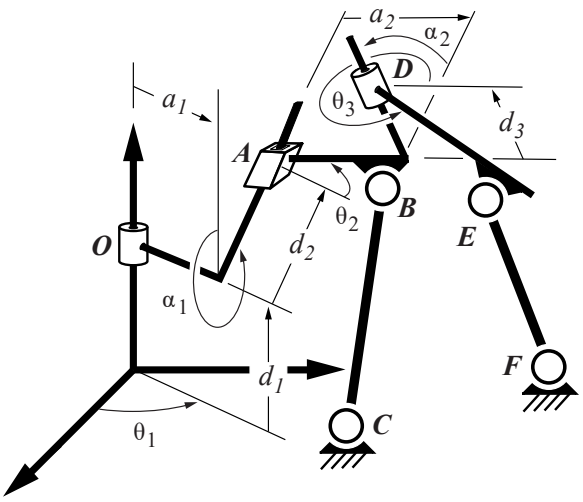


Figure 3.9: The spatial RPR-2SS linkage constructed by constraining a spatial RPR serial chain using two SS dyads that connect the second and third links to the ground frame.

3.3 RPR-2SS Mechanism Synthesis

Table 3.6: Denavit-Hartenberg table for the RPR serial chain. θ_1, d_2, θ_3 are joint variables. The remaining parameters are selected by the designer.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	d_1	α_1	a_1
2	θ_2	d_2	α_2	a_2
3	θ_3	d_3	–	–

3.3.1 The Spatial RPR Chain

To design the RPR-2SS spatial six-bar linkage the RPR chain must first be specified by the designer. The Denavit-Hartenburg parameters in Table 3.6 are shown in Figure 3.9, where θ_1, d_2, θ_3 are joint variables and θ_2,d_1, d_3, $\alpha_i, i = 1, 2$ and $a_i, i = 1, 2$ are specified by the designer.

3.3.2 SS Constraint Synthesis

The joint trajectories $\theta_1(t)$, $d_2(t)$, and $\theta_3(t)$ are prescribed functions of parameter t that define the movement of the RPR serial chain. To create a one degree of freedom system the RPR chain can be constrained by two SS constraints. The SS dyad **BC** connects frame L_2 on link **AD** to the ground frame, and the other SS dyad **EF** connects frame L_3 on link **DE** to the ground frame, shown in Figure 3.9. The loop **OABC** is an RPSS four bar linkage, which Chen and Roth [14] show has a maximum number of five relative position that results in a solution space that is a twentieth-order space curve. See Innocenti[44] and McCarthy and Soh[54] for the SS link design equations.

For an SS dyad the design equations have the linear product decomposition [54] $< x, y, z, 1 >< u, v, w, 1 >= 0$, and has the following expanded form,

$$< xu, xv, xw, yu, yv, yw, zu, zv, zw, x, y, z, u, v, w, 1 > \tag{3.40}$$

However, in the case of constraining an RP serial chain the polynomial is structured such that $xw = yw = uz = vz = zw = 0$. This structure allows for a maximum of six task points leading to five design equations, leaving a free parameter in the z direction. To constrain this parameter an additional constraint equation that contains either w, z or both must be added.

The SS constraint synthesis follows the same procedure as in the RRR, but will only have six task points. The designer specified parameter for constraints **BC** and **EF** must be added to the sets of equations $\mathcal{A}_j$ and $\mathcal{B}_j$, respectively. For link **BC**, the free parameter equation must include either z or w. There is an additional free parameter for the **EF** constraint, that must include either o or r. For example, the free parameter equation of link **BC** can be the length of link **OC**, where $u^2 + v^2 + w^2 = 0$ or to fix joint **B** or **C** to a plane in space,

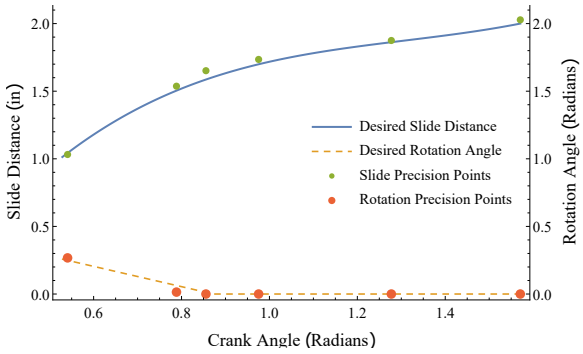


Figure 3.10: The desired slide and rotation function and the respective task points are shown. The task points have been adjusted from the curves during design process.

where $z * x = 0$ or $w = 0$, respectively.

The two sets of equations can be solve independently to find pairs of SS dyads that guide the RPR serial chain through six task points $\mathbf{q}_j, j = 1, \ldots, 6$. The maximum number of SS dyads depends on chosen free parameter equations.

3.3.3 RPR-2SS Synthesis Example

Table 3.7: Denavit-Hartenberg table for the RPR serial chain.

Joint	θ_i	d_i	α_i	a_i
1	θ_1	0.	0.79	1.
2	0.	d_2	0.	0.
3	θ_3	1.	–	–

For the example RPR-2SS synthesis the joint trajectories are given by

$$\begin{aligned}
\theta_1 =& t, \\
d_2 =& -1.25 + 6.37t - 4.56t^2 + 1.16t^3, \\
\theta_3 =& \begin{cases} -\frac{15}{20}t + 0.65 & t < 0.87 \\ 0 & t \geq 0.87. \end{cases}
\end{aligned} \tag{3.41}$$

Table 3.8: Six task points (radians and inches) of the RPR spatial chain chosen from the joint trajectories to design the SS dyads

Point $\mathbf{q}_j$	θ_1	d_2	θ_3
1	1.57	2.03	0.
2	1.28	1.87	0.
3	0.98	1.73	0.
4	0.86	1.65	0.
5	0.79	1.54	0.01
6	0.54	1.03	0.27

These functions have been plotted in Figure 3.10. The six task points selected from these trajectories are given in Table 3.8. The free parameters were chosen such that the joints **C** and **F** are fixed to planes $w = 0$ and $r = 0$, respectively.

The task points are substituted into the design equations $\mathcal{A}_i, j = 2, \ldots, 6$ for the link **BC**, which results in the following,

$$\begin{aligned}
\mathcal{A}_2: \quad & 0.09ux - 0.58uy + 0.21u + 0.58vx + 0.09vy - 0.06v \\
& + 0.22w - 0.22x - 0.22z + 0.023 = 0 \\
\mathcal{A}_3: \quad & 0.34ux - 1.12uy + 0.34u + 1.12vx + 0.34vy - 0.23v \\
& + 0.41w - 0.41x - 0.41z + 0.09 = 0 \\
\mathcal{A}_4: \quad & 0.49ux - 1.31uy + 0.4u + 1.31vx + 0.49vy - 0.35v \\
& + 0.53w - 0.53x - 0.53z + 0.14 = 0 \\
\mathcal{A}_5: \quad & 0.58ux - 1.41uy + 0.49u + 1.41vx + 0.58vy - 0.49v \\
& + 0.69w - 0.69x - 0.69z + 0.24 = 0 \\
\mathcal{A}_6: \quad & 0.97ux - 1.71uy + 0.72u + 1.71vx + 0.97vy - 1.21v \\
& + 1.41w - 1.41x - 1.41z + 0.99 = 0
\end{aligned} \tag{3.42}$$

From Chen and Roth [14], we find that the RPSS chain can be obtained as the intersection

of a fourth order surface with a fifth order surface to obtain a 20th order space curve in the moving body. Intersecting the 20th order curve with the hyperplane w=0 should yield 20 solutions. Based on the design parameters we specified, our solver yielded only five solutions, four complex valued solutions and one real valued solution for $\mathbf{B}^1 = (x, y, z)$ and $\mathbf{C} = (u, v, w)$. The real valued solution is listed in Table 3.9.

Table 3.9: Real-valued solutions to design equations $\mathcal{A}_j$ for $\mathbf{BC}$

	u	v	w	x	y	z
1	1.24	1.29	0.00	0.26	0.49	0.30

The task points are substituted into the design equations $\mathcal{B}_i, j = 2, \ldots, 6$ for the link $\mathbf{EF}$, which results in the following,

$$
\begin{aligned}
\mathcal{B}_2: \quad & 0.09mp + 0.58mq - 0.22m - 0.58np + 0.09nq - 0.22o \\
& + 0.21p - 0.06q + 0.22r + 0.023 = 0 \\
\mathcal{B}_3: \quad & 0.34mp + 1.12mq - 0.41m - 1.12np + 0.34nq - 0.41o \\
& + 0.34p - 0.23q + 0.41r + 0.086 = 0 \\
\mathcal{B}_4: \quad & 0.49mp + 1.31mq - 0.53m - 1.31np + 0.49nq - 0.53o \\
& + 0.4p - 0.35q + 0.53r + 0.14 = 0 \\
\mathcal{B}_5: \quad & 0.57mp + 1.4mq - 0.67m - 1.4np + 0.57nq - 0.02nr \\
& + 0.01op + 0.01oq - 0.71o + 0.48p - 0.48q + 0.71r \\
& - 2 * 10^{-4}n - 1 * 10^{-4}mr + 1 * 10^{-4}or + .24 = 0 \\
\mathcal{B}_6: \quad & 0.67mp + 1.49mq - 0.04mr - 1.03m - 1.46np \\
& + 0.69nq - 0.37nr - 0.07n + 0.3op + 0.22oq + 0.04or \\
& - 1.78o + 0.47p - 0.92q + 1.78r + 1.06 = 0
\end{aligned}
\tag{3.43}
$$

The solution of these equations together with the constraint equation $r = 0$ produce six complex valued solutions and four real valued solution for $\mathbf{E}^1 = (m, n, o)$ and $\mathbf{F} = (p, q, r)$. The real valued solutions are listed in Table 3.10.

Table 3.10: Real-valued solutions to design equations $\mathcal{B}_j$ for **EF**

	p	q	r	m	n	o
1	3.78	3.41	0.00	0.31	0.56	0.78
2	4.99	-4.04	0.00	-0.14	0.14	5.30
3	1.78	1.53	0.00	0.30	0.49	0.47
4	0.39	-0.02	0.00	-0.33	-0.20	1.00

The set of one solution and four solutions combine to create four candidate linkages that are analyzed to verify performance.

3.3.4 Analysis of RPR-2SS Mechanism

The RPR-2SS mechanism is comprised of two loops: i) **OABC** forms an RPSS closed chain, ii) **OADEF** forms an RPRSS closed chain. The determination of the two SS dyads gives the coordinates of $\mathbf{B^1C}$ and $\mathbf{E^1F}$ and the lengths $h = |\mathbf{B^1C}|$ and $k = |\mathbf{E^1F}|$. To determine the movement of the system, first specify the input crank angle θ_1 then solve the loop constraint equations for d_2 and θ_3.

The loop **OABC** has the loop constraint equation takes the form of

$$A(\Delta\theta_1)\Delta d_2^2 + B(\Delta\theta_1)\Delta d_2 + C(\Delta\theta_1) = 0, \tag{3.44}$$

where the coefficients are listed in Appendix Eq 3.59. The solution to this equation is

$$\Delta d_2 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \tag{3.45}$$

The loop **OADEF** has the loop constraint equation,

$$D(\Delta\theta_1, \Delta d_2)\cos\Delta\theta_3 + E(\Delta\theta_1, \Delta d_2)\sin\Delta\theta_3 = F(\Delta\theta_1, \Delta d_2), \tag{3.46}$$

where the coefficients are listed in Appendix Eq 3.61. The solution to this equation is

$$\Delta\theta_3 = \arctan\frac{E}{D} \pm \arccos\frac{F}{\sqrt{D^2 + E^2}}. \tag{3.47}$$

3.4 PRP-2SS Mechanism Synthesis

Table 3.11: Denavit-Hartenberg table for the PRP serial chain. d_1, θ_2, d_3 are joint variables. The remaining parameters are selected by the designer.

Link i	θ_i	d_i	α_i	a_i
1	θ_1	d_1	α_1	a_1
2	θ_2	d_2	α_2	a_2
3	θ_3	d_3	–	–

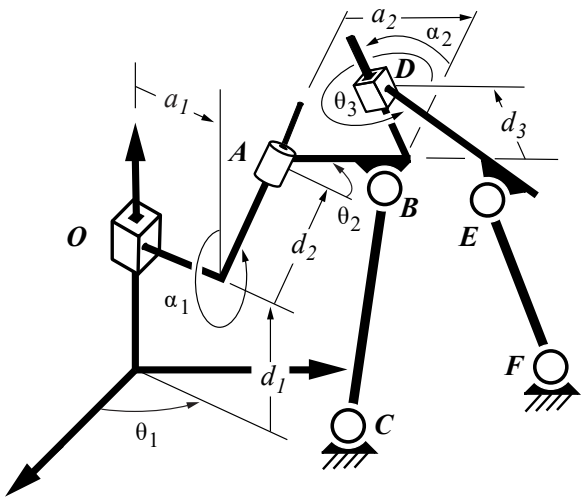


Figure 3.11: The spatial PRP-2SS linkage constructed by constraining a spatial PRP serial chain using two SS dyads that connect the second and third links to the ground frame.

3.4.1 The Spatial PRP Chain

To design the PRP-2SS spatial six-bar linkage the PRP chain must first be specified by the designer. The Denavit-Hartenburg parameters in Table 3.11 are shown in Figure 3.11, where d_1, θ_2, d_3 are joint variables and θ_1,, d_2, θ_3, $\alpha_i, i = 1, 2$ and $a_i, i = 1, 2$ are specified by the designer.

3.4.2 SS Constraint Synthesis

The joint trajectories $d_1(t)$, $\theta_2(t)$, and $d_3(t)$ are prescribed functions of parameter t that define the movement of the PRP serial chain. To create a one degree of freedom system the PRP chain can be constrained by two SS constraints. The SS dyad **BC** connects frame L_2 on link **AD** to the ground frame, and the other SS dyad **EF** connects frame L_3 on link **DE** to the ground frame, shown in Figure 3.11. The loop **OABC** is an PRSS four bar linkage. Chen and Roth [14] show that a RPSS has a maximum number of five relative position that results in a solution space that is a twentieth-order space curve. They also show that the kinematic inversion of a linkage takes the same form as the original, which means that the

locus of points that satisfy the link constraint must be the same. Since PRSS is an inversion of the RPSS linkage, the PRSS linkage must also be able to satisfy a maximum number of five relative positions. See Innocenti[44] and McCarthy and Soh[54] for the SS link design equations.

For an SS dyad the design equations have the linear product decomposition [54] $< x, y, z, 1 >< u, v, w, 1 >= 0$, and has the following expanded form,

$$< xu, xv, xw, yu, yv, yw, zu, zv, zw, x, y, z, u, v, w, 1 > \tag{3.48}$$

However, in the case of constraining a PR serial chain the polynomial is structured such that $xw = yw = uz = vz = zw = 0$. This structure allows for a maximum of six task points leading to five design equations, leaving a free parameter in the z direction. To constrain this parameter an additional constraint equation that contains either w, z or both must be added.

The designer specified parameter for constraints **BC** and **EF** must be added to the sets of equations $\mathcal{A}_j$ and $\mathcal{B}_j$, respectively. For link **BC**, the free parameter equation must include either z or w. There is an additional free parameter for the **EF** constraint, that must include either o or r. For example, the free parameter equation of link **BC** can be the length of link **OC**, where $u^2 + v^2 + w^2 = 0$ or to fix joint **B** or **C** to a plane in space, where $z * x = 0$ or $w = 0$, respectively.

The two sets of equations can be solve independently to find pairs of SS dyads that guide the PRP serial chain through six task points $\mathbf{q}_j, j = 1, \ldots, 6$. The maximum number of SS dyads depends on chosen free parameter equations.

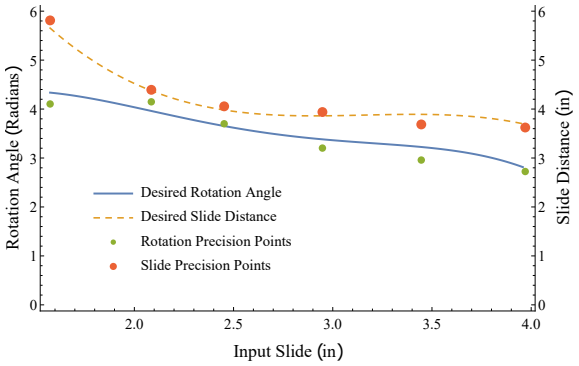


Figure 3.12: The desired slide and rotation function and the respective task points are shown. The task points have been adjusted from the curves during design process.

3.4.3 PRP-2SS Synthesis Example

Table 3.12: Denavit-Hartenberg table for the PRP serial chain.

Joint	θ_i	d_i	α_i	a_i
1	-0.768	$d1$	-2.802	0.347
2	θ_2	2.025	1.830	1.105
3	0.	$d3$	0	0

Table 3.13: Six task points (radians and inches) of the PRP spatial chain chosen from the joint trajectories to design the SS dyads

Point $\mathbf{q}_j$	d_1	θ_2	d_3
1	3.93	2.85	3.72
2	3.5	3.2	3.89
3	3.	3.36	3.86
4	2.5	3.62	3.95
5	2.	4.04	4.52
6	1.5	4.36	5.94

The following functions that are used as an example to design the PRP-2SS mechanism.

$$
\begin{aligned}
d_1 &= t, \\
\theta_2 &= -2.70 + 13.32t - 8.69t^2 + 2.29t^3 - 0.22t^4, \\
d_3 &= 18.90 - 14.46t + 4.61t^2 - 0.49t^3
\end{aligned}
\tag{3.49}
$$

Figure 3.12 shows the plot of these functions. Table 3.13 gives the six task points that are selected from these curves. The free parameters were chosen such that the joints **C** and **F** are fixed to planes $w = 0$ and $r = 0$, respectively.

The task points are substituted into the design equations $\mathcal{A}_i, j = 2, \ldots, 6$ for the link **BC**, which results in the following,

$$\begin{aligned}
\mathcal{A}_2: \quad & 0.05ux - 0.44uy - 0.1uz + 0.27u + 0.43vx + 0.05vy \\
& + 0.12vz - 0.57v - 0.33x + 0.45y - 1.02z + 0.34 = 0 \\
\mathcal{A}_3: \quad & 0.21ux - 0.88uy - 0.17uz + 0.42u + 0.86vx + 0.21v \\
& y + 0.26vz - 1.21v - 0.64x + 0.75y - 2.01z + 1.25 = 0 \\
\mathcal{A}_4: \quad & 0.83ux - 1.61uy - 0.21uz + 0.22u + 1.51vx + 0.83vy \\
& + 0.58vz - 2.49v - 1.29x + 1.06y - 3.08z + 3.07 = 0 \\
\mathcal{A}_5: \quad & 1.62ux - 1.96uy - 0.1uz - 0.47u + 1.77vx + 1.61vy \\
& + 0.84vz - 3.4v - 1.74x + 1.03y - 3.94z + 4.89 = 0 \\
\mathcal{A}_6: \quad & 1.53ux - 1.94uy - 0.12uz - 0.39u + 1.76vx + 1.53vy \\
& + 0.82vz - 3.33v - 1.26x + 0.99y - 4.85z + 6.51 = 0
\end{aligned} \tag{3.50}$$

Chen and Roth [14] show that the RPSS chain can be obtained as the intersection of a fourth order surface with a fifth order surface to obtain a 20th order space curve in the moving body. Intersecting the 20th order curve with the hyperplane w=0 should yield 20 solutions. Based on the design parameters we specified, our solver yielded only ten solutions, six complex valued solutions, three infinite solutions and one real valued solution for $\mathbf{B}^1 = (x, y, z)$ and $\mathbf{C} = (u, v, w)$. The real valued solution is listed in Table 3.14.

The design equations $\mathcal{B}_i, j = 2, \ldots, 6$ for the link **EF** that have substituted the task points

Table 3.14: Real-valued solutions to design equations $\mathcal{A}_j$ for **BC**

	u	v	w	x	y	z
1	0.804	0.585	0	-2.3	2.7	0.631

result in the following,

$$\begin{aligned}
\mathcal{B}_2: \quad & 0.05mp + 0.43mq - 0.38m - 0.44np + 0.05nq + 0.33n \\
& - 0.1op + 0.12oq - 1.02o + 0.34p - 0.47q + 0.32 = 0 \\
\mathcal{B}_3: \quad & 0.21mp + 0.86mq - 0.87m - 0.88np + 0.21nq + 0.16n \\
& - 0.17op + 0.26oq - 2.04o + 0.88p - 0.78q + 1.24 = 0 \\
\mathcal{B}_4: \quad & 0.83mp + 1.51mq - 1.59m - 1.61np + 0.83nq + 0.25n \\
& - 0.21op + 0.58oq - 3.13o + 1.05p - 2.26q + 3.10 = 0 \\
\mathcal{B}_5: \quad & 1.62mp + 1.77mq - 2.29m - 1.96np + 1.61nq - 0.41n \\
& - 0.1op + 0.84oq - 4.02o + 1.04p - 3.64q + 5.39 = 0 \\
\mathcal{B}_6: \quad & 1.53mp + 1.76mq - 2.81m - 1.94np + 1.53nq - 3.09n \\
& - 0.12op + 0.82oq - 5.08o + 3.95p - 3.83q + 10.81 = 0
\end{aligned} \tag{3.51}$$

The constraint equation $r = 0$ solved in conjunction with the five design equations produce six complex valued solutions, two infinite solutions and two real valued solution for $\mathbf{E}^1 = (m, n, o)$ and $\mathbf{F} = (p, q, r)$. The real valued solutions are listed in Table 3.15.

Table 3.15: Real-valued solutions to design equations $\mathcal{B}_j$ for **EF**

	p	q	r	m	n	o
1	0.49	0.36	0.	-33.89	6.59	7.63
2	-1.7	0.83	0.	1.02	0.23	-0.67

The set of one solution and four solutions combine to create four candidate linkages that are analyzed to verify performance.

3.4.4 Analysis of PRP-2SS Mechanism

The PRP-2SS mechanism is comprised of two loops: i) **OABC** forms an PRSS closed chain, ii) **OADEF** forms an PRPSS closed chain. The determination of the two SS dyads gives the coordinates of $\mathbf{B^1C}$ and $\mathbf{E^1F}$ and the lengths $h = |\mathbf{B^1C}|$ and $k = |\mathbf{E^1F}|$. To determine the movement of the system, first specify the input crank angle d_1 then solve the loop constraint equations for θ_2 and $d3_3$.

The loop **OABC** has the loop constraint equation,

$$A(\Delta d_1)\cos\Delta\theta_2 + B(\Delta d_1)\sin\Delta\theta_2 = C(\Delta d_1), \tag{3.52}$$

where the coefficients are given by listed in Appendix Eq 3.62. The solution to this equation is given by

$$\Delta\theta_2 = \arctan\frac{B}{A} \pm \arccos\frac{C}{\sqrt{A^2+B^2}}. \tag{3.53}$$

Expand this equation to obtain

$$A(\Delta d_1, \Delta\theta_2)\Delta d_3^2 + B(\Delta d_1, \Delta\theta_2)\Delta d_3 + C(\Delta d_1, \Delta\theta_2) = 0, \tag{3.54}$$

where the coefficients are listed in Appendix Eq 3.63. The solution to this equation is

$$\Delta d_3 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \tag{3.55}$$

The task points are then checked to ensure that they lie on only one of the four branches, seen in Eq 3.39, then the branch is checked for continuity.

3.5 Summary

The kinematic synthesis of the RRR-RR-SS, RRR-2SS, RPR-2SS and the PRP-2SS mechanisms is described in this chapter. Following the synthesis technique the choice of spatial serial chain and input functions are up to the designer, which allows the designer to coordinate any one input and two outputs or to drive a spatial serial chain through a specified motion defined by the relative joint positions.

Appendix: Coefficients for the RRR-2SS Analysis Equations

The coefficients for Eq. 3.33 are listed here:

$$
\begin{aligned}
A(\Delta\theta_1) =& 2a_1x + 2d_1y\sin\alpha_1 - 2ux\cos\Delta\theta_1 \\
&+ 2uy\cos\alpha_1\sin\Delta\theta_1 - 2vx\sin\Delta\theta_1 \\
&- 2vy\cos\alpha_1\cos\Delta\theta_1 - 2wy\sin\alpha_1, \\
B(\Delta\theta_1) =& -2a_1y + 2d_1x\sin\alpha_1 + 2ux\cos\alpha_1\sin\Delta\theta_1 \\
&+ 2uy\cos\Delta\theta_1 - 2vx\cos\alpha_1\cos\Delta\theta_1 \\
&+ 2vy\sin\Delta\theta_1 - 2wx\sin\alpha_1, \text{ and} \\
C(\Delta\theta_1) =& v^2 + w^2 + x^2 + y^2 + z^2 + d_2^2 + u^2 - 2d_1w + 2d_2z \\
&+ d_1^2 + a_1^2 - b^2 - 2uz\sin\alpha_1\sin\Delta\theta_1 \\
&+ 2\cos\alpha_1(d_1 - w)(d_2 + z) - 2a_1v\sin\Delta\theta_1 \\
&+ \cos\Delta\theta_1(2v\sin\alpha_1(d_2 + z) - 2a_1u) \\
&- 2d_2u\sin\alpha_1\sin\Delta\theta_1.
\end{aligned} \tag{3.56}
$$

The coefficients for Eq. 3.37 are listed here:

$$
\begin{aligned}
D(\Delta\theta_1, \Delta\theta_2) =& 2a_1 m \cos\Delta\theta_2 + 2a_2 m - 2a_1 n \cos\alpha_2 \sin\Delta\theta_2 \\
&+ 2d_1 m \sin\alpha_1 \sin\Delta\theta_2 + 2d_2 n \sin\alpha_2 \\
&+ 2d_1 n \sin\alpha_1 \cos\alpha_2 \cos\Delta\theta_2 + 2d_1 n \sin\alpha_2 \cos\alpha_1 \\
&+ 2mp \cos\alpha_1 \sin\Delta\theta_1 \sin\Delta\theta_2 - 2mp \cos\Delta\theta_1 \cos\Delta\theta_2 \\
&- 2mq \cos\alpha_1 \sin\Delta\theta_2 \cos\Delta\theta_1 \\
&- 2mq \sin\Delta\theta_1 \cos\Delta\theta_2 - 2mr \sin\alpha_1 \sin\Delta\theta_2 \\
&- 2np \sin\alpha_1 \sin\alpha_2 \sin\Delta\theta_1 \\
&+ 2np \cos\alpha_1 \cos\alpha_2 \sin\Delta\theta_1 \cos\Delta\theta_2 \\
&+ 2np \cos\alpha_2 \sin\Delta\theta_2 \cos\Delta\theta_1 \\
&- 2nq \cos\alpha_1 \cos\alpha_2 \cos\Delta\theta_1 \cos\Delta\theta_2 \\
&+ 2nq \sin\alpha_1 \sin\alpha_2 \cos\Delta\theta_1 + 2nq \cos\alpha_2 \sin\Delta\theta_1 \sin\Delta\theta_2 \\
&- 2nr \sin\alpha_1 \cos\alpha_2 \cos\Delta\theta_2 - 2nr \sin\alpha_2 \cos\alpha_1, \\
E(\Delta\theta_1, \Delta\theta_2) =& - 2a_1 m \cos\alpha_2 \sin\Delta\theta_2 - 2a_1 n \cos\Delta\theta_2 - 2a_2 n \\
&+ 2d_1 m \sin\alpha_1 \cos\alpha_2 \cos\Delta\theta_2 + 2d_2 m \sin\alpha_2 \\
&+ 2d_1 m \sin\alpha_2 \cos\alpha_1 - 2d_1 n \sin\alpha_1 \sin\Delta\theta_2 \\
&- 2mp \sin\alpha_1 \sin\alpha_2 \sin\Delta\theta_1 + 2mp \cos\alpha_2 \sin\Delta\theta_2 \cos\Delta\theta_1 \\
&+ 2mp \cos\alpha_1 \cos\alpha_2 \sin\Delta\theta_1 \cos\Delta\theta_2 \\
&- 2mq \cos\alpha_1 \cos\alpha_2 \cos\Delta\theta_1 \cos\Delta\theta_2 + 2mq \sin\alpha_1 \sin\alpha_2 \cos\Delta\theta_1 \\
&+ 2mq \cos\alpha_2 \sin\Delta\theta_1 \sin\Delta\theta_2 - 2mr \sin\alpha_1 \cos\alpha_2 \cos\Delta\theta_2 \\
&- 2mr \sin\alpha_2 \cos\alpha_1 - 2np \cos\alpha_1 \sin\Delta\theta_1 \sin\Delta\theta_2 \\
&+ 2np \cos\Delta\theta_1 \cos\Delta\theta_2 + 2nq \cos\alpha_1 \sin\Delta\theta_2 \cos\Delta\theta_1 \\
&+ 2nq \sin\Delta\theta_1 \cos\Delta\theta_2 + 2nr \sin\alpha_1 \sin\Delta\theta_2, \text{ and}
\end{aligned}
\tag{3.57}
$$

$$
\begin{aligned}
F(\Delta\theta_1, \Delta\theta_2) =& 2a_2d_1 \sin\alpha_1 \sin\Delta\theta_2 + 2a_1d_3 \sin\alpha_2 \sin\Delta\theta_2 \\
&+ 2\cos\alpha_1(a_2p \sin\Delta\theta_1 \sin\Delta\theta_2 - a_2q \sin\Delta\theta_2 \cos\Delta\theta_1 \\
&+ \sin\alpha_2(d_3 + o)\cos\Delta\theta_2(q\cos\Delta\theta_1 - p\sin\Delta\theta_1) \\
&+ \cos\alpha_2(d_3 + o)(d_1 - r) - d_2r + d_1d_2) \\
&+ 2a_1a_2\cos\Delta\theta_2 + 2a_1o\sin\alpha_2\sin\Delta\theta_2 - 2a_1p\cos\Delta\theta_1 \\
&- 2a_2p\cos\Delta\theta_1\cos\Delta\theta_2 - 2a_1q\sin\Delta\theta_1 - 2a_2q\sin\Delta\theta_1\cos\Delta\theta_2 \\
&- 2a_2r\sin\alpha_1\sin\Delta\theta_2 + a_1^2 + a_2^2 - b^2 \\
&- 2d_1d_3\sin\alpha_1\sin\alpha_2\cos\Delta\theta_2 - 2d_1o\sin\alpha_1\sin\alpha_2\cos\Delta\theta_2 \\
&+ 2\cos\alpha_2(d_3 + o)(d_2 - p\sin\alpha_1\sin\Delta\theta_1 + q\sin\alpha_1\cos\Delta\theta_1) \\
&+ 2d_3o - 2d_2p\sin\alpha_1\sin\Delta\theta_1 - 2d_3p\sin\alpha_2\sin\Delta\theta_2\cos\Delta\theta_1 \\
&- 2d_3q\sin\alpha_2\sin\Delta\theta_1\sin\Delta\theta_2 + 2d_2q\sin\alpha_1\cos\Delta\theta_1 \\
&+ 2d_3r\sin\alpha_1\sin\alpha_2\cos\Delta\theta_2 - 2op\sin\alpha_2\sin\Delta\theta_2\cos\Delta\theta_1 \\
&- 2oq\sin\alpha_2\sin\Delta\theta_1\sin\Delta\theta_2 + 2or\sin\alpha_1\sin\alpha_2\cos\Delta\theta_2 \\
&+ p^2 + q^2 + r^2 - 2d_1r + d_1^2 + d_2^2 + d_3^2 + m^2 + n^2 + o^2. \qquad (3.58)
\end{aligned}
$$

Appendix: RPR-2SS Coefficients for the Analysis Equations

The coefficients for Eq. 3.44 are listed here:

$$
\begin{aligned}
\mathrm{A}(\Delta\theta_1) =& 1 \\
\mathrm{B}(\Delta\theta_1) =& (2d_1\cos(\alpha_1) - 2u\sin(\alpha_1)\sin(\Delta\theta_1) + 2v\sin(\alpha_1)\cos(\Delta\theta_1) \\
& - 2w\cos(\alpha_1) + 2z) \\
\mathrm{C}(\Delta\theta_1) =& (-2a_1u\cos(\Delta\theta_1) - 2a_1v\sin(\Delta\theta_1) + 2a_1x\cos(\theta_2) \\
& - 2a_1y\sin(\theta_2) + a_1^2 - b^2 - 2d_1w + 2d_1x\sin(\alpha_1)\sin(\theta_2) \\
& + 2d_1y\sin(\alpha_1)\cos(\theta_2) + 2d_1z\cos(\alpha_1) + d_1^2 + u^2 \\
& + 2ux\cos(\alpha_1)\sin(\Delta\theta_1)\sin(\theta_2) - 2ux\cos(\Delta\theta_1)\cos(\theta_2) \\
& + 2uy\cos(\alpha_1)\sin(\Delta\theta_1)\cos(\theta_2) + 2uy\cos(\Delta\theta_1)\sin(\theta_2) \\
& - 2uz\sin(\alpha_1)\sin(\Delta\theta_1) + v^2 - 2vx\cos(\alpha_1)\cos(\Delta\theta_1)\sin(\theta_2) \\
& - 2vx\sin(\Delta\theta_1)\cos(\theta_2) - 2vy\cos(\alpha_1)\cos(\Delta\theta_1)\cos(\theta_2) \\
& + 2vy\sin(\Delta\theta_1)\sin(\theta_2) + 2vz\sin(\alpha_1)\cos(\Delta\theta_1) + w^2 \\
& - 2wx\sin(\alpha_1)\sin(\theta_2) - 2wy\sin(\alpha_1)\cos(\theta_2) - 2wz\cos(\alpha_1) \\
& + x^2 + y^2 + z^2)
\end{aligned}
\tag{3.59}
$$

The coefficients for Eq. 3.46 are listed here:

$$
\begin{aligned}
\mathrm{D}(\Delta\theta_1) =& (2a_1 m\cos(\theta_2) + 2a_2 m - 2a_1 n\cos(\alpha_2)\sin(\theta_2) \\
&+ 2d_1 m\sin(\alpha_1)\sin(\theta_2) + 2d_1 n\sin(\alpha_1)\cos(\alpha_2)\cos(\theta_2) \\
&+ 2d_1 n\sin(\alpha_2)\cos(\alpha_1) + 2mp\cos(\alpha_1)\sin(\Delta\theta_1)\sin(\theta_2) \\
&- 2mp\cos(\Delta\theta_1)\cos(\theta_2) - 2mq\cos(\alpha_1)\cos(\Delta\theta_1)\sin(\theta_2) \\
&- 2mq\sin(\Delta\theta_1)\cos(\theta_2) - 2mr\sin(\alpha_1)\sin(\theta_2) \\
&+ 2\Delta d_2 n\sin(\alpha_2) + 2np\cos(\alpha_1)\cos(\alpha_2)\sin(\Delta\theta_1)\cos(\theta_2) \\
&+ 2np\cos(\alpha_2)\cos(\Delta\theta_1)\sin(\theta_2) - 2np\sin(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_1) \\
&- 2nq\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_1)\cos(\theta_2) + 2nq\cos(\alpha_2)\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ 2nq\sin(\alpha_1)\sin(\alpha_2)\cos(\Delta\theta_1) \\
&- 2nr\sin(\alpha_1)\cos(\alpha_2)\cos(\theta_2) - 2nr\sin(\alpha_2)\cos(\alpha_1)) \\
\mathrm{E}(\Delta\theta_1) =& (-2a_1 m\cos(\alpha_2)\sin(\theta_2) - 2a_1 n\cos(\theta_2) - 2a_2 n \\
&+ 2d_1 m\sin(\alpha_1)\cos(\alpha_2)\cos(\theta_2) + 2d_1 m\sin(\alpha_2)\cos(\alpha_1) \\
&- 2d_1 n\sin(\alpha_1)\sin(\theta_2) + 2\Delta d_2 m\sin(\alpha_2) + 2mp\cos(\alpha_1)\cos(\alpha_2)\sin(\Delta\theta_1)\cos(\theta_2) \\
&+ 2mp\cos(\alpha_2)\cos(\Delta\theta_1)\sin(\theta_2) - 2mp\sin(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_1) \\
&- 2mq\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_1)\cos(\theta_2) + 2mq\cos(\alpha_2)\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ 2mq\sin(\alpha_1)\sin(\alpha_2)\cos(\Delta\theta_1) - 2mr\sin(\alpha_1)\cos(\alpha_2)\cos(\theta_2) \\
&- 2mr\sin(\alpha_2)\cos(\alpha_1) - 2np\cos(\alpha_1)\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ 2np\cos(\Delta\theta_1)\cos(\theta_2) + 2nq\cos(\alpha_1)\cos(\Delta\theta_1)\sin(\theta_2) \\
&+ 2nq\sin(\Delta\theta_1)\cos(\theta_2) + 2nr\sin(\alpha_1)\sin(\theta_2))
\end{aligned}
\tag{3.60}
$$

$$
\begin{aligned}
\mathrm{F}(\Delta\theta_1) =& (2a_2d_1\sin(\alpha_1)\sin(\theta_2) + 2a_1d_3\sin(\alpha_2)\sin(\theta_2) + 2\cos(\alpha_1)(a_2p\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ a_2(-q)\cos(\Delta\theta_1)\sin(\theta_2) + d_1\Delta d_2 + \sin(\alpha_2)(d_3+o)\cos(\theta_2)(q\cos(\Delta\theta_1) - p\sin(\Delta\theta_1)) \\
&+ \cos(\alpha_2)(d_3+o)(d_1-r) - \Delta d_2 r) + 2a_1a_2\cos(\theta_2) + 2a_1o\sin(\alpha_2)\sin(\theta_2) \\
&- 2a_2p\cos(\Delta\theta_1)\cos(\theta_2) - 2a_1p\cos(\Delta\theta_1) - 2a_2q\sin(\Delta\theta_1)\cos(\theta_2) - 2a_1q\sin(\Delta\theta_1) \\
&- 2a_2r\sin(\alpha_1)\sin(\theta_2) + a_1^2 + a_2^2 - b^2 - 2d_1d_3\sin(\alpha_1)\sin(\alpha_2)\cos(\theta_2) \\
&- 2d_1o\sin(\alpha_1)\sin(\alpha_2)\cos(\theta_2) \\
&+ 2\cos(\alpha_2)(d_3+o)(\Delta d_2 - p\sin(\alpha_1)\sin(\Delta\theta_1) + q\sin(\alpha_1)\cos(\Delta\theta_1)) + 2d_3o \\
&- 2d_3p\sin(\alpha_2)\cos(\Delta\theta_1)\sin(\theta_2) \\
&- 2d_3q\sin(\alpha_2)\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ 2d_3r\sin(\alpha_1)\sin(\alpha_2)\cos(\theta_2) - 2d_1r + d_1^2 + d_3^2 \\
&+ \Delta d_2^2 + m^2 + n^2 + o^2 - 2op\sin(\alpha_2)\cos(\Delta\theta_1)\sin(\theta_2) \\
&- 2oq\sin(\alpha_2)\sin(\Delta\theta_1)\sin(\theta_2) \\
&+ 2or\sin(\alpha_1)\sin(\alpha_2)\cos(\theta_2) + p^2 \\
&- 2\Delta d_2p\sin(\alpha_1)\sin(\Delta\theta_1) + q^2 \\
&+ 2\Delta d_2q\sin(\alpha_1)\cos(\Delta\theta_1) + r^2)
\end{aligned}
\tag{3.61}
$$

Appendix: Coefficients for the Analysis Equations

The coefficients for Eq. 3.52 are listed here:

$$
\begin{aligned}
\mathrm{A}(\Delta\theta_1) =& 2a_1x - 2uxCos(\theta) - 2vyCos(\alpha_1)Cos(\theta) + 2d_1ySin(\alpha_1) \\
& - 2wySin(\alpha_1) - 2vxSin(\theta) + 2uyCos(\alpha_1)Sin(\theta) \\
\mathrm{B}(\Delta\theta_1) =& -2a_1y + 2d_1x\sin(\alpha_1) + 2ux\cos(\alpha_1)\sin(\Delta\theta_1) \\
& + 2uy\cos(\Delta\theta_1) - 2vx\cos(\alpha_1)\cos(\Delta\theta_1) + 2vy\sin(\Delta\theta_1) \\
& - 2wx\sin(\alpha_1) \\
\mathrm{C}(\Delta\theta_1) =& \cos(\Delta\theta_1)(2v\sin(\alpha_1)(d_2+z) - 2a_1u) - 2a_1v\sin(\Delta\theta_1) \\
& + a_1^2 - b^2 - 2d_2u\sin(\alpha_1)\sin(\Delta\theta_1) \\
& + 2\cos(\alpha_1)(d_1-w)(d_2+z) - 2d_1w + 2d_2z + d_1^2 + d_2^2 \\
& + u^2 - 2uz\sin(\alpha_1)\sin(\Delta\theta_1) + v^2 + w^2 + x^2 + y^2 + z^2
\end{aligned}
\tag{3.62}
$$

The coefficients for Eq. 3.54 are listed here:

$$
\begin{aligned}
\mathrm{D}(\Delta\theta_1) =& 1 \\
\mathrm{E}\Delta\theta_1) =& 2a_1 \sin(\alpha_2)\sin(\Delta\theta_2) - 2d_1 \sin(\alpha_1)\sin(\alpha_2)\cos(\Delta\theta_2) \\
&+ 2d_1 \cos(\alpha_1)\cos(\alpha_2) + 2d_2 \cos(\alpha_2) + 2o \\
&- 2p \sin(\alpha_1)\cos(\alpha_2)\sin(\Delta\theta_1) \\
&- 2p \sin(\alpha_2)\cos(\alpha_1)\sin(\Delta\theta_1)\cos(\Delta\theta_2) \\
&- 2p \sin(\alpha_2)\sin(\Delta\theta_2)\cos(\Delta\theta_1) \\
&- 2q \sin(\alpha_2)\sin(\Delta\theta_1)\sin(\Delta\theta_2) \\
&+ 2q \sin(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_1) \\
&+ 2q \sin(\alpha_2)\cos(\alpha_1)\cos(\Delta\theta_1)\cos(\Delta\theta_2) \\
&+ 2r \sin(\alpha_1)\sin(\alpha_2)\cos(\Delta\theta_2) \\
&- 2r \cos(\alpha_1)\cos(\alpha_2)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{F}(\Delta\theta_1) = &- b^2 + m^2 + n^2 + o^2 + p^2 + q^2 + r^2 + a_1^2 + a_2^2 + d_1^2 \\
&+ d_2^2 - 2or\cos(\alpha_1)\cos(\alpha_2) \\
&- 2mp\cos(\Delta\theta_1)\cos(\Delta\theta_2)\cos(\Delta\theta_3) \\
&- 2nq\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_1)\cos(\Delta\theta_2)\cos(\Delta\theta_3) \\
&+ 2oq\cos(\alpha_2)\cos(\Delta\theta_1)\sin(\alpha_1) \\
&- 2nr\cos(\alpha_2)\cos(\Delta\theta_2)\cos(\Delta\theta_3)\sin(\alpha_1) \\
&+ 2oq\cos(\alpha_1)\cos(\Delta\theta_1)\cos(\Delta\theta_2)\sin(\alpha_2) \\
&- 2nr\cos(\alpha_1)\cos(\Delta\theta_3)\sin(\alpha_2) \\
&+ 2or\cos(\Delta\theta_2)\sin(\alpha_1)\sin(\alpha_2) \\
&+ 2nq\cos(\Delta\theta_1)\cos(\Delta\theta_3)\sin(\alpha_1)\sin(\alpha_2) \\
&- 2mq\cos(\Delta\theta_2)\cos(\Delta\theta_3)\sin(\Delta\theta_1) \\
&+ 2np\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_2)\cos(\Delta\theta_3)\sin(\Delta\theta_1) \\
&- 2op\cos(\alpha_2)\sin(\alpha_1)\sin(\Delta\theta_1) \\
&- 2op\cos(\alpha_1)\cos(\Delta\theta_2)\sin(\alpha_2)\sin(\Delta\theta_1) \\
&- 2np\cos(\Delta\theta_3)\sin(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_1) \\
&- 2mq\cos(\alpha_1)\cos(\Delta\theta_1)\cos(\Delta\theta_3)\sin(\Delta\theta_2) \\
&+ 2np\cos(\alpha_2)\cos(\Delta\theta_1)\cos(\Delta\theta_3)\sin(\Delta\theta_2) \\
&- 2mr\cos(\Delta\theta_3)\sin(\alpha_1)\sin(\Delta\theta_2) \\
&- 2op\cos(\Delta\theta_1)\sin(\alpha_2)\sin(\Delta\theta_2) \\
&+ 2mp\cos(\alpha_1)\cos(\Delta\theta_3)\sin(\Delta\theta_1)\sin(\Delta\theta_2) \\
&+ 2nq\cos(\alpha_2)\cos(\Delta\theta_3)\sin(\Delta\theta_1)\sin(\Delta\theta_2)
\end{aligned}
$$

$$
\begin{aligned}
&- 2oq\sin(\alpha_2)\sin(\Delta\theta_1)\sin(\Delta\theta_2) \\
&+ 2np\cos(\Delta\theta_1)\cos(\Delta\theta_2)\sin(\Delta\theta_3) \\
&- 2mq\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_1)\cos(\Delta\theta_2)\sin(\Delta\theta_3) \\
&- 2mr\cos(\alpha_2)\cos(\Delta\theta_2)\sin(\alpha_1)\sin(\Delta\theta_3) \\
&- 2mr\cos(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_3) \\
&+ 2mq\cos(\Delta\theta_1)\sin(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_3) \\
&+ 2nq\cos(\Delta\theta_2)\sin(\Delta\theta_1)\sin(\Delta\theta_3) \\
&+ 2mp\cos(\alpha_1)\cos(\alpha_2)\cos(\Delta\theta_2)\sin(\Delta\theta_1)\sin(\Delta\theta_3) \\
&- 2mp\sin(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_1)\sin(\Delta\theta_3) \\
&+ 2nq\cos(\alpha_1)\cos(\Delta\theta_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3) \\
&+ 2mp\cos(\alpha_2)\cos(\Delta\theta_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3) \\
&+ 2nr\sin(\alpha_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3) \\
&- 2np\cos(\alpha_1)\sin(\Delta\theta_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3) \\
&+ 2mq\cos(\alpha_2)\sin(\Delta\theta_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3) \\
&- 2p\cos(\Delta\theta_1)a_1 + 2m\cos(\Delta\theta_2)\cos(\Delta\theta_3)a_1 \\
&- 2q\sin(\Delta\theta_1)a_1 - 2n\cos(\alpha_2)\cos(\Delta\theta_3)\sin(\Delta\theta_2)a_1 \\
&+ 2o\sin(\alpha_2)\sin(\Delta\theta_2)a_1 - 2n\cos(\Delta\theta_2)\sin(\Delta\theta_3)a_1 \\
&- 2m\cos(\alpha_2)\sin(\Delta\theta_2)\sin(\Delta\theta_3)a_1 - 2p\cos(\Delta\theta_1)\cos(\Delta\theta_2)a_2
\end{aligned}
$$

$$
\begin{aligned}
&+ 2m\cos(\Delta\theta_3)a_2 - 2q\cos(\Delta\theta_2)\sin(\Delta\theta_1)a_2 \\
&- 2q\cos(\alpha_1)\cos(\Delta\theta_1)\sin(\Delta\theta_2)a_2 - 2r\sin(\alpha_1)\sin(\Delta\theta_2)a_2 \\
&+ 2p\cos(\alpha_1)\sin(\Delta\theta_1)\sin(\Delta\theta_2)a_2 - 2n\sin(\Delta\theta_3)a_2 \\
&+ 2\cos(\Delta\theta_2)a_1a_2 - 2rd_1 + 2o\cos(\alpha_1)\cos(\alpha_2)d_1 \\
&+ 2n\cos(\alpha_2)\cos(\Delta\theta_2)\cos(\Delta\theta_3)\sin(\alpha_1)d_1 \\
&+ 2n\cos(\alpha_1)\cos(\Delta\theta_3)\sin(\alpha_2)d_1 \\
&- 2o\cos(\Delta\theta_2)\sin(\alpha_1)\sin(\alpha_2)d_1 \\
&+ 2m\cos(\Delta\theta_3)\sin(\alpha_1)\sin(\Delta\theta_2)d_1 \\
&+ 2m\cos(\alpha_2)\cos(\Delta\theta_2)\sin(\alpha_1)\sin(\Delta\theta_3)d_1 \\
&+ 2m\cos(\alpha_1)\sin(\alpha_2)\sin(\Delta\theta_3)d_1 \\
&- 2n\sin(\alpha_1)\sin(\Delta\theta_2)\sin(\Delta\theta_3)d_1 + 2\sin(\alpha_1)\sin(\Delta\theta_2)a_2d_1 \\
&- 2r\cos(\alpha_1)d_2 + 2o\cos(\alpha_2)d_2 + 2q\cos(\Delta\theta_1)\sin(\alpha_1)d_2 \\
&+ 2n\cos(\Delta\theta_3)\sin(\alpha_2)d_2 - 2p\sin(\alpha_1)\sin(\Delta\theta_1)d_2 \\
&+ 2m\sin(\alpha_2)\sin(\Delta\theta_3)d_2 + 2\cos(\alpha_1)d_1d_2
\end{aligned}
\tag{3.63}
$$

Chapter 4

Design Applications

This chapter discusses the design methodology used to produce linkages to be used in specific applications. The motivation for the applications and the method for choosing the input functions is discussed.

4.1 Design Methodology

The same fundamental design methodology was applied to all of the mechanisms when searching for solutions that satisfied the desired designs. A specific solving algorithm was developed and applied to the RRR-2SS, RPR-2SS and PRP-2SS linkages.

The synthesis of the mechanisms followed the process is shown in Figure 4.1. It consists of three primary steps that are repeated as specified by the designer: i) values for the task, task positions, are randomly selected from within the tolerance zones around the precision points specified by the designer; ii) the design equations formulated and solved using these task values to identify candidate designs; then iii) each design is analyzed to evaluate its performance. Successful designs are saved. The tolerance zones are selected by the designer

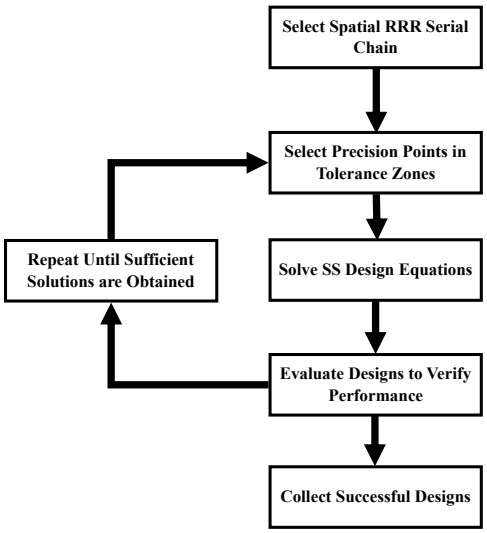


Figure 4.1: The process used to find successful solutions to the design equations involves adjustment of the task points within user defined tolerance zones. Iteration of this procedure produces a large number of design candidates.

as the deviation from the original the precision points that will still produce a mechanism that will still satisfy the original design constraints.

The successful designs then were ranked by the ratio of the lengths of the longest to shortest links, $\kappa = \frac{\text{longest link}}{\text{shortest link}}$. Smaller values of this parameter are considered more preferable for packaging, and designs with κ value greater than the designer's specified choice were eliminated. The remaining designs were sorted by the RMS error of their swing and pitch curves compared to the desired swing and pitch curves.

4.1.1 Solver Strategies for RRR-2SS, RPR-2SS and PRP-2SS

The two sets of design equations were solved separately from each other. The SS dyad which constrains the first two links of the of the spatial serial chain are solved first, the RRSS, RPSS and PRSS links for the RRR-2SS, RPR-2SS and PRP-2SS linkages, respectively. The original precision point is used in the design equations to check for an exact solution. If no solution is found then the task positions for the first two joints are randomly selected

from the tolerance zones and used in the design equations. This process is repeated until a solution to the design equations for the SS dyad constraint to the first two links of the spatial serial chain. The solution is then analyzed for branching and continuity defects.

Once a solution without defects is found for the first constraint, those inputs are used in the design equations for the second constraint. The task points for the third link are used in the design equations and randomized within its tolerance zones for a specified number of iterations. Typically 100 iterations were performed. Each solution found is checked for branching and continuity defects. If the solution is found to be defect free the solution is collected.

After the iterations were completed the solver returned to the outer loop and continued to iterate the task points for the first two links. This cycle is repeated until the desired number of iterations is achieved.

4.2 Flapping Wing Mechanism

4.2.1 Input Functions

The joint trajectories for the RRR chain that move this system as recommended by Yan and Taha [94] are giving by

$$\begin{aligned} \theta_1 =& t, \\ \theta_2 =& \frac{\pi}{2} - \frac{\pi}{3}\cos t, \\ \theta_3 =& \frac{\pi}{2} - \frac{\pi}{3}\sin t. \end{aligned} \tag{4.1}$$

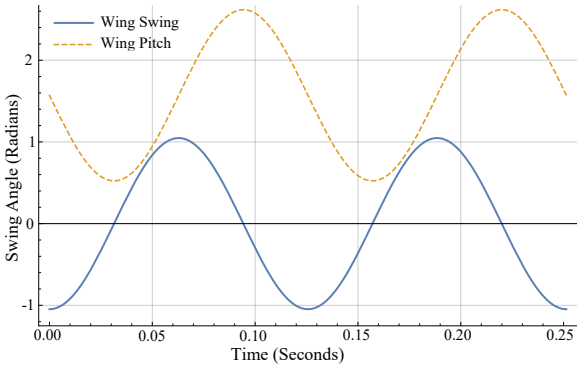


Figure 4.2: The wing swing and wing pitch functions.

as shown in Figure 4.2.

4.2.2 Planar Spatial Mechanism

4.2.2.1 Wing Sing and Wing Pitch Requirements

A study of flapping wing designs for micro air vehicles[5] shows that planar crank-rocker linkages with a passive position of the wing pitch provide effective wing performance. However, Yan and Taha [94] show that coordinated control of the wing pitch and wing swing movement improves the aerodynamics of a micro air vehicle.

They demonstrate effective aerodynamic performance with wing swing and wing pitch functions, $q(\theta)$ and $\phi(\theta)$, given by

$$q(\theta) = \frac{\pi}{2} - \frac{\pi}{3}\cos\theta, \quad \phi(\theta) = \frac{\pi}{2} - \frac{\pi}{3}\sin\theta, \tag{4.2}$$

where the driving angle is $\theta = \omega t$, and ω is the flapping frequency, Figure 4.2.

The precision points of θ and ϕ were chosen by applying Chebyshev Spacing on Equation 4.1. The calculated input crank angles(θ) are then substituted into the planar linkage equation,

see Eq 3.6, to obtain the wing swing angle $\bar{\gamma}$. The values of the tolerance zones for each precision point $\pm\delta\theta_i$ and $\pm\delta\phi_i$ are selected by the designer. The values are seen in Table 4.1.

	Wing Pitch Angle Requirements (Radians)						
i	1	2	3	4	5	6	7
θ_i	0	0.15	1.30	3.14	4.99	6.13	6.28
$\delta\theta_i$	0	$\pm$0.26	$\pm$0.26	$\pm$0.26	$\pm$0.26	$\pm$0.26	0
ϕ_i	1.57	1.41	0.56	1.57	2.58	1.73	1.57
$\delta\phi_i$	0	$\pm$0.26	$\pm$0.26	$\pm$0.26	$\pm$0.26	$\pm$0.26	0
$\bar{\gamma}_i$	0.52	0.54	1.34	2.62	1.41	0.55	0.52

Table 4.1: Table of initial parameter angles for seven precision points.

The planar mechanism was chosen to have a total wing swing angle $\sigma = \pi/3$ from $q(\theta)$ in Eq 3.5, a base length $r_1 = 5$in, and a transmission angle of 0.521 or 29.9 degrees. The corresponding values of r_2, r_3, and r_4 are seen in Table 4.2. These values are substituted into Eq 3.5 which gives $\gamma(\theta)$ a minimum at $\gamma(0.045) = 0.568$ degrees. Thus, $\theta_0 = -0.045$ or -2.6 degrees and $k_0 = 0.045$ or 2.6 degrees and the wing swing equation is

$$\begin{aligned}\bar{\gamma}(\Delta\theta) = &\tan^{-1}\frac{2.24 - 0.17\cos(\Delta\theta + 0.04)}{-0.17\sin(\Delta\theta + 0.04)} \\ &+ \cos^{-1}\left(\frac{4.33\cos(\Delta\theta + 0.04) - 0.45}{\sqrt{0.17\sin\Delta\theta - 3.88\cos\Delta\theta + 25.15}}\right) - 0.04. \end{aligned} \tag{4.3}$$

To align the planar and spatial linkages the input crank axis of the spatial linkage was placed such that $t = -5$in and the output crank axis such that $g = 0$. Knowing t, g, θ_i, ϕ_i and $\bar{\gamma}_i$ for $i = 1, \ldots, 7$ and solving Eq 3.13 for (u, v, w, x, y, z) gives the coordinates of $\mathbf{A}^1$ and $\mathbf{B}^1$ in the global frame W. The θ_i values were selected in the range of $\pm\delta\theta_i$ and the ϕ_i values were

Dimensions for Planar Linkage					
GN	**ND**	**DE**	**EG**	θ_0	k_0
5.00	0.39	4.99	0.45	-0.04	-0.04

Table 4.2: Solution for planar crank rocker for given input parameters

selected in the range of $\pm\delta\phi_i$. The $\bar{\gamma}_i$ values were recalculated using Eq 4.3 and the values of θ_i.

Five hundred variations produced 3582 solutions of which 29 met the design requirements. The 29 design candidates were sorted in ascending order by the link ratio κ, where $\kappa = \frac{Longest\ Link\ Length}{Shortest\ Link\ Length}$. The RMS error of ϵ between each of the top six results and desired wing pitch function of $\phi(\theta_k)$ given in Eq 4.1 was calculated using

$$\epsilon = \sqrt{\frac{\sum_{k=1}^{n}(\bar{\phi}(\theta_k) - \phi(\theta_k))}{n}}, \tag{4.4}$$

where $\bar{\phi}(\theta_k)$ is the wing pitch function from the solution, and n is the number of samples.

Design number 1 was selected because it has the lowest RMS error of the designs with a link ratio less than 10, see Table 4.3. Results with link ratios less than 10 improved the ease of manufacturing. Its associated inputs are in Table 4.4 and the output is plotted in Figure 4.3.

	Ranked Design Candidates							
			A Coordinates			**B** Coordinates		
	ϵ	κ	u	v	w	x	y	z
1	**0.89**	**2.86**	**2.25**	**0.57**	**−3.32**	**−1.16**	**−1.22**	**1.23**
2	1.39	2.92	7.64	-3.24	-4.31	2.73	0.12	1.67
3	1.29	3.28	-7.06	-4.66	-4.14	1.21	-2.64	0.22
4	1.37	3.42	-3.98	-1.87	-4.12	0.72	-1.72	0.47
5	1.35	12.9	-6.42	-0.52	-5.46	0.15	0.05	-0.64
6	1.03	16.6	-3.50	-0.02	-4.68	0.01	-0.19	0.31

Table 4.3: Table of coordinate solutions

The mechanism was modeled in SolidWorks see Figure 4.4, 4.5, and 4.6. Figure 4.7 compares the SolidWorks output, the generated configuration output, and the required wing pitch angle. The SolidWorks models shows some discrepancy due to minor adjustments to the

Selected Task Position for Solution 4							
θ	0.	0.40	1.17	3.35	5.24	6.09	6.28
ϕ	1.57	1.19	0.80	1.83	2.76	1.81	1.57
γ	0.52	0.65	1.24	2.58	1.18	0.56	0.52

Table 4.4: Table of of randomized input parameters

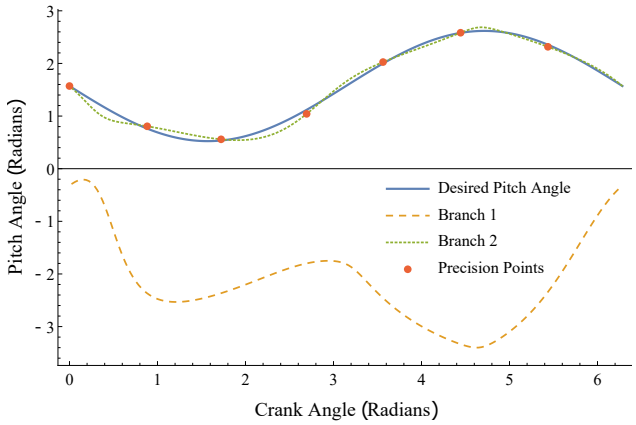


Figure 4.3: The Required Pitch Path vs Selected Pitch Path

structure that allow it to be physically constructed.

Solution 4: Dimensions for Spatial Linkage					
NA	**AB**	**BC**	**CG**	θ_s	ϕ_s
4.48	6.57	1.44	1.27	2.70	3.66

Table 4.5: Table of Link Lengths for the Spatial Mechanism

4.2.3 Spatial Six Bar Mechanism

In order to design the flapping wing mechanism, we start with the RRR serial chain defined in Table 3.2. These values were chosen by the designer to fit the workspace and packaging requirements. The RRR chain matches the scale of a hummingbird as seen Aerovironment's Nano Hummingbird (Keennon et al [46]) and the motor is oriented such that it fits within the body of the bird.

This serial chain will be installed in the micro air vehicle by mounting link L_1 to the body so the ground link F is rotated by a motor. This is shown in Figure 4.8 where the S-joints

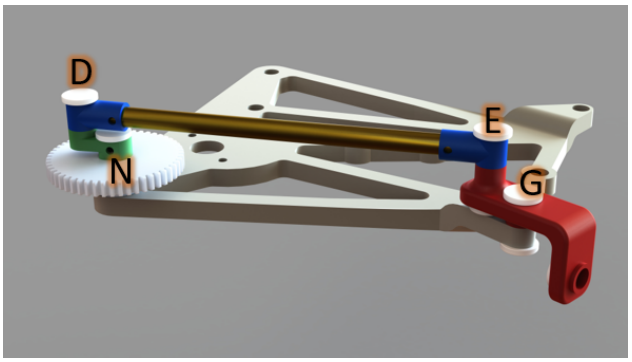


Figure 4.4: The Planar Mechanism **NGED**

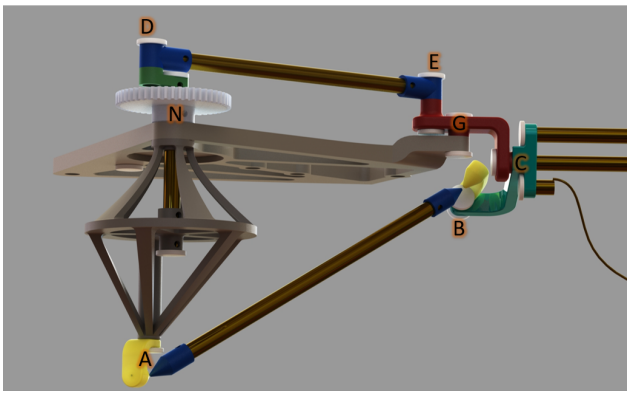


Figure 4.5: Planar Spatial Flapping Wing Mechanism

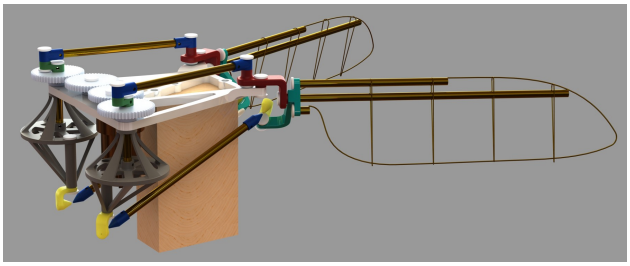


Figure 4.6: Solid Model of the Full Flapping Wing Assembly

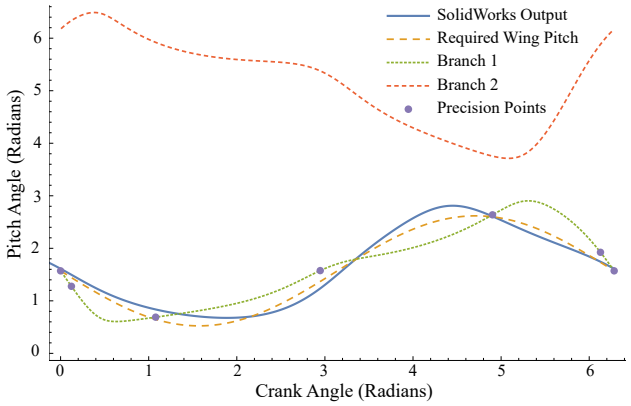


Figure 4.7: The SolidWorks Pitch Angle vs Required Pitch Path vs Selected Pitch Path

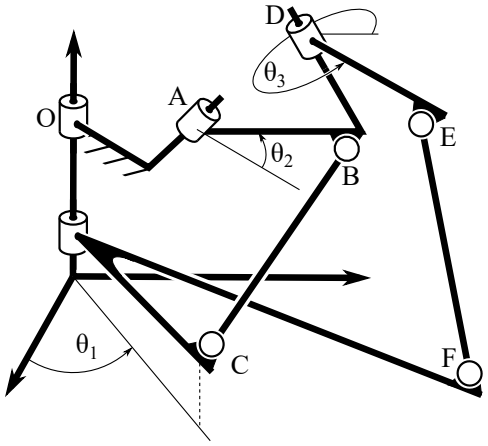


Figure 4.8: The link OA of the RRR-2SS linkage is held fixed so the interconnected cranks supporting the joints C and F simultaneously drive the links AD and DE.

Table 4.6: Seven precision points (radians) which define the center of the search zones

Point $\mathbf{q}_j$	θ_1	θ_2	θ_3
1	0.00	0.52	1.57
2	0.90	0.92	0.75
3	1.80	1.80	0.55
4	2.69	2.51	1.12
5	3.59	2.51	2.03
6	4.49	1.80	2.59
7	5.39	0.92	2.39

Table 4.7: The difference $\Delta\theta_i$ between the resulting input, swing and pitch angles and the desired function values at each of the precision points (radians).

Point $\mathbf{q}_j$	$\Delta\theta_1$	$\Delta\theta_2$	$\Delta\theta_3$
1	0.0	0.0	0.0
2	-0.014	0.073	0.054
3	-0.072	0.079	0.008
4	0.002	0.020	-0.075
5	-0.027	-0.030	0.001
6	-0.044	-0.041	-0.008
7	0.051	0.023	-0.074

C and **F** are mounted to interconnected cranks that simultaneously drive wing swing, link **AD**, and wing pitch, link **DE**.

The task points are shifted slightly in the design process from the original precision points shown in table 4.6. The difference between the selected task points and the original precision points that lie on the desired function curves are shown in Table 4.7, and plotted in Figure 3.7. The design equations are derived and solved in section 3.2.4.

For each of the 16 combinations of solutions for **BC** and **EF**, substitute the coordinates into equations (3.34) and (3.38) to evaluate the movement of the candidate design for the Flapping Wing Mechanism. Of these 16 candidates only the combinations of solutions 1 and 3 for **BC** and solution 3 for **EF** yielded linkages that moved smoothly through all of the task points.

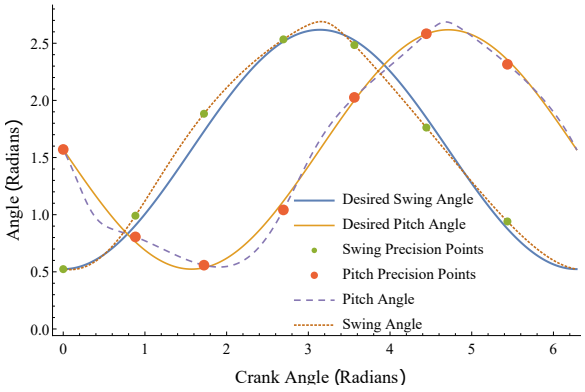


Figure 4.9: The movement of the swing control with link **BC** and pitch control with link **EF**.

The candidate selected for this mechanism combines solution 3 for **BC** and solution 3 for **EF** and is listed in Table 4.8. Solution 1 for **BC** was eliminated because the length of **BC** would be over 300cm for a micro air vehicle that is to have wings on the order of 30cm in length

Table 4.8: The links selected for the Flapping Wing Mechanism.

BC	u	v	w	x	y	z
	-1.54	0.06	2.65	-4.47	0.53	5.99
EF	p	q	r	m	n	o
	3.02	-1.06	-6.87	-2.05	3.39	2.53

The ability of the Flapping Wing Mechanism to drive the desired swing and pitch trajectories is demonstrated in Figure 4.9.

A geometric model of the Flapping Wing Mechanism is shown in Figure 4.10. The wing swing and wing pitch are driven by an actuator and gear train system that connect cranks at the top and bottom of the mechanism. Figure 4.11 is a view from the opposite side that shows that the two cranks move together to simultaneously drive the swing and pitch of the wing. A rear view of the mechanism shows the perpendicular wing swing and wing pitch axes at the center of the device.

The tolerance zones were for this problem ±5 degrees (±0.087 radians) around the precision

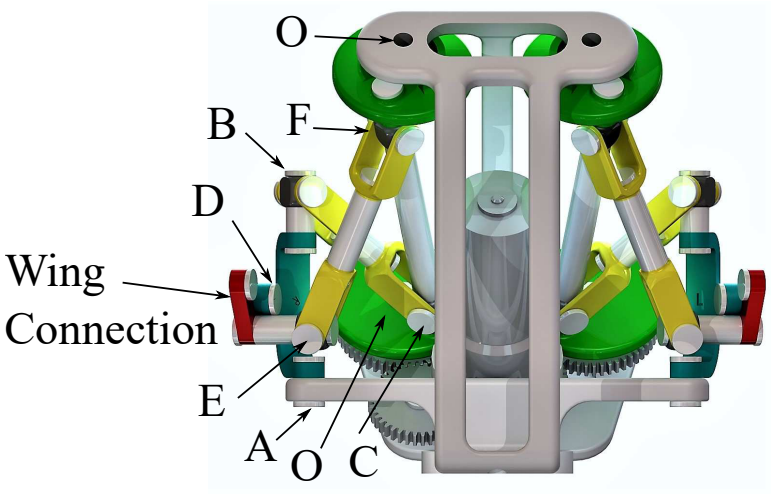


Figure 4.10: Geometric model of the RRR-2SS Flapping Wing Mechanism. The wing is attached to link **DE**. The pitch of the wing is controlled by link **EF**, which connects the lower gear and the wing. Link **ABD**, controls the swing of the wing and connects the wing, the structural frame, and link **BC**.

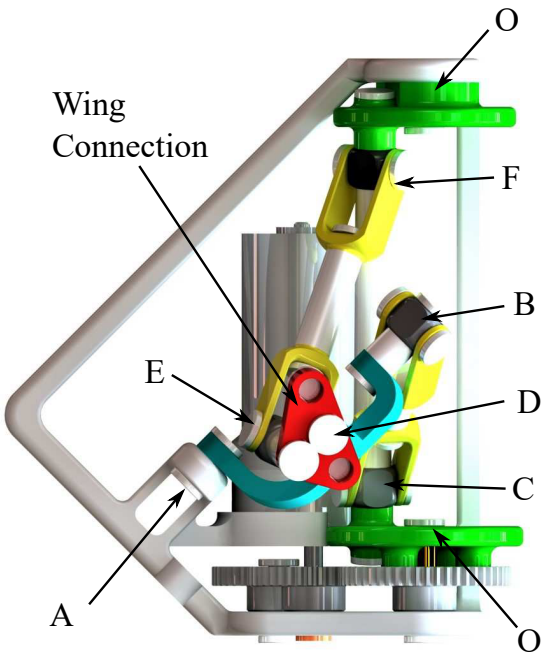


Figure 4.11: A motor drives links **OC** and **OF**, at the same velocity via a simple gear train. Link **OC** and **OF** control swing and pitch, respectively.

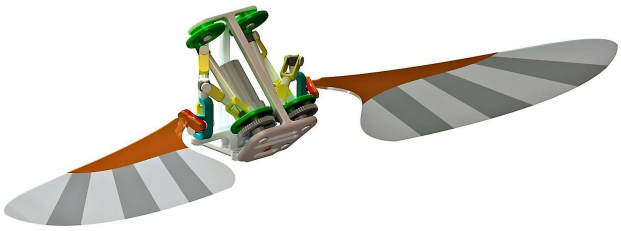


Figure 4.12: The geometric model with the wings attached showing the rear of the model.

points given in Table 4.6. The process was iterated 1100 times to obtain 91 successful designs.

The successful designs then were ranked by the ratio of the lengths of the longest to shortest links, $\kappa = \frac{\text{longest link}}{\text{shortest link}}$. Smaller values of this parameter are considered more preferable for packaging, and designs with $\kappa > 10$ were eliminated. The remaining designs were sorted by the RMS error of their swing and pitch curves compared to the desired swing and pitch curves.

Of the 91 successful designs, 18 designs had a link length ratio of less than 10. The design with the lowest RMS error was selected with $\kappa = 8.200$ and an RMS error of 0.159. The SS dyads of the selected design are shown Table 4.8.

This design uses a motor to drive links **OC** and **OF**, which are connected by a simple gear train, to drive them at the same velocity as seen in Figure 4.11. The wing is attached to link **DE**, shown in Figure 4.10. The pitch of the wing is controlled by link **EF**, which connects the lower gear and the wing, shown in Figure 4.10. Link **ABD**, shown in Figure 4.10, controls the swing of the wing and connects the wing, the structural frame and link **BC**. Link **BC**, shown in Figure 4.12, connects the upper gear and link **ABD**.

4.2.3.1 Comparison with Existing Micro Air Vehicles

The Aerovironment Nano Hummingbird (Keennon et al [46] uses a four-bar linkage and cable driven to produce the wing swing movement, and relies on flexure of the wing due to

aerodynamic drag to produce the pitch. Sheshadri et al [76] and Conn et al [19] use a pair of phased four-bar linkages to control swing and pitch for each wing. Plecnik [64] use four six-bar linkages to control the four joints of a serial chain that models the wing gait of a black-billed magpie.

The flapping wing mechanism presented here provides control of both swing and pitch with a six-bar linkage for each wing. This has many fewer parts than a pair of phased four-bar linkages, and is only slightly more complicated than Aerovironments wing mechanism.

4.3 Self Cleaning Valves

Two different mechanisms the RPR-2SS and the PRP-2SS are used in the self cleaning valve application. The first actuates the valve through a rotating pressure plate, while the second actuates the valve with a piston.

4.3.1 RPR-2SS Spatial Six Bar Mechanism

4.3.1.1 Input Functions

For this design, the trajectory of the slide $d_2(t)$ was chosen to be a cubic function that provides a decreasing rate of slide relative to the input rotation. The sealing rotation $\theta_3(t)$ is a linear function that stops when the slide reaches 1.75 in. This allows the plunger to slot

into the port closing the valve. These joint trajectories are given by

$$\begin{aligned}
\theta_1 =& t, \\
d_2 =& -1.25 + 6.37t - 4.56t^2 + 1.16t^3, \\
\theta_3 =& \begin{cases} -\frac{15}{20}t + 0.65 & t < 0.87 \\ 0 & t \geq 0.87. \end{cases}
\end{aligned} \tag{4.5}$$

The sealing rotation must have zero rotation during the pure plunging phase or else there will be a physical collision between the plungers and the grill they are cleaning. The slide was chosen to have a move quickly during the sealing rotation phase and more slowly during the pure plunging phase. This should give a better mechanical advantage during the plunging phase which will allow for a great plunging force.

4.3.1.2 Spatial Serial Chain Selection

Table 4.9: Denavit-Hartenberg table for the RPR serial chain.

Joint	θ_i	d_i	α_i	a_i
1	θ_1	0.	0.79	1.
2	0.	d_2	0.	0.
3	θ_3	1.	–	–

Table 4.10: Six task points (radians and inches) of the RPR spatial chain chosen from the joint trajectories to design the SS dyads

Point $\mathbf{q}_j$	θ_1	d_2	θ_3
1	1.57	2.03	0.
2	1.28	1.87	0.
3	0.98	1.73	0.
4	0.86	1.65	0.
5	0.79	1.54	0.01
6	0.54	1.03	0.27

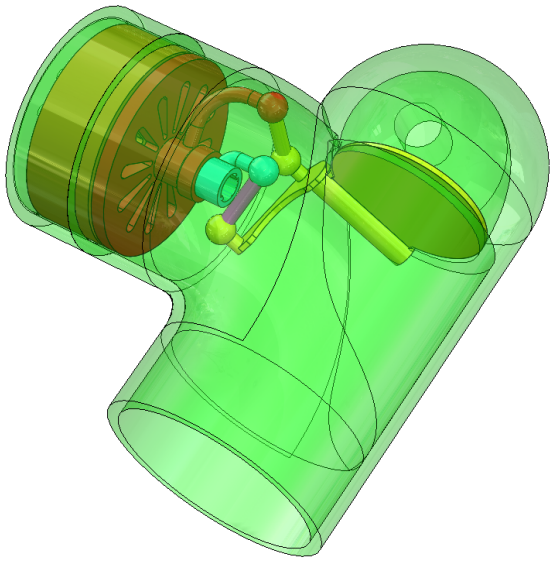


Figure 4.13: The mechanism is mounted inside a T pipe that has one end capped. Flow enters at the bottom left and exits at the top left.

4.3.2 Soil Conditioning Valve Mechanism

Our soil conditioning valve mechanism fits within the 4in diameter pipe that leads to the 3in diameter port, Figure 4.13. This packaging limits the physical dimensions of the RPR spatial chain, which are listed in Table 4.9. The RPR mechanism uses the prismatic joint to control the plunger, and final revolute joint to control the sealing operation, and the system is actuated by the first revolute joint, Figure 4.14. In this figure fluid flows into the pipe from the bottom left and exits from the top left. Fluid does not flow around the circular pressure plate. The RPR serial chain is to be mounted to the pipe as Link L_1, so the ground link F is rotated by the fluid pressure within the pipe similar to a swing check valve. Closure of the valve is actuated by a spring when fluid pressure is relieved.

4.3.2.1 Chosen Mechanism

Figure 4.15 shows the S-joints **C** and **F** are connected to a single crank that drives the slide of link **ABD** and the rotation of link **DE**. The slide of the plunger and the sealing rotations

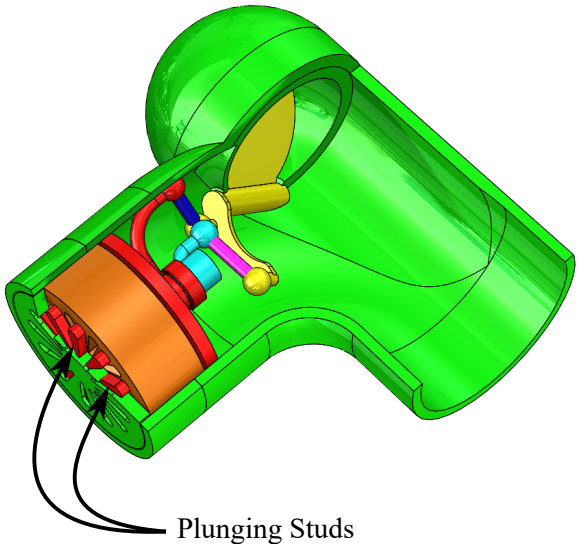


Figure 4.14: When the valve closes, it also plunges grate as a self cleaning action. The studs of the plunger can be seen during the extension phase.

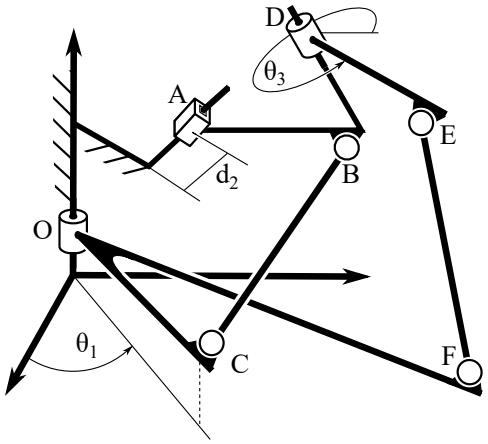


Figure 4.15: The link OA of the RPR-2SS linkage is held fixed. Points C and F are connected to a single crank which drive links ABD and DE.

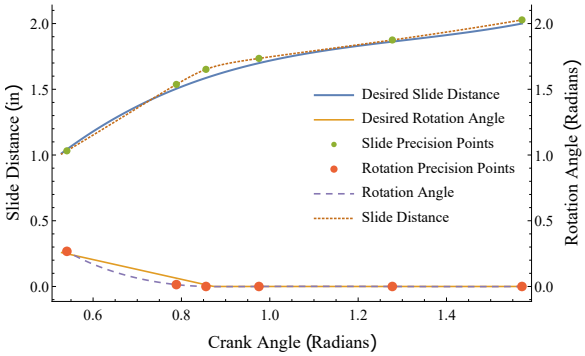


Figure 4.16: Comparison Between the desired curve and the curve generated by the design.

are specified by the designer.

The design equations for the selected task points are shown in section 3.3.3.

Substitute the coordinates of each of the four combinations of solutions to **BC** and **EF** into equations 3.45 and 3.47. From the 4 candidates the combinations of solution 1 for **BC** and solutions 2 and 4 for **EF** created linkages that moved smoothly through all the task points.

The selected mechanism combines solution 1 for **BC** and solution 4 for **EF**. Solution 2 for **EF** was eliminated because the link lengths would have been too long to fit in the desired pipe. The performance of the mechanism to control the slide and sealing rotation are compared to the desired curves in Figure 4.16.

The valve can be seen opening and in the closed position in Figures 4.17 and 4.18. The valve opens when fluid pressurizes the pipe rotating the circular pressure plate. The rotation of the pressure plate drives the slide of the plunger and the sealing rotation. When fluid pressure decreases a spring behind the pressure plate will reset the valve to a closed position.

The mechanism can be seen in detail in Figure 4.19. The sealing drum is shown to be transparent and the pipe is hidden. Pressure plate is connected to the crank link **OCF**. The plunger is connected to link **ABD**. The sealing rotation is controlled by link **DE**. Link **BC** connects the crank and the plunger. Link **EF** which connects the sealing rotation link to

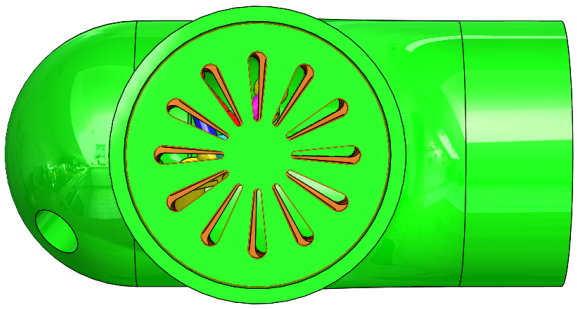


Figure 4.17: The valve is in the process of opening.

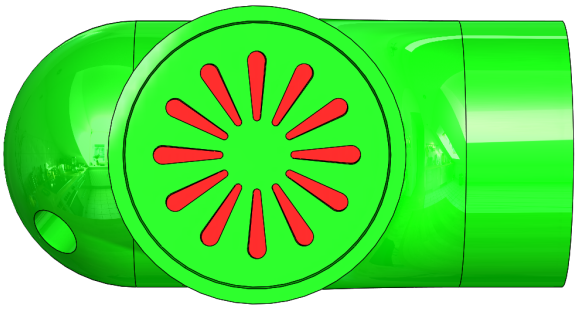


Figure 4.18: The fluid pressure inside of the pipe has decreased below the desired thresh hold, so the mechanism cleans and seals the port.

the crank.

4.3.3 Design Process

The design process of Soil Conditioning Valve Mechanism follows a similar design process as Wang et al. [90] shown in Figure 4.1. First the designer chooses the functions of the slide and sealing rotation and chooses task points on those curves within a specified tolerance zone. The design equations for the slide and sealing rotation are solved to find design candidates. The design candidates are then analyses to confirm its performance.

The tolerance zones used are detailed in Table 4.11. The zones with no allowable deviation were points that the designer could not have deviate. For example, the task points 1, 2, 3 and 4 for θ_3 are zero with no allowable deviation are because the mechanism must not rotate while plunging the port. However, the plunging distance and input crank rotation are not

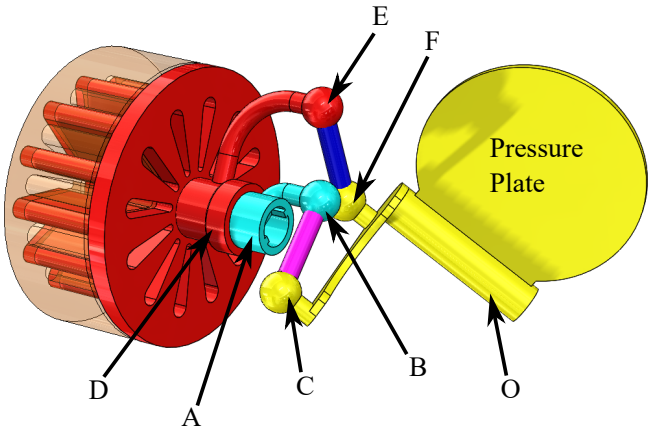


Figure 4.19: This figure shows mechanism with the sealing drum transparent and the pipe hidden. The plunging studs are now visible through the sealing drum.

as constrained, thus can be varied.

The design process was iterated 2600 times resulting in 459 successful designs. Each iteration the design equations are solved using new task points chosen at random within the tolerance zones.

The algorithm was run using Mathematica 10.3 on a workstation with dual 8 core Xeon E5-2620 v4 CPUs running at 2.1 GHz using 16 total threads. The average time of 0.069 seconds per iteration was found by running the algorithm 11 additional times varying the number of iterations from 2500 to 5200.

The successful designs are sorted in ascending order of the ratio, κ, of the longest to shortest link lengths. Designs with a link length ratio $\kappa > 10$ were eliminated, leaving 185 designs. The remaining designs were sorted by the RMS error of the plunge slide and sealing rotations versus their desired movements defined by Equation 4.5. Geometric models of the designs are made and animated starting from the designs with the lowest RMS error. The animation allows the designer to determine whether the linkage stays within the desired envelope, the valve body, throughout its movement. The animation also helps the designer determine the feasibility of avoiding interference when physically constructing the linkage. The designer

then choses the mechanism that best fits both the physical constraints and the desired performance.

Table 4.11: The specified tolerance zones from which the task points are chosen in each iteration.

Point $\mathbf{q}_j$	θ_1	d_2	θ_3
1	1.57	2.00 ± 0.05	0.00
2	1.31 ± 0.09	1.88 ± 0.05	0.00
3	1.05 ± 0.09	1.75 ± 0.05	0.00
4	0.87 ± 0.09	1.60 ± 0.05	0.00
5	0.79 ± 0.09	1.50 ± 0.05	0.07 ± 0.09
6	0.52 ± 0.09	1.00 ± 0.05	0.26 ± 0.09

4.3.4 PRP-2SS Spatial Six Bar Mechanism

The application of the PRP-2SS spatial mechanism for the self cleaning valve is inspired by the idea of using a prismatic joint to actuate the valve instead of a revolute joint. This will allow for a piston action rather than a rotation. The piston will be easier to manufacture as well as seal inside of pipe. The dimensions and performance of the mechanism is presented in this section, but a three dimensional model was not made.

4.3.4.1 Input Functions

For this design, the trajectory of the slide $d_3(t)$ was chosen to be a cubic function that provides a decreasing rate of slide relative to the input rotation. The sealing rotation $\theta_2(t)$ is a linear function that stops when the slide reaches 1.75 in. This allows the plunger to slot

into the port closing the valve. These joint trajectories are given by

$$
\begin{aligned}
d_1 =& t, \\
d_3 =& -1.59 + 7.87x - 6.31x^2 + 1.77x^3, \\
\theta_2 =& \begin{cases} -\frac{15}{20}t + 0.65 & t < 0.87 \\ 0 & t \geq 0.87. \end{cases}
\end{aligned} \tag{4.6}
$$

The sealing rotation must have very little rotation during the pure plunging phase or else there will be a physical collision between the plungers and the grill they are cleaning. The slide was chosen to have a move quickly during the sealing rotation phase and more slowly during the pure plunging phase. This should give a better mechanical advantage during the plunging phase which will allow for a great plunging force.

4.3.4.2 Spatial Serial Chain and Task Position Selection

The spatial serial chain and the associated task positions are shown in Tables 4.12, 4.13, and 4.14.

Table 4.12: Denavit-Hartenberg table for the PRP serial chain for the Valve.

Joint	θ_i	d_i	α_i	a_i
1	0	d_1	0.79	1.
2	θ_2	1	0.	0.
3	0	d_3	–	–

4.3.4.3 Synthesis Equations

The six task points selected from these trajectories are given in Table 4.14. The free parameters were chosen such that the joints **C** and **F** are fixed to planes $w = 0$ and $r = 0$,

Table 4.13: Six precision points (radians and inches) and tolerance zones of the PRP spatial chain for the valve.

Point $\mathbf{q}_j$	d_1	θ_2	d_3
1	1.50 ± 0.05	0.00 ± 0.01	2.00 ± 0.05
2	1.30 ± 0.05	0.00 ± 0.01	1.87 ± 0.05
3	1.00 ± 0.05	0.00 ± 0.01	1.75 ± 0.05
4	0.90 ± 0.05	0.00 ± 0.26	1.68 ± 0.05
5	0.80 ± 0.05	0.05 ± 0.26	1.58 ± 0.05
6	0.50 ± 0.05	0.28 ± 0.26	1.00 ± 0.05

Table 4.14: Six selected task points (radians and inches) of the PRP spatial chain.

Point $\mathbf{q}_j$	d_1	θ_2	d_3
1	1.47	0.	1.99
2	1.32	-0.02	1.9
3	1.05	0.	1.71
4	0.89	0.08	1.66
5	0.75	0.16	1.58
6	0.5	0.31	1.05

respectively.

The task points are substituted into the design equations $\mathcal{A}_i, j = 2, \ldots, 6$ for the link $\mathbf{BC}$,

which results in the following,

$$
\begin{aligned}
\mathcal{A}_2: \quad & vy(1.1*10^{-4}) + vz(1.1*10^{-4}) + ux(2.2*10^{-4}) - 0.02uy - 0.02uz \\
& + 0.03u + 0.02vx - 0.02v - 0.03x + 0.02y - 0.27z + 0.018 = 0 \\
\mathcal{A}_3: \quad & vx(-3.7*10^{-3}) + y(-3.7*10^{-3}) + uy(3.7*10^{-3}) + uz(3.7*10^{-3}) \\
& + v(3.7*10^{-3}) + x(3.9*10^{-3}) + u(-5.4*10^{-3}) + vy(3.4*10^{-6}) \\
& + vz(3.4*10^{-6}) + ux(6.9*10^{-6}) - 0.84z + 0.18 = 0 \\
\mathcal{A}_4: \quad & vy(3.2*10^{-3}) + vz(3.2*10^{-3}) + ux(6.4*10^{-3}) + 0.11uy + 0.11uz \\
& - 0.17u - 0.11vx + 0.11v + 0.09x - 0.12y - 1.28z + 0.41 = 0 \\
\mathcal{A}_5: \quad & 0.03ux + 0.22uy + 0.22uz - 0.35u - 0.22vx + 0.01vy + 0.01vz + 0.2v \\
& + 0.14x - 0.23y - 1.66z + 0.70 = 0 \\
\mathcal{A}_6: \quad & 0.1ux + 0.44uy + 0.44uz - 0.74u - 0.44vx + 0.05vy + 0.05vz + 0.36v \\
& + 0.12x - 0.46y - 2.39z + 1.48 = 0
\end{aligned}
\tag{4.7}
$$

The solutions to the design equations (4.7) for links **BC** are shown in table 4.15 The task

Table 4.15: Real-valued solutions to design equations $\mathcal{A}_j$ for **BC**

	u	v	w	x	y	z
1	$1.99*10^{18}$	$5.35*10^{17}$	0	0.	0.	0.
2	$-1.2*10^{15}$	$-2.61*10^{14}$	0	1.	1.28	0.189
3	3.77	1.35	0	-2.96	0.52	0.206

points are substituted into the design equations $\mathcal{B}_i, j = 2, \ldots, 6$ for the link **EF**, which results

in the following,

$$
\begin{aligned}
\mathcal{B}_2: \quad & nq(1.1*10^{-4}) + nr(1.1*10^{-4}) + oq(1.1*10^{-4}) + or(1.1*10^{-4}) \\
& + mp(2.2*10^{-4}) + 0.02mq + 0.02mr - 0.03m - 0.02np + 0.15n \\
& - 0.02op - 0.4o + 0.03p - 0.15q + 0.4r + 0.045 = 0 \\
\mathcal{B}_3: \quad & mq(-3.7*10^{-3}) + mr(-3.7*10^{-3}) + np(3.7*10^{-3}) + op(3.7*10^{-3}) \\
& + m(3.9*10^{-3}) + p(-5.4*10^{-3}) + nq(3.4*10^{-6}) + nr(3.4*10^{-6}) \\
& + oq(3.4*10^{-6}) + or(3.4*10^{-6}) + mp(6.9*10^{-6}) + 0.4n - 1.24o - 0.4q + 1.24r + 0.46 = 0 \\
\mathcal{B}_4: \quad & nq(3.2*10^{-3}) + nr(3.2*10^{-3}) + oq(3.2*10^{-3}) + or(3.2*10^{-3}) \\
& + mp(6.4*10^{-3}) - 0.11mq - 0.11mr + 0.09m + 0.11np + 0.35n + 0.11op \\
& - 1.75o - 0.17p - 0.36q + 1.74r + 0.79 = 0 \\
\mathcal{B}_5: \quad & 0.03mp - 0.22mq - 0.22mr + 0.14m + 0.22np + 0.01nq + 0.01nr + 0.34n \\
& + 0.22op + 0.01oq + 0.01or - 2.23o - 0.35p - 0.37q + 2.21r + 1.28 = 0 \\
\mathcal{B}_6: \quad & 0.1mp - 0.44mq - 0.44mr + 0.12m + 0.44np + 0.05nq + 0.05nr + 0.87n \\
& + 0.44op + 0.05oq + 0.05or - 3.72o - 0.74p - 0.97q + 3.62r + 3.66 = 0
\end{aligned}
\tag{4.8}
$$

The solutions to the design equations (4.8) for links **EF** are shown in table 4.16.

Table 4.16: Real-valued solutions to design equations $\mathcal{B}_j$ for **EF**

	p	q	r	m	n	o
1	$1.20*10^{29}$	$7.57*10^{14}$	0.	1.	$1.79*10^{14}$	$-1.79*10^{14}$
2	$-1.91*10^{14}$	5.53	0.	1.	2.2	-0.74
3	2.38	-1.71	0.	3.4	-5.2	-0.79

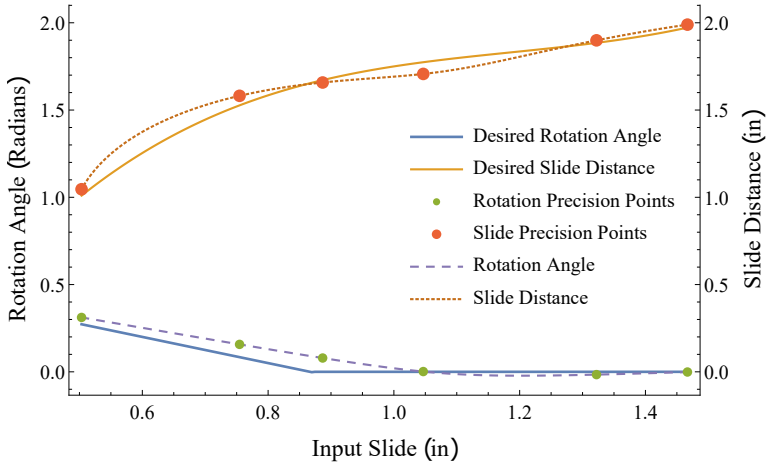


Figure 4.20: The performance of the valve compared to the desired curves is shown.

4.3.4.4 Chosen Mechanism

The selected links are shown in table 4.17. These performance of this linkage is shown in Figure 4.20 The design process was iterated 42,967 times and obtained 20,373 successful

Table 4.17: The links selected for the PRP-2SS Valve Mechanism.

BC	u	v	w	x	y	z
	3.77	1.35	0.	-2.96	0.52	0.21
EF	p	q	r	m	n	o
	2.38	-1.71	0.	3.4	-5.2	-0.79

designs. The algorithm was run using Mathematica 11.2 on a workstation with dual 8 core Xeon E5-2620 v4 CPUs running at 2.1 GHz using 16 total threads. The successful designs are sorted in ascending order of the ratio, κ, of the longest to shortest link lengths in the initial position. The upper limit for allowable link length ratios of 5 was chosen to generate compact linkages. The 4 designs in which $\kappa < 5$ were sorted in ascending order by the RMS error between the front and rear leg movements versus the respective desired movements defined by Equation 4.6. Finally the mechanism with links shown in table 4.17 was selected.

4.4 Flying Squirrel

The jumping of a flying squirrel consists of several distinct movements of their limbs and body. The complex jumping movement has been simplified to three key movements, the rapid extension of the rear legs, and the rotation of the front legs to propel the squirrel forwards and upwards and then the extension of the arms outwards to stretch the patagium, gliding flaps. This mechanism is designed to be actuated by a single spring which will push the rear prismatic joint backwards acting as the rear legs. This movement will then cause the rotation of the front legs as well as an extension of the patagium.

4.4.1 PRP-2SS Spatial Six Bar Mechanism

4.4.1.1 Input Functions

The functions were defined by the position of the patagium extension to the position of each of the joints. For this design, the trajectory of the rotation of the front arms $\theta_2(t)$ was chosen to be a quartic function that provides an initial acceleration, a slow down then final acceleration. For the slide extension of the rear leg extension $d_3(t)$ the trajectory was chosen to be a cubic function that initially allows for a rapid extension of the rear legs and slow extension of the patagium, followed by a faster extension of the patagium and slower extension of the rear legs. These joint trajectories are given by

$$\begin{aligned} d_1 =& t, \\ \theta_2 =& -2.70 + 13.32t - 8.69t^2 + 2.29t^3 - 0.22t^4, \\ d_3 =& 18.90 - 14.46t + 4.61t^2 - 0.49t^3 \end{aligned} \tag{4.9}$$

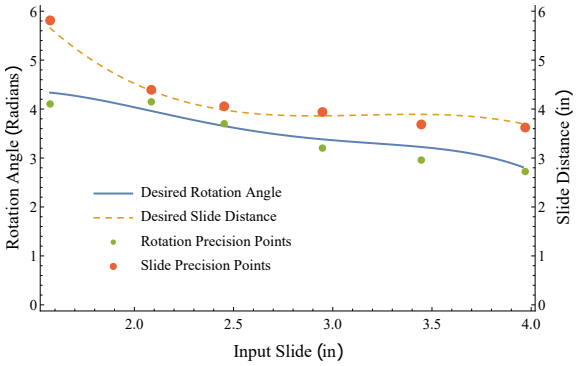


Figure 4.21: The desired slide and rotation function from Equations 4.9 and the respective task points are shown.

4.4.1.2 Spatial Serial Chain Selection

Table 4.18: Denavit-Hartenberg table for the PRP serial chain.

Joint	θ_i	d_i	α_i	a_i
1	-0.768	$d1$	-2.802	0.347
2	θ_2	2.025	1.830	1.105
3	0.	$d3$	0	0

Table 4.19: Six task points (radians and inches) of the PRP spatial chain chosen from the joint trajectories to design the SS dyads

Point $\mathbf{q}_j$	d_1	θ_2	d_3
1	3.93	2.85	3.72
2	3.5	3.2	3.89
3	3.	3.36	3.86
4	2.5	3.62	3.95
5	2.	4.04	4.52
6	1.5	4.36	5.94

4.4.1.3 Chosen Mechanism

The serial chain and a set of task positions that gives successful solutions are substituted into the design equations and solved in section 3.4.3. The solutions from those design equations

Table 4.20: The links selected for the Jumping Mechanism.

BC	u	v	w	x	y	z
	0.80	0.58	0	-2.30	2.70	0.63
EF	p	q	r	m	n	o
	-1.70	0.83	0	1.02	0.23	-0.67

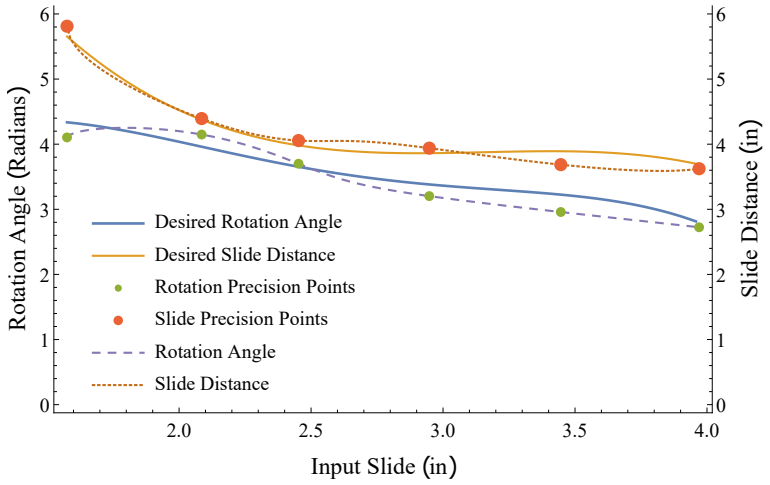


Figure 4.22: The movement of front leg rotation with link **BC** and the slide that extends the patagium with link **EF**.

are used to create the jump-gliding mechanism.

Substitute the coordinates of each of the four combinations of solutions to **BC** and **EF** into equations 3.53 and 3.55. From the three candidates the combinations of solution 1 for **BC** and solution 2 for **EF** created linkages that moved smoothly through all the task points.

The selected mechanism combines solution 1 for **BC** and solution 2 for **EF**. The mechanism's performance controlling the front and rear leg movements in comparison to the desired curves can be seen in 4.22.

The mechanism is shown in Figure 4.23. This assembly shows only the left legs of the squirrel. The mirror of the linkage will be added to control the right side. In this application link **ABD** is held fixed. Joint **D** is actuated by a spring which pushes the rear leg backwards propelling the mechanism. Link **OA** controls the rotation of the front leg which assists in the jump. Link **OC** controls the extension of the patagium. Links **BC** and **EF** are the SS

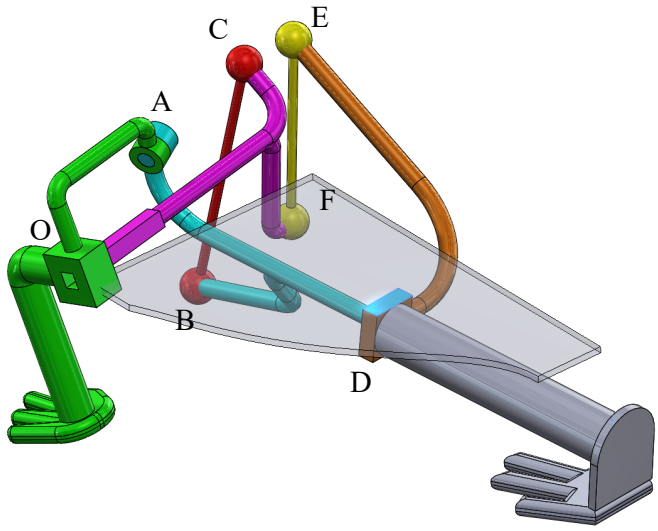


Figure 4.23: The mechanism is labeled with the corresponding joints. In this configuration Link ABD is held fixed.

dyads.

The jumping mechanism can be seen in three positions. Figure 4.24 shows the initial position the rear legs and patagium extension joints are fully retracted and the front legs are down. In this position the spring that drive the rear leg joints is fully compressed.

Figure 4.25 shows moments after the spring has been released. The rear legs have extended and the front legs have rotated backwards. The patagium has begun to deploy.

Figure 4.26 show the final position in the air with the patagium fully extended, the rear legs fully extended backwards and the front legs are parallel to the body and rear legs.

The tolerance zones used are detailed in Table 4.21. The rear leg extension d_3 has a larger tolerance zone than the patagium's slide since the travel is further. The angle of the front leg is given a tolerance of 15 degrees. These tolerances were given to allow for a wide variety of solutions.

The design process was iterated 32000 times resulting in 7856 successful designs. Each iteration the design equations are solved using new task points chosen at random within the

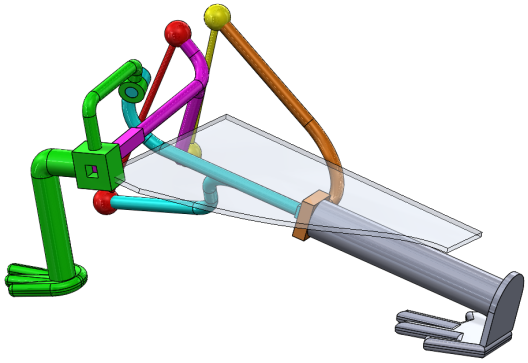


Figure 4.24: The starting position for the jumping mechanism with the spring, not shown, in full compression.

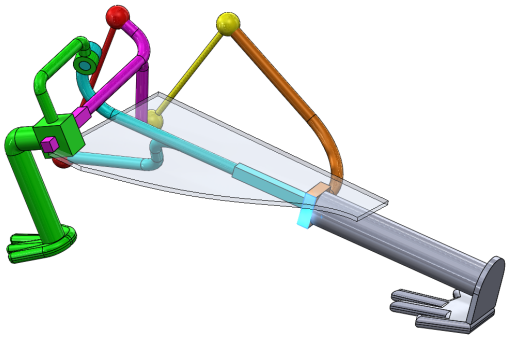


Figure 4.25: The spring has released and the rear leg has begun to extend. The patagium has begun to extend.

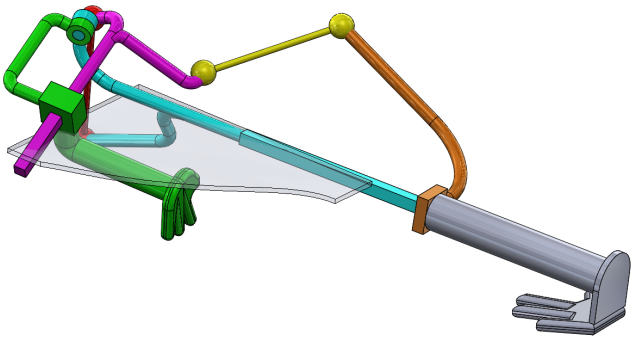


Figure 4.26: The rear leg has fully extended and the front legs have rotated to be parallel with the body. The patagium has fully extended for gliding.

tolerance zones.

The algorithm was run using Mathematica 11.2 on a workstation with dual 8 core Xeon E5-2620 v4 CPUs running at 2.1 GHz using 16 total threads. The average time of 0.022 seconds per iteration.

The successful designs are sorted in ascending order of the ratio, κ, of the longest to shortest link lengths in the initial position. The upper limit for allowable link length ratios of 3 was chosen to generate compact linkages. The 132 designs in which $\kappa < 3$ were sorted in ascending order by the RMS error between the front and rear leg movements versus the respective desired movements defined by Equation 3.49. The designs are modeled using 3D CAD software and animated. The animation serves to show the volume that the linkage passes through during its movement as well as interferences and other issues that would make physical construction significantly more difficult. The designer then weighs how well the linkage fits both the physical constrains and the desired performance and chooses a linkage.

Table 4.21: The specified tolerance zones from which the task points are chosen in each iteration.

Point $\mathbf{q}_j$	d_1	θ_2	d_3
1	3.93 ± 0.10	2.85 ± 0.26	3.72 ± 0.50
2	3.50 ± 0.10	3.20 ± 0.26	3.89 ± 0.50
3	3.00 ± 0.10	3.36 ± 0.15	3.86 ± 0.50
4	2.50 ± 0.10	3.62 ± 0.26	3.95 ± 0.50
5	2.00 ± 0.10	4.04 ± 0.26	4.52 ± 0.50
6	1.50 ± 0.10	4.36 ± 0.26	5.94 ± 0.09

4.5 Summary

The design applications mentioned here show the ability of this synthesis method to produce useful mechanism designs for a variety of applications ranging from biomimetic to specific

mechanical applications. By changing the dimensions of the spatial serial chain and the input functions these linkages can be used in a wide variety of applications limited only by the imagination of the designer. In the design methodology used above only the functions were randomized during the search for mechanisms that would satisfy the design requirements, however, it is also possible to establish a tolerance zone for the spatial serial chain to further expand the search area. These design examples have shown the flexibility of this design methodology to find useful linkages for a wide variety of applications.

Chapter 5

Manufacture

This chapter covers the manufacturing methods used to build the physical models as well as specific considerations and challenges that were discovered during the construction of these models. The specific pieces used to make the flapping wing mechanisms are shown in detail and the design iterations are described.

5.1 Conversion to Buildable Model

The geometric models created in Solidworks needed significant modification to be able to be physically constructed. All interferences needed to be removed, so the geometry of the linkages needed to be adjusted. As long as the relative positions of the joints do not change the link connecting them can have any geometry. This property allows for almost any linkage to be configured in such a way that interference is avoided. However other factors must be taken into account to ensure that the linkage can successfully move through the desired trajectory. If the linkage passes close to a singularity then the compliance in the linkage may allow the mechanism to switch to another branch, which would then may produce the

wrong movement. The compliance of the mechanism is very important to consider because kinematic synthesis theory in general assumes infinitely rigid linkages. Observations seem to have shown that as long as the branches are sufficiently far apart that the compliance does not allow for the mechanisms to cross branches then the mechanisms are able to still complete the motions satisfactorily even with significant deflections. However, this behavior has not been quantified.

5.2 Joint Fabrication

The fabrication of the joints provided challenges. The spherical joints were particularly difficult as standard ball joints did not have a sufficient range of motion. A compliant joint using wire rope and a single revolute joints provided the three degrees of freedom needed for a spherical joint, see Figure 5.1. The wire rope can give the three degrees of freedom of a spherical joint, however the wire rope will get too twisted so the additional revolute joint was added to avoid twisting of the rope. The cones used in the constructed model were 3D printed using FDM with ABS as the material. These pieces required some post machining, for example the hole in for the wire had to drilled and the hole for the revolute joint needed to be reamed.

Another method of creating a spherical-spherical dyad link with is to use two universal joints with an additional allowable rotation between the two. The spherical joints built using this method were constructed in several iterations. Initially the joints were constructed using a 3D printer using ABS filament, however the parts were found not to be able to handle the dynamic loads of the mechanism with out becoming too bulky. So the joints were then machined out of aluminum, see Figure 5.2.

Revolute joints were constructed using 3D printed parts whose holes were reamed for smooth-

Figure 5.1: The Wire Rope Compliant Spherical Joint

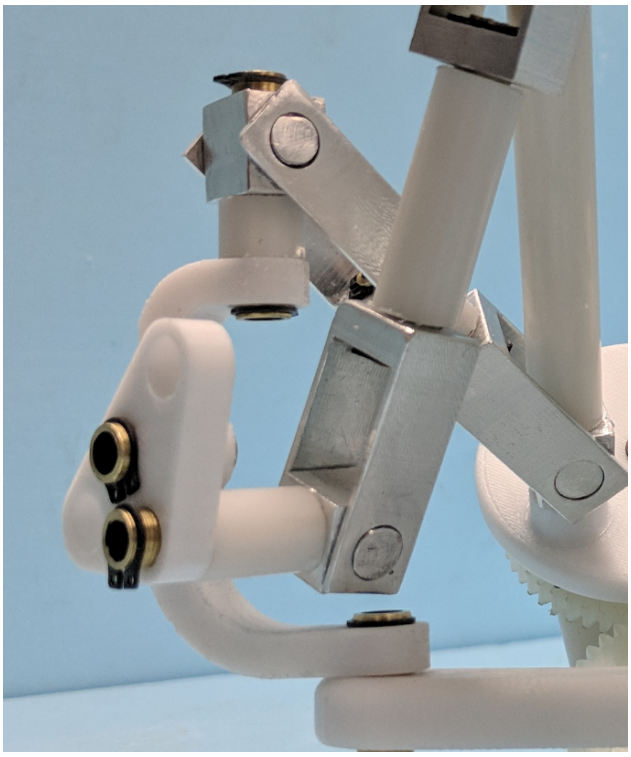


Figure 5.2: Two Aluminum Universal-Revolute Spherical Joint

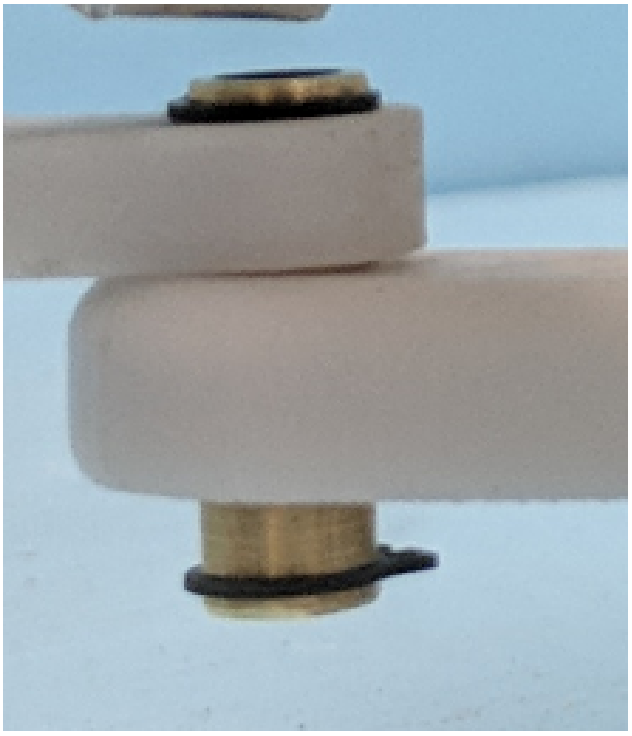


Figure 5.3: Brass Tube Revolute Joint

ness. The axis of the joints were brass tube that had retaining ring grooves machined into them, see Figure 5.3. The retaining rings are used at the end points of the joints to prevents brass tube from slipping out.

5.3 Rapid Prototyping

A Stratasys Fortus 450m using ABS filament was used for the rapid prototyping. Due to the complex geometry of many of the parts, a 3D printer was used to avoid complex machining procedures. To make the links as rigid and strong as possible the links were printed as solid rather than using a sparse matrix pattern. Inspite of using a solid pattern, the links that were roughly 1/4" in diameter and 3 inches long were not rigid enough to resist the forces generated by the mechanism. This resulted in large deflections that would cause the mechanism to seize. However, the main structures were 3D printed and had sufficient stiffness

and significantly simplified construction. The bearing surfaces for the joints all required post machining to create a smooth surface.

Several options are available that may have helped this issue that were not attempted. Changing the filament material to a stronger material is an option that was not explored. Also other additive manufacturing methods such as SLS were not attempted and may result in improved parts with higher precision that would provide improved performance.

5.4 Machined Parts

When parts were not able to be created using abs in the Fortus 450m the parts were then machined on manual mills and lathes using aluminum. However due to limited experience and access to the shop the manufacturing tolerances were on the order of 0.060 inches during the first iteration. This created joints with a significant amount of backlash. The backlash was to the extent that the mechanism would pass through a singularity and begin to move along the wrong branch. As stricter quality control requirements were implemented the manufacturing tolerances reduced until the mechanism was able to move consistently through the full range of motion. In future iterations the parts should be made using a CNC mill and lathe. The tight tolerances and high precision available in modern CNC machining will allow for the manufacturing of mechanisms with smaller backlash, which may in turn improve the overall movement of the mechanism.

5.5 Stock Parts

The ease of availability and the good quality of stock parts highly incentives the designer to to utilize them wherever possible. In the mechanisms constructed, gears, bushings, snap

rings/c-clips and some straight connectors were constructed from stock parts. Utilizing the stock parts ensured that the gears would mesh properly and saved time because some pieces only required adapter to be constructed or a simple modification such as a snap ring grove to be machined or a dowel hole to be drilled. The time savings realized by utilizing standard parts makes designing of the mechanism around increasing the use of such parts worth the additional design time and effort.

5.6 Planar Spatial Flapping Wing Mechanism

The physical model of the linkage was built, shown in Figure 5.4 and 5.5. The wing span of the flapping mechanism is 34in. The straight links and end caps were machined from aluminum or brass tubing. The remaining links and base were made from polycarbonate using additive manufacturing. The revolute joints used brass tubes as bushings while the spherical joints used a compliant joint that used a wire rope to connect the links. The right and left wings are driven by a gear train that connects a motor to their input cranks. The mechanism was driven at a rate of approximately 2-3 Hz, but was not driven any faster due to its size. The focus of this model is to confirm the movement of the proposed linkage. Future research will pursue developing methods to miniaturize and construct a model with a wing span of 5 in.

The link **ND** and the end caps for links **DE** and **AB** are made from machined parts, see Figure 5.6. The end caps for link **DE** and **AB** were originally made from 3D printing ABS, but the wall thicknesses required were too thin for the applied loads, so the end caps suffered from a bulk shear rupture originating from the pin holes. The torque applied to link **ND** caused the ABS version to break also, thus the transition to aluminum.

The purchased parts include the motor, gears, brass tube, pins and end caps, see Figure 5.7

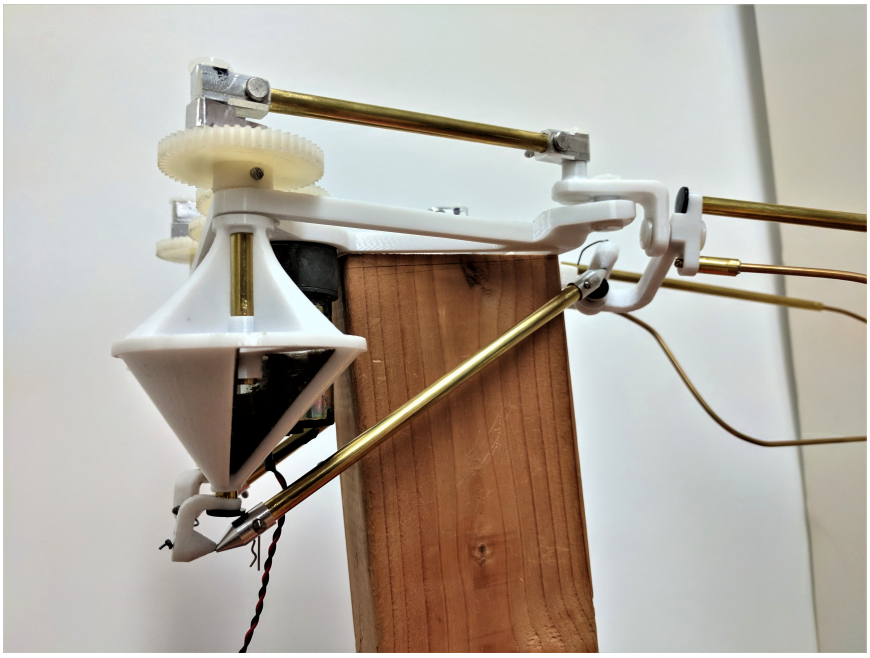
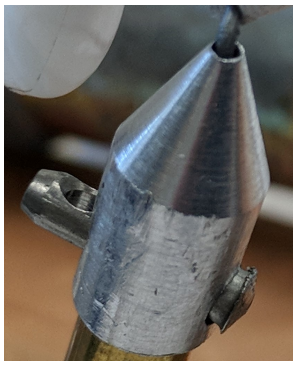
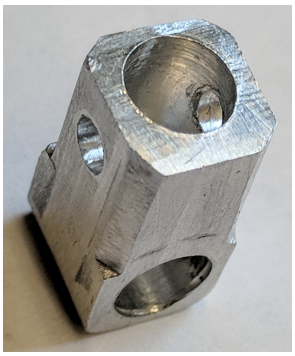


Figure 5.4: Physical Model of the Flapping Wing Mechanism

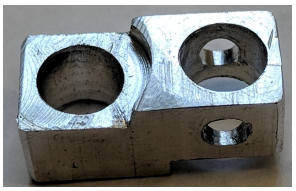
Figure 5.5: Gear Train and Input Cranks of the Physical Model of the Flapping Wing Mechanism.



(a) End Caps for Link **AB**



(b) End Caps for Link **DE**



(c) Link **ND**

Figure 5.6: Machined Parts for the Planar Spatial Flapping Wing Mechanism.

and 5.8. The brass tubes were used as both link elements fastened to the end caps using pins cut from 1/8" round stock as well as used as the pin connection for revolute joints. The end caps were used to constrain the axial movement of the revolute joints.

The parts that were successfully 3D printed using ABS were the links **EGC**, **NA**, **CB**, the mating connections to link **AB** and the base, see Figure 5.9 and 5.10. The links and components that had complex shapes such as many precise holes or holes with skew axes were chosen to be 3D printed. All of the holes were printed undersized and then reamed in order to ensure a good fit.

5.7 RRR-2SS Flapping Wing Mechanism

The physical model of the RRR-2SS flapping wing mechanism can be seen in Figures 5.12 and 5.13. Here the SS dyads can be seen to be two universal joints with a revolute joint between them. The revolute joint between them is a brass tube held in place by a set of c clips which sit in machined groves in the tube. The universal joints are held apart by a white tube that acts as a spacer outside of the brass tube. The construction went through several iterations. Initially many parts of mechanism were 3D printed, but it was quickly found that many of the links had excessive compliance or insufficient strength, so gradually

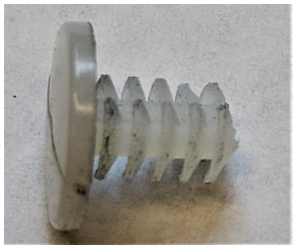
(a) Planar Spatial FWM Motor

(b) Gears in Gear Train.

(c) Brass Tube used between joints **N** and **D**

Figure 5.7: Purchased Parts for the Planar Spatial Flapping Wing Mechanism.

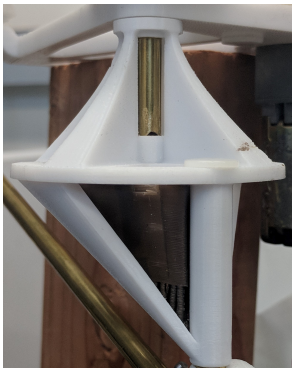
(a) Pin Connector



(b) Plug Used in Brass Tubes

Figure 5.8: Purchased Parts for the Planar Spatial Flapping Wing Mechanism.

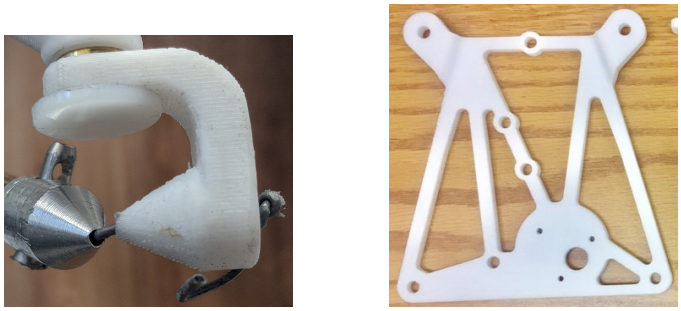
(a) Link **EGC**



(b) Link **NA**

(c) Link **CB**

Figure 5.9: 3D printed parts for Planar Spatial Flapping Wing Mechanism.



(a) Spherical Joint (b) Base Structure

Figure 5.10: 3D printed parts for Planar Spatial Flapping Wing Mechanism.

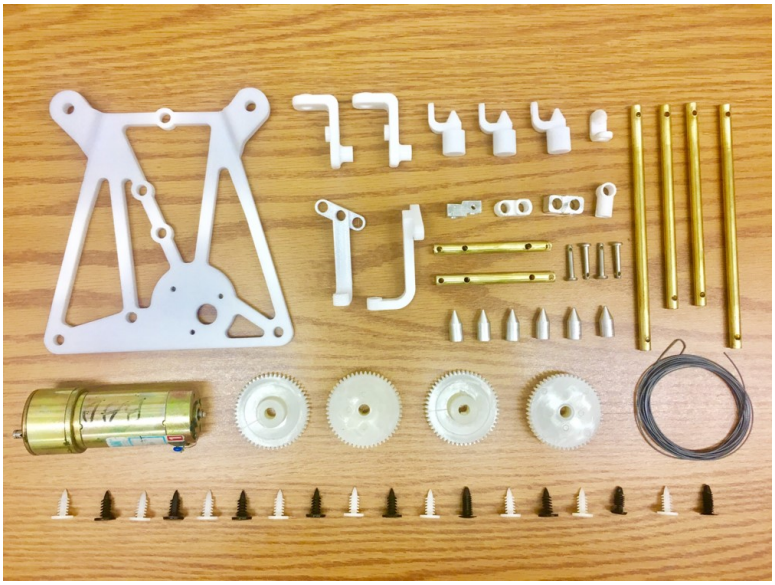


Figure 5.11: Parts use to build the Planar Spatial Flapping Wing Mechanism.

the pieces were replaced with machined aluminum and purchased parts such as brass tubes. In this iteration the plugs used in the previous design were too large to fit, so the plugs were replaced with snap rings with grooves machined into the brass tubes. The complexity of a few of the parts would require a CNC mill to manufacture, so the designs of those links were adjusted such that they would be able to constructed from 3D printed ABS.

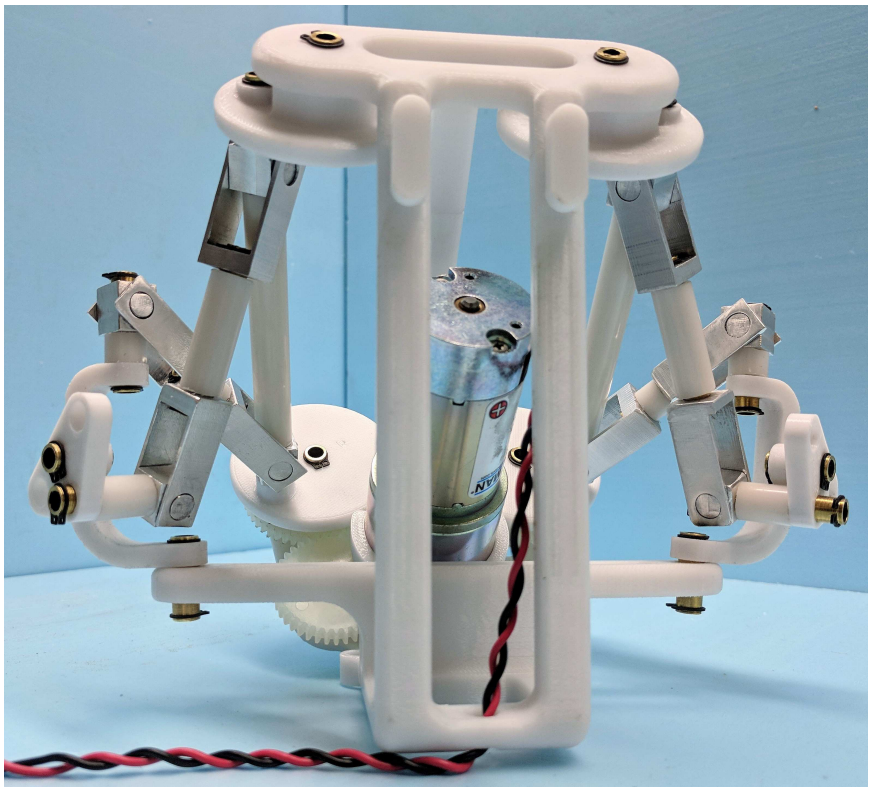


Figure 5.12: Front view of the physical model without the wings.

The machined parts consist of the univeral joints as well as the link **FC**, see Figure 5.14. The original **FC** link used a brass tube and a spacer, with the brass tube penetrating the disk unconstrained axially at the top of the mechanism. However, there was excessive backlash between the disk and the tube which caused the mechanism to lock up. So a new keyed **FC** link was designed and constructed, Figure 5.14b.

The purchased parts consisted of the motor, the brass tube, the nylon tube, the gears and the c-clips/snap rings, see Figure 5.15 and 5.16. The brass tube had to be cut to length and

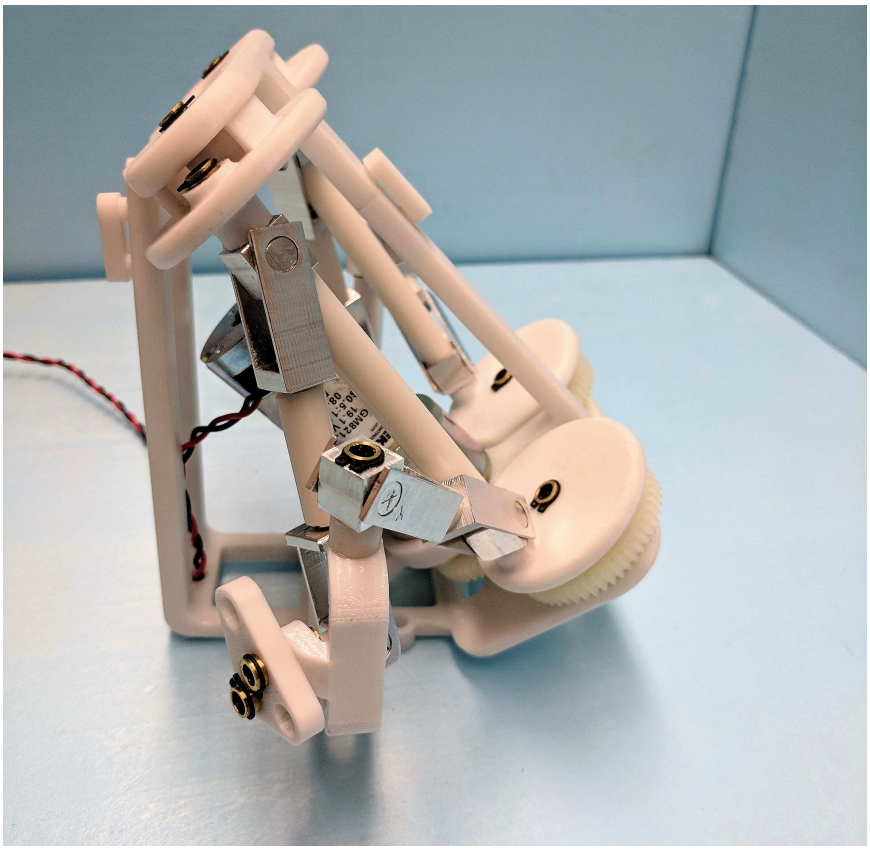
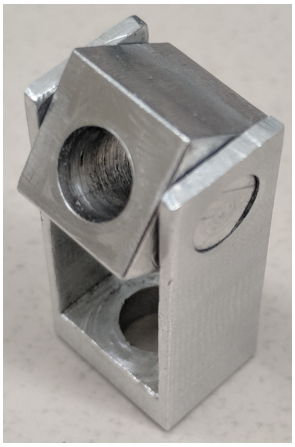


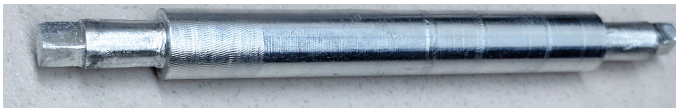
Figure 5.13: The wing swing mechanism of the physical model.

snap ring grooves machined into the ends of the tube. The nylon tubes only had to be cut to length. The gears needed to have keying features machined into them in order to ensure that the load was transmitted from the gear to the driving links. The keying features are shown in Figure 5.15c are the holes drilled in the gear connecting it to the disk of link **OC**.

The 3D printed parts were made from ABS also using the Stratasys Fortus 450m. The components that were 3D printed were links **DE**, **BAD**, **OF**, **OC** and the Frame, see Figure 5.17 and 5.18. Link **BAD** had to undergo a redesign because the original link failed during testing. The redesigned link had a significantly thicker cross section around the joint, see Figure 5.18. Links **OF** and **OC** were modified from the original pass through design to have a square insert to ensure that the link connecting them remained rigid and minimized the backlash between the two joints as they were supposed to move together as a single unit.



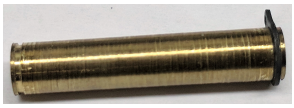
(a) Univeral Joint



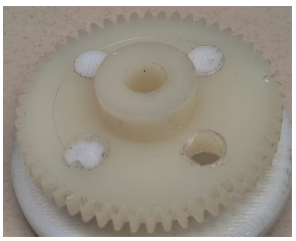
(b) Link **FC**

Figure 5.14: 3D printed parts for RRR-2SS Mechanism.

(a) Electric Motor

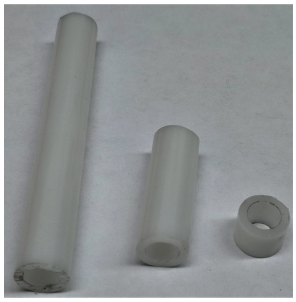


(b) Brass Tube with Groove



(c) Gears

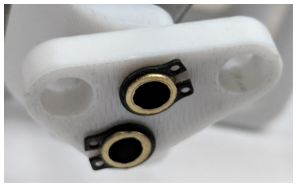
Figure 5.15: Purchased parts for RRR-2SS Mechanism



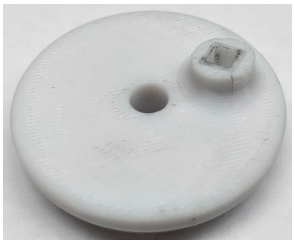
(a) Nylon Spacer Tube

(b) Snap Ring / C-clips

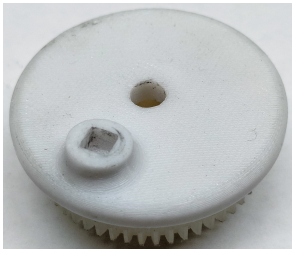
Figure 5.16: Purchased parts for RRR-2SS Mechanism.



(a) Link **DE**

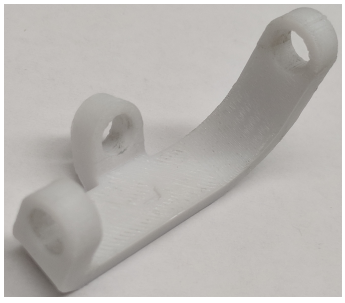


(b) Link **OF**

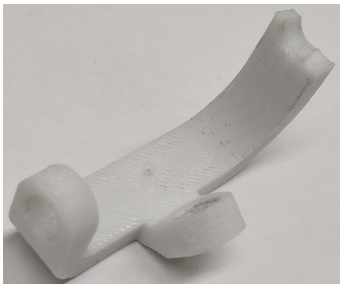


(c) Link **OC**

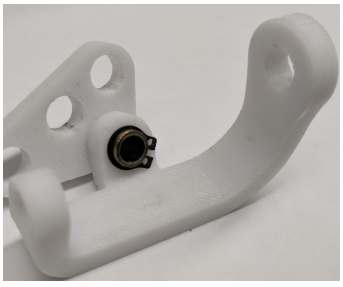
Figure 5.17: 3D printed parts for RRR-2SS Mechanism



(a) Original Link **BAD**



(b) Broken Link **BAD**



(c) Redesigned Link **BAD**

Figure 5.18: Redesign of Link **BAD**.

5.8 Summary

Physical Models of the RRR-RR-SS and the RRR-2SS flapping wing mechanisms have been constructed. The linkages were resistant to some backlash, but this often caused significant stress concentrations to develop breaking the links. There are still significant challenges with building spatial linkages given the limited angular movement available from off the shelf ball joints. The wire rope compliant joint and the universal joint with an extra rotation both seem to be able to accomplish the necessary movement, but both systems seem to have problems with backlash due to manufacturing tolerances. Future iterations may require precision machined parts. Further research and development is needed to produce a systematic method to reliably construct these types of spatial linkages.

Chapter 6

Conclusions

6.1 Contributions

The research presented in this dissertation provides a new capability for the design of spatial six-bar linkages. By defining the task in terms of a spatial serial chain, we can use the synthesis equations for spatial SS dyads to solve for the dimensions of the constraining links. This provides a computationally practical approach the spatial linkage synthesis.

The solution of the synthesis equations are evaluated to assess performance. The process is iterated with random task variations within designer specified tolerances in order to obtain a variety of successful designs. For example, 2600 iterations of the synthesis routine for RPR-2SS valve mechanism required approximately 180 secs, and yielded 459 successful designs. The packaging requirements of a linkage can be represented by the ratio of longest to shortest link dimensions, which was used to sort the resulting spatial six-bar linkage designs.

This design methodology resulted in a new flapping mechanism that coordinates the wing swing and wing pitch to achieve effective hovering flight. It also yielded a new valve mecha-

nism to control foam injection in tunnel boring machines. Finally, a new linkage that models the leg movement to deploy a patagium, or skin flap, of a flying squirrel was obtained.

6.2 Future Research

This dissertation has focused on the design of spatial linkages that have two ternary links connected by binary links, the Stephenson topology. This procedure can be generalized to obtain spatial six-bar linkages that have connected ternary links called the Watt topology. It can be generalized further by finding points in the moving links of the serial chains that lie on surfaces other than a sphere, such as a plane, cylinder, hyperboloid, and torus [80].

This procedure for the synthesis of a spatial six-bar linkage generalizes previous research on the synthesis of planar six-bar linkages [78]. The same technique can generalize the synthesis of planar eight-bar linkages to spatial eight-bar linkages by constraining four-link spatial serial chains, and possibly by constraining six-link spatial closed chains.

Bibliography

[1] P. Avila-Hernandez and F. Cuenca-Jimenez. Design and synthesis of a 2 DOF 9-bar spatial mechanism for a prosthetic thumb. *Mechanism and Machine Theory*, 121:697–717, mar 2018.

[2] S.-M. Baek, D.-Y. Lee, and K.-J. Cho. Curved compliant facet origami-based self-deployable gliding wing module for jump-gliding. In *Volume 5B: 40th Mechanisms and Robotics Conference*. ASME, aug 2016.

[3] S. Bai and J. Angeles. A robust solution of the spatial burmester problem. *Journal of Mechanisms and Robotics*, 4(3):031003, 2012.

[4] S. Bai, D. Wang, and H. Dong. A unified formulation for dimensional synthesis of stephenson linkages. *Journal of Mechanisms and Robotics*, 8(4):041009, 2016.

[5] M. Balta, K. A. Ahmed, P. L. Wang, J. M. McCarthy, and H. E. Taha. Design and manufacturing of flapping wing mechanisms for micro air vehicles. In *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0509, 2017.

[6] G. J. Berman and Z. J. Wang. Energy-minimizing kinematics in hovering insect flight. *Journal of Fluid Mechanics*, 582(1):153,168, 2007.

[7] K. L. Bishop. The relationship between 3-d kinematics and gliding performance in the southern flying squirrel, glaucomys volans. *Journal of Experimental Biology*, 209(4):689–701, feb 2006.

[8] R. J. Brodell and A. Soni. Design of the crank-rocker mechanism with unit time ratio. *Journal of Mechanisms*, 5(1):1–4, 1970.

[9] R. R. Bulatovic, S. R. Dordevic, and V. S. Dordevic. Cuckoo search algorithm: a metaheuristic approach to solving the problem of optimum synthesis of a six-bar double dwell linkage. *Mechanism and Machine Theory*, 61:1–13, 2013.

[10] L. Burmester. Lehrbuch der kinematik (arthur felix, leipzig 1886). *Design equations for finitely and infinitesimally separated position synthesis of binary link and combined link chains*, 1886.

[11] J. J. Cervantes-Sánchez, L. Gracia, J. M. Rico-Martínez, H. I. Medellín-Castillo, and E. J. González-Galván. A novel and efficient kinematic synthesis approach of the spherical 4r function generator for five and six precision points. *Mechanism and Machine Theory*, 44(11):2020–2037, 2009.

[12] J. J. Cervantes-Sanchez, M. A. Moreno-Baez, L. A. Aguilera-Cortes, and E. J. Gonzalez-Galvan. Kinematic design of the rssr-sc spatial linkage based on rotatability conditions. *Mechanism and Machine Theory*, 40:1126–1144, 2005.

[13] J. J. Cervantes-Sánchez, J. M. Rico-Martínez, V. H. Pérez-Muñoz, and A. Bitangilagy. Function generation with the rrrcr spatial linkage. *Mechanism and Machine Theory*, 74:58–81, 2014.

[14] P. Chen and B. Roth. Design equations for finitely and infinitesimally separated position synthesis position synthesis of binary links and combined link chains. *Journal of Engineering for Industry*, 91(1), 1969.

[15] P. Chen and B. Roth. A unified theory for the finitely and infinitesimally separated position problems of kinematic synthesis. *Journal of Engineering for Industry*, 91(1):203, 1969.

[16] C. H. Chiang, Y.-N. Yang, and W. H. Chieng. Four-position synthesis for spatial mechanisms with two independent loops. *Mechanism and Machine Theory*, 29(2):265–279, 1994.

[17] J. Chu and J. Sun. A new approach to dimension synthesis of spatial four-bar linkage through numerical atlas method. *Journal of Mechanisms and Robotics*, 2(4):041004, 2010.

[18] W.-Y. Chung. Synthesis of spatial mechanism ur-2ss for path generation. *Journal of Mechanisms and Robotics*, 7:041009–1–9, 2015.

[19] A. T. Conn, S. C. Burgess, and C. S. Ling. Design of a parallel crank-rocker flapping mechanism for insect-inspired micro air vehicles. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 221(10):1211–1222, 2007.

[20] V. Cossalter, M. Da Lio, and A. Doria. Optimum synthesis of spatial function generator mechanisms. *Meccanica*, 28(4):263–268, 1993.

[21] J. D'Alessio, K. Russell, W.-T. Lee, and R. S. Sodhi. On the application of RRSS motion generation and RRSS axode generation for the design of a concept prosthetic knee. *Mechanics Based Design of Structures and Machines*, 45(3):406–414, aug 2016.

[22] N. Davitashvili and O. Gelashvili. Synthesis of a spatial five-link mechanism with two degrees of freedom according to the given laws of motion. In *Proceedings of EUCOMES 08*, pages 159–166. Springer Netherlands, 2009.

[23] G. C. deCroon, K. M. E. deClercq, R. Ruijsink, B. Remes, and C. deWagter. Design, aerodynamics, and vision based control of the delfly. *International Journal of Micro Air Vehicles*, 1(2):71–97, 2009.

[24] J. Denavit and R. S. Hartenberg. Approximate synthesis of spatial linkages. *Journal of Applied Mechanics*, 27(1):201–206, 1960.

[25] A. L. Desbiens, M. Pope, F. Berg, Z. E. Teoh, J. Lee, and M. Cutkosky. Efficient jumpgliding: Theory and design considerations. In *2013 IEEE International Conference on Robotics and Automation*. IEEE, may 2013.

[26] A. Dhingra, J. Cheng, and D. Kohli. Synthesis of six-link, slider-crank and four-link. mechanisms for function, path and motion generation using homotopy with m-homogenization. *Journal of Mechanical Design*, 116:1122–1122, 1994.

[27] D. B. Doman, M. W. Oppenheimer, and D. O. Sigthorsson. Wingbeat shape modulation for flapping-wing micro-air-vehicle control during hover. *Journal of Guidance, Control and Dynamics*, 33(3):724–739, 2010.

[28] J. Duffy and H. Habib-Olahi. A displacement analysis of spatial five-link 3r-2c mechanisms—i. on the closures of the rcrcr mechanism. *Journal of Mechanisms*, 6(3):289–301, 1971.

[29] J. Duffy and H. Habib-Olahi. A displacement analysis of spatial five-link 3r-2c mechanisms part 2: Analysis of the rrcrc mechanism. *Journal of Mechanisms*, 6(4):463–473, 1972.

[30] J. Duffy and H. Habib-Olahi. A displacement analysis of spatial five-link 3r-2c mechanisms part 3. analysis of the rcrrc mechanism. *Mechanism and Machine Theory*, 7(1):71–84, 1972.

[31] EFNARC. *Specification and Guidelines for the use of specialist products for Mechanised Tunnelling (TBM) in Soft Ground and Hard Rock*, April 2005.

[32] C. P. Ellington. The aerodynamics of hovering insect flight: Iv. aerodynamic mechanisms. *Philosophical Transactions Royal Society London Series B*, 305:79–113, 1984.

[33] R. L. Essner. Three-dimensional launch kinematics in leaping, parachuting and gliding squirrels. *Journal of Experimental Biology*, 205(16):2469–2477, 2002.

[34] R. L. Essner. *LOCOMOTION, MORPHOLOGY, AND HABITAT USE IN ARBOREAL SQUIRRELS (RODENTIA: SCIURIDAE)*. PhD thesis, Ohio University, 2003.

[35] F. Freudenstein. An analytical approach to the design of four-link mechanisms. *Trans. ASME*, 76(3):483–492, 1954.

[36] X. Ge, Q. J. Ge, and F. Gao. A novel algorithm for solving design equations for synthesizing platform linkages. In *Volume 5C: 39th Mechanisms and Robotics Conference*. ASME, aug 2015.

[37] X. Ge, A. Purwar, and Q. J. Ge. From 5-SS platform linkage to four-revolute jointed planar, spherical and bennett mechanisms. In *Volume 5B: 40th Mechanisms and Robotics Conference*. ASME, aug 2016.

[38] J. Gerdes, A. Holness, A. Perez-Rosado, L. Roberts, A. Greisinger, E. Barnett, J. Kempny, D. Lingam, C.-H. Yeh, H. A. Bruck, et al. Robo raven: a flapping-wing air vehicle with highly compliant and independently controlled wings. *Soft Robotics*, 1(4):275–288, 2014.

[39] R. L. Goldingay and B. D. Taylor. Gliding performance and its relevance to gap crossing by the squirrel glider (petaurus norfolcensis). *Australian Journal of Zoology*, 57(2):99, 2009.

[40] V. Gupta. Kinematic analysis of plane and spatial mechanisms. *Journal of Engineering for Industry*, 95(2):481–486, 1973.

[41] R. S. Hartenberg and J. Denavit. *Kinematic synthesis of linkages*. McGraw-Hill, 1964.

[42] Z. Hu and X. Deng. Design and performance of insect inspired high frequency flapping wing robots. *Stainless steel*, 8:0–32, 2002.

[43] W.-M. Hwang and Y.-J. Chen. Defect-free synthesis of stephenson-ii function generators. *Journal of Mechanisms and Robotics*, 2(4):041012, 2010.

[44] C. Innocenti. Polynomial solution of the spatial burmester problem. *Journal of Mechanical Design*, 117(1):64–68, March 1995.

[45] S. Jancsecz, R. Krause, and L. Langmaack. Advantages of soil conditioning in shield tunnelling: experiences of lrts izmir. *Challenges for the 21st Century, Alten et al (eds)*, pages 865–875, 1999.

[46] M. Keennon, K. Klingebiel, H. Won, and A. Andriukov. Development of the nano hummingbird: A tailless flapping wing micro air vehicle. In *AIAA Aerospace Sciences Meeting*, pages 1–24, Reston, VA, 2012. AIAA.

[47] M. Kovac, Wassim-Hraiz, O. Fauria, J.-C. Zufferey, and D. Floreano. The EPFL jumpglider: A hybrid jumping and gliding robot with rigid or folding wings. In *2011 IEEE International Conference on Robotics and Biomimetics*. IEEE, dec 2011.

[48] X. Li, Q. J. Ge, and F. Gao. A unified algorithm for geometric design of platform linkages with spherical and plane constraints. In *Volume 5B: 38th Mechanisms and Robotics Conference*. ASME, aug 2014.

[49] X. Li, W. Wang, Y. Tang, L. Wang, T. Bai, F. Zhao, and Y. Bai. Research on gliding aerodynamic effect of deformable membrane wing for a robotic flying squirrel. *Journal of Bionic Engineering*, 15(2):379–396, mar 2018.

[50] Q. Liao and J. M. McCarthy. On the seven position synthesis of a 5-ss platform linkage. *Journal of Mechanical Design*, 123(1):74–79, 2001.

[51] K. Y. Ma, S. M. Felton, and R. J. Wood. Design, fabrication, and modeling of the split actuator microrobotic bee. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1133–1140. IEEE, 2012.

[52] U. Maidl. *Erweiterung der Einsatzbereiche der Erddruckschilde durch bodenkonditionierung mit Schaum*. Ruhr-Universität Bochum, Institut für Konstruktiven Ingenieurbau, 1992.

[53] C. Mazzotti, M. Troncossi, and V. Parenti-Castelli. Dimensional synthesis of the optimal rssr mechanism for a set of variable design parameters. *Meccanica*, 52(10):2439–2447, 2017.

[54] J. M. McCarthy and G. S. Soh. *Geometric design of linkages*, volume 11. Springer Science & Business Media, 2010.

[55] C. McLarnan. Synthesis of six-link plane mechanisms by numerical analysis. *Journal of Engineering for Industry*, 85(1):5–10, 1963.

[56] K. Nafees and A. Mohammad. Dimensional synthesis of six-bar stephenson ii linkage for fifteen precision points path generation. *Perspectives in Science*, 8:485–487, 2016.

[57] K. Nafees and A. Mohammad. Dimensional synthesis of six-bar stephenson iii mechanism for 12 precision points path generation. *International Journal of Mechanisms and Robotic Systems*, 3(1):80–90, 2016.

[58] M. W. Oppenheimer, D. B. Doman, and D. O. Sigthorsson. Dynamics and control of a biomimetic vehicle using biased wingbeat forcing functions. *Journal Guidance, Control and Dynamics*, 34(1):204–217, 2011.

[59] P. Pamidi and F. Freudenstein. On the motion of a class of five-link, rcrcr, spatial mechanisms. *Journal of Engineering for Industry*, 97(1):334–339, 1975.

[60] A. Perez-Gracia. Synthesis of spatial RPRP closed linkages for a given screw system. *Journal of Mechanisms and Robotics*, 3(2):021009, 2011.

[61] M. Plecnik, J. M. McCarthy, and C. W. Wampler. Kinematic synthesis of a watt i six-bar linkage for body guidance. In *Advances in Robot Kinematics*, pages 317–325. Springer, 2014.

[62] M. M. Plecnik and J. M. McCarthy. Numerical synthesis of six-bar linkages for mechanical computation. *Journal of Mechanisms and Robotics*, 6(3):031012, 2014.

[63] M. M. Plecnik and J. M. McCarthy. Computational design of stephenson ii six-bar function generators for 11 accuracy points. *Journal of Mechanisms and Robotics*, 8(1):011017, 2016.

[64] M. M. Plecnik and J. M. McCarthy. Controlling the movement of a trr spatial chain with coupled six-bar function generators for biomimetic motion. *Journal of Mechanisms and Robotics*, 8(5):051005, 2016.

[65] M. M. Plecnik and J. M. McCarthy. Design of stephenson linkages that guide a point along a specified trajectory. *Mechanism and Machine Theory*, 96:38–51, 2016.

[66] M. M. Plecnik and J. M. McCarthy. Kinematic synthesis of stephenson iii six-bar function generators. *Mechanism and Machine Theory*, 97:112–126, 2016.

[67] A. Ramezani, S.-J. Chung, and S. Hutchinson. A biomimetic robotic platform to study flight specializations of bats. *Science Robotics*, 2(3), 2017.

[68] A. M. Rao and G. Sandor. Extension of freudenstein's equation to geared linkages. *Journal of Engineering for Industry*, 93(1):201–210, 1971.

[69] A. M. Rao, G. N. Sandor, D. Kohli, and A. Soni. Closed form synthesis of spatial function generating mechanism for the maximum number of precision points. *Journal of Engineering for Industry*, 95(3):725–736, 1973.

[70] U. Rehm. Maschineller tunnelvortrieb unter sehr schwierigen geologischen verhältnissen. *Tunnel- Tiefbautagung Gyoer*, page 99 ff, 2004.

[71] R. Sancibrian. Improved grg method for the optimal synthesis of linkages in function generation problems. *Mechanism and Machine Theory*, 46(10):1350–1375, 2011.

[72] G. N. Sandor, L. J. Xu, and S. P. Yang. Computer-aided synthesis of two-closed-loop rssr-ss spatial motion generator with branching and sequence constraints. *Mechanism and Machine Theory*, 21:345–350, 1986.

[73] J. S. Scheibe, K. E. Paskins, S. Ferdous, and D. Birdsill. Kinematics and functional morphology of leaping, landing, and branch use inGlaucomys sabrinus. *Journal of Mammalogy*, 88(4):850–861, aug 2007.

[74] L. Schenato, D. Campolo, and S. S. Sastry. Controllability issues in flapping flight for biomemetic mavs. In *42nd IEEE conference on Decision and Control*, volume 6, pages 6441–6447. IEEE, 2003.

[75] A. Schoenflies. *Geometrie der Bewegung in synthetischer Darstellung*. BG Teubner, 1886.

[76] P. Seshadri, M. Benedict, and I. Chopra. Understanding micro air vehicle flapping-wing aerodynamics using force and flowfield measurements. *Journal of Aircraft*, 2013.

[77] P. Shiakolas, D. Koladiya, and J. Kebrle. On the optimum synthesis of six-bar linkages using differential evolution and the geometric centroid of precision positions technique. *Mechanism and Machine Theory*, 40(3):319–335, 2005.

[78] G. S. Soh and J. M. McCarthy. The synthesis of six-bar linkages as constrained planar 3r chains. *Mechanism and Machine Theory*, 43(2):160–170, 2008.

[79] G. S. Soh and F. Ying. Motion generation of planar six-and eight-bar slider mechanisms as constrained robotic systems. *Journal of Mechanisms and Robotics*, 7(3):031018, 2015.

[80] H.-J. Su, J. M. McCarthy, and L. T. Watson. Generalized linear product homotopy algorithms and the computation of reachable surfaces. *Journal of Computing and Information Science in Engineering*, 4(3):226, 2004.

[81] C. H. Suh. Design of space mechanisms for function generation. *Journal of Engineering for Industry*, 90(3):507–512, 1968.

[82] A. Svoboda. *Computing mechanisms and linkages*. McGraw-Hill, 1948.

[83] H. E. Taha, A. H. Nayfeh, and M. R. Hajj. Effect of the aerodynamic-induced parametric excitation on the longitudinal stability of hovering mavs/insects. *Nonlinear Dynamics*, 78(4):2399–2408, 2014.

[84] H. E. Taha, S. Tahmasian, C. A. Woolsey, A. H. Nayfeh, and M. R. Hajj. The need for higher-order averaging in the stability analysis of hovering mavs/insects. *Bioinspiration and Biomimetics*, 10(1):016002, 2015. Selected in the Bioinspiration & Biomimetics Highlights of 2015.

[85] M. Thewes. *Adhasion von Tonböden beim Tunnelvortrieb mit Flüssigkeitsschilden [Adhesion of clay soil during tunneling with liquid shields]*. PhD thesis, Universitaet Wuppertal, 1999.

[86] L. Tsai. Enumeration of kinematic structures according to function, florida, 2001.

[87] B. Y. Tsuge, M. M. Plecnik, and J. M. McCarthy. Homotopy directed optimization to design a six-bar linkage for a lower limb with a natural ankle trajectory. *Journal of Mechanisms and Robotics*, 8(6):061009, 2016.

[88] J. J. Uicker, G. R. Pennock, J. E. Shigley, et al. *Theory of machines and mechanisms*, volume 1. Oxford University Press New York, 2011.

[89] P. L. Wang and J. M. McCarthy. Design of a flapping wing mechanism to coordinate both wing swing and wing pitch. *Journal of Mechanisms and Robotics*, 10(2):025003, jan 2018.

[90] P. L. Wang, H. E. Taha, and J. M. McCarthy. Synthesis of a flapping wing mechanism using a constrained spatial rrr serial chain. *Journal of Mechanisms and Robotics*, 10(1), 2018.

[91] G. Wei and J. S. Dai. A spatial eight-bar linkage and its association with the deployable platonic mechanisms. *Journal of Mechanisms and Robotics*, 6(2):021010, mar 2014.

[92] T. Weis-Fogh. Quick estimates of flight fitness in hovering animals, including novel mechanisms for lift production. *Journal of Experimental Biology*, 59(1):169–230, 1973.

[93] J. T. Wilson. Analytical kinematic synthesis by finite displacements. *Journal of Engineering for Industry*, 87(2):161–169, 1965.

[94] Z. Yan, H. E. Taha, and M. R. Hajj. Effects of aerodynamic modeling on the optimal wing kinematics for hovering mavs. *Aerospace Science and Technology*, 45:39–49, 2015.

[95] A. Yang. Displacement analysis of spatial five-link mechanisms using (3× 3) matrices with dual-number elements. *Journal of Engineering for Industry*, 91(1):152–156, 1969.

[96] D. Youlatos and A. Samaras. Arboreal locomotor and postural behaviour of european red squirrels (sciurus vulgaris l.) in northern greece. *Journal of Ethology*, 29(2):235–242, dec 2010.

[97] M. S. Yuan. Displacement analysis of the rcrcr five-link spatial mechanism. *Journal of Mechanisms*, 6(1):119–134, 1971.

[98] M. S. C. Yuan. Displacement analysis of the rrccr five-link spatial mechanism. *Journal of Applied Mechanics*, 37:689, 1970.

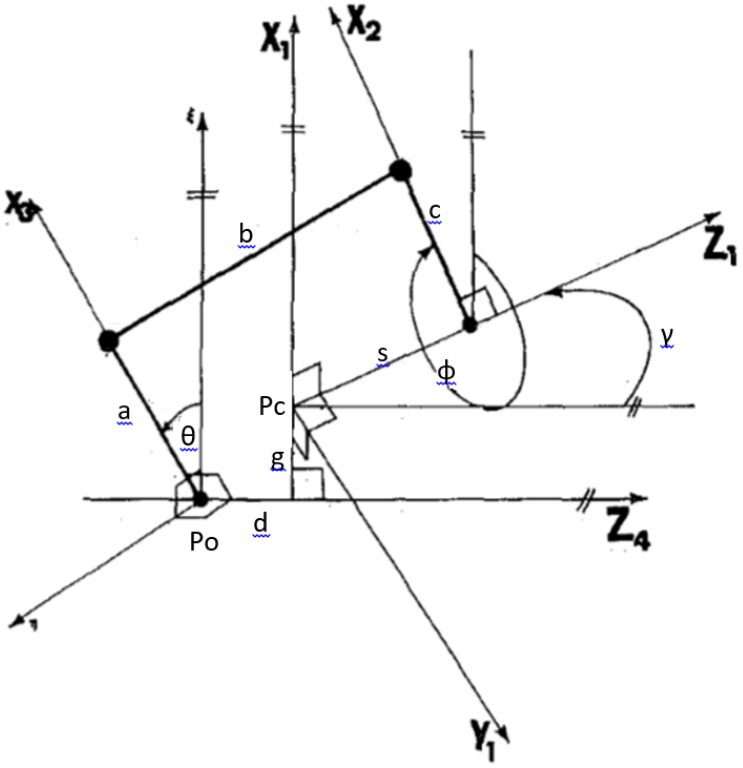
[99] X. Zhang, P. Lopez-Custodio, and J. S. Dai. Compositional submanifolds of prismatic-universal-prismatic and skewed prismatic-revolute-prismatic kinematic chains and their derived parallel mechanisms. *Journal of Mechanisms and Robotics*, 10(3):031001, mar 2018.

[100] P. Zhao, A. Purwar, and Q. Ge. Task driven unified synthesis of planar four-bar and six-bar linkages with revolute and prismatic joints for five position synthesis. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V06AT07A061–V06AT07A061. American Society of Mechanical Engineers, 2013.

Appendix A

RRR-RR-SS Flapping Wing Mechanism Mathematica Code

Variable γ value for RSSR



Basic Functions and Constants

```
Zmat[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], 0, 0},
   {Sin[x[[1]]], Cos[x[[1]]], 0, 0}, {0, 0, 1, x[[2]]}, {0, 0, 0, 1}};
Xmat[x_] := {{1, 0, 0, x[[2]]}, {0, Cos[x[[1]]], -Sin[x[[1]]], 0},
   {0, Sin[x[[1]]], Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Ymat[x_] := {{Cos[x[[1]]], 0, Sin[x[[1]]], 0}, {0, 1, 0, x[[2]]},
   {-Sin[x[[1]]], 0, Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Disp[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], x[[2]]},
   {Sin[x[[1]]], Cos[x[[1]]], x[[3]]}, {0, 0, 1}};
pt2linedist[x0_, x1_, x2_] := Norm[Cross[(x0 - x1), (x0 - x2)]] / Norm[x2 - x1];
pt2lineproj[x0_, x1_, x2_] := x1 + (Dot[x0 - x1, x2 - x1] / Dot[x2 - x1, x2 - x1]) * (x2 - x1);
rem = Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}];
```

Symbolic Manipulation For Synthesis

θ: input values

ϕ: output values

γ: moving revolute joint

```
RSSRin = {{θ1, θ2, θ3, θ4, θ5, θ6, θ7},
   {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7}, {γ1, γ2, γ3, γ4, γ5, γ6, γ7}};
```

Rotation matricies for the input crank

```
RSSRinputsA = Table[Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{RSSRin[[1, i]], g1}], {i, 7}];
```

Rotation Matricies for the output

```
RSSRinputsB = Table[Xmat[{RSSRin[[3, i]], g}].Zmat[{RSSRin[[2, i]], s}], {i, 7}];
```

Symbolic Manipulation For Analysis

Analysis Equations: There are additional rotations and translations compared to the planar four bar linkage, but once the additional rotations and traslations the solution of the output equation is the same as that for a planar four bar function generator

```
analysiseqn = ExpandAll[
   TrigExpand[Dot[Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{θ, g1}].Xmat[{0, a}].{0, 0, 0, 1} -
        Xmat[{γ, g}].Zmat[{ϕ, s}].Xmat[{0, c}].{0, 0, 0, 1},
       Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{θ, g1}].Xmat[{0, a}].{0, 0, 0, 1} -
        Xmat[{γ, g}].Zmat[{ϕ, s}].Xmat[{0, c}].{0, 0, 0, 1}]] - b^2];
Acoeff = Coefficient[analysiseqn, Cos[ϕ]];
Bcoeff = Coefficient[analysiseqn, Sin[ϕ]];
Ccoeff = Simplify[analysiseqn - Acoeff * Cos[ϕ] - Bcoeff * Sin[ϕ]];
ϕeqn1 = ArcTan[Acoeff, Bcoeff] + ArcCos[-Ccoeff / Sqrt[Acoeff^2 + Bcoeff^2]];
ϕeqn2 = ArcTan[Acoeff, Bcoeff] - ArcCos[-Ccoeff / Sqrt[Acoeff^2 + Bcoeff^2]];
```

```
analysiseqn4paper =
  ExpandAll[TrigExpand[Dot[Zmat[{Pi/2, d}].Xmat[{Pi/2, t}].Zmat[{θ, 0}].
        ({u, v, w, 1}) - Xmat[{γ, g}].Zmat[{ϕ, 0}].({x, y, z, 1}),
      Zmat[{Pi/2, d}].Xmat[{Pi/2, t}].Zmat[{θ, 0}].({u, v, w, 1}) -
       Xmat[{γ, g}].Zmat[{ϕ, 0}].({x, y, z, 1})]] - b^2];
Acoeff4paper = Coefficient[analysiseqn4paper, Cos[ϕ]];
Bcoeff4paper = Coefficient[analysiseqn4paper, Sin[ϕ]];
Ccoeff4paper =
  Simplify[analysiseqn4paper - Acoeff4paper * Cos[ϕ] - Bcoeff4paper * Sin[ϕ]];
ϕeqn14paper = ArcTan[Acoeff4paper, Bcoeff4paper] +
   ArcCos[-Ccoeff4paper / Sqrt[Acoeff4paper^2 + Bcoeff4paper^2]];
ϕeqn24paper = ArcTan[Acoeff4paper, Bcoeff4paper] -
   ArcCos[-Ccoeff4paper / Sqrt[Acoeff4paper^2 + Bcoeff4paper^2]];
```

```
analysiseqn4paper
```

$$-b^2+d^2+g^2+t^2+u^2+v^2-2\,g\,w+w^2+x^2+y^2+z^2-2\,d\,z\,\text{Cos}[\gamma]+2\,t\,u\,\text{Cos}[\theta]+2\,d\,v\,\text{Cos}[\theta]-2\,v\,z\,\text{Cos}[\gamma]\,\text{Cos}[\theta]+2\,g\,x\,\text{Cos}[\phi]-2\,w\,x\,\text{Cos}[\phi]-2\,t\,y\,\text{Cos}[\gamma]\,\text{Cos}[\phi]-2\,u\,y\,\text{Cos}[\gamma]\,\text{Cos}[\theta]\,\text{Cos}[\phi]+2\,t\,z\,\text{Sin}[\gamma]+2\,u\,z\,\text{Cos}[\theta]\,\text{Sin}[\gamma]-2\,d\,y\,\text{Cos}[\phi]\,\text{Sin}[\gamma]-2\,v\,y\,\text{Cos}[\theta]\,\text{Cos}[\phi]\,\text{Sin}[\gamma]+2\,d\,u\,\text{Sin}[\theta]-2\,t\,v\,\text{Sin}[\theta]-2\,u\,z\,\text{Cos}[\gamma]\,\text{Sin}[\theta]+2\,v\,y\,\text{Cos}[\gamma]\,\text{Cos}[\phi]\,\text{Sin}[\theta]-2\,v\,z\,\text{Sin}[\gamma]\,\text{Sin}[\theta]-2\,u\,y\,\text{Cos}[\phi]\,\text{Sin}[\gamma]\,\text{Sin}[\theta]-2\,g\,y\,\text{Sin}[\phi]+2\,w\,y\,\text{Sin}[\phi]-2\,t\,x\,\text{Cos}[\gamma]\,\text{Sin}[\phi]-2\,u\,x\,\text{Cos}[\gamma]\,\text{Cos}[\theta]\,\text{Sin}[\phi]-2\,d\,x\,\text{Sin}[\gamma]\,\text{Sin}[\phi]-2\,v\,x\,\text{Cos}[\theta]\,\text{Sin}[\gamma]\,\text{Sin}[\phi]+2\,v\,x\,\text{Cos}[\gamma]\,\text{Sin}[\theta]\,\text{Sin}[\phi]-2\,u\,x\,\text{Sin}[\gamma]\,\text{Sin}[\theta]\,\text{Sin}[\phi]$$

```
Simplify[
 Expand[Simplify[Expand[Acoeff4paper /. {ψ → Pi/2, η → Pi/2, θ → μ, ϕ → ρ}]]]]
```

$$2\,g\,x-2\,w\,x-2\,t\,y\,\text{Cos}[\gamma]-2\,u\,y\,\text{Cos}[\gamma-\mu]-2\,d\,y\,\text{Sin}[\gamma]-2\,v\,y\,\text{Sin}[\gamma-\mu]$$

```
Simplify[
 Expand[Simplify[Expand[Bcoeff4paper /. {ψ → Pi/2, η → Pi/2, θ → μ, ϕ → ρ}]]]]
```

$$-2\,(g\,y-w\,y+t\,x\,\text{Cos}[\gamma]+u\,x\,\text{Cos}[\gamma-\mu]+d\,x\,\text{Sin}[\gamma]+v\,x\,\text{Sin}[\gamma-\mu])$$

```
Sort[Simplify[Expand[Ccoeff4paper /. {ψ → Pi/2, η → Pi/2, θ → μ, ϕ → ρ}]]]
```

$$-b^2+d^2+g^2+t^2+u^2+v^2-2\,g\,w+w^2+x^2+y^2+z^2-2\,d\,z\,\text{Cos}[\gamma]-2\,v\,z\,\text{Cos}[\gamma-\mu]+2\,t\,u\,\text{Cos}[\mu]+2\,d\,v\,\text{Cos}[\mu]+2\,t\,z\,\text{Sin}[\gamma]+2\,u\,z\,\text{Sin}[\gamma-\mu]+2\,d\,u\,\text{Sin}[\mu]-2\,t\,v\,\text{Sin}[\mu]$$

```
Ocoord = Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{θ, g1}].{0, 0, 0, 1};
Acoord = Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{θ, g1}].Xmat[{0, a}].{0, 0, 0, 1};
Bcoord = Xmat[{γ, g}].Zmat[{ϕ, s}].Xmat[{0, c}].{0, 0, 0, 1};
Ccoord = Xmat[{γ, g}].Zmat[{ϕ, s}].{0, 0, 0, 1};
```

The planar equation for a four bar linkage can be obtained by substituting in 0 for all but the elements that remain in the same plane.

```
Planarϕeqn1 = Simplify[ϕeqn1 /. {γ → 0, d → 0, s → 0, t → 0(*,ψ→ 0,η→ 0,g1→ 0*)}];
Planarϕeqn2 = Simplify[ϕeqn2 /. {γ → 0, d → 0, s → 0, t → 0(*,ψ→ 0,η→ 0,g1→ 0*)}];
```

```
Planarϕeqn1
```

```
ArcCos[ (-a^2 + b^2 - c^2 - g^2 - g1^2 +
      2 a g Cos[Θ] Cos[ψ] + 2 g g1 Sin[η] Sin[ψ] - 2 a g Cos[η] Sin[Θ] Sin[ψ]) /
    (2 Sqrt[c^2 (a^2 Cos[Θ]^2 - 2 a g Cos[Θ] Cos[ψ] + Cos[ψ]^2 (g1 Sin[η] - a Cos[η] Sin[Θ])^2 +
          (g - g1 Sin[η] Sin[ψ] + a Cos[η] Sin[Θ] Sin[ψ])^2)])] +
 ArcTan[c (g - a Cos[Θ] Cos[ψ] - g1 Sin[η] Sin[ψ] + a Cos[η] Sin[Θ] Sin[ψ]),
  c (Cos[ψ] (g1 Sin[η] - a Cos[η] Sin[Θ]) - a Cos[Θ] Sin[ψ])]
```

Equations to find new d and s values (translations along the the input and output axis of rotation): The solutions from the synthesis will give you new d and s values. Originally lengths of the common normal and axes of rotation are specified. The synthesis finds the coordinates for two poitns with coordinates of (u,v,w) and (x,y,z). These points will rotate about the axes of rotation and a link will connect the original D and S points to the new Pt B and Pt C, and will make a cone shape when rotating. So a calculations must be done to calculate the new common normal and the new displacement along the axis of rotation in order to solve the analysis.

```
Dnew = d;
projptS = Simplify[pt2lineproj[{x, y, z}, Xmat[{γ, g}].Zmat[{ϕ, s}].{0, 0, 0, 1}.rem,
    Xmat[{γ, g}].Zmat[{ϕ, s}].{0, 0, 1, 1}.rem]];
(*Snew=s+Norm[projpt-Zmat[{0,d}].Xmat[{γ,g}].Zmat[{ϕ,s}].{0,0,0,1}.rem];*)
Snew = Sign[Dot[Xmat[{γ, g}].{0, 0, 1, 1}.rem - Xmat[{γ, g}].{0, 0, 0, 1}.rem, projptS]] *
  Norm[projptS - Xmat[{γ, g}].{0, 0, 0, 1}.rem];
projptG1 = Simplify[pt2lineproj[{u, v, w}, Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{Θ, g1}].
    {0, 0, 0, 1}.rem, Zmat[{ψ, d}].Xmat[{η, t}].Zmat[{Θ, g1}].{0, 0, 1, 1}.rem]];
G1new = Sign[Dot[Zmat[{ψ, d}].Xmat[{η, t}].{0, 0, 1, 1}.rem -
      Zmat[{ψ, d}].Xmat[{η, t}].{0, 0, 0, 1}.rem, projptG1]] *
   Norm[projptG1 - Zmat[{ψ, d}].Xmat[{η, t}].{0, 0, 0, 1}.rem];
```

The new A and C values represent the projected distance of the points to the axis of rotation.

```
Anew = Norm[{u, v, w} - projptG1];
Cnew = Norm[{x, y, z} - projptS];
```

Modules for Generation

G is the moving crank position for the input at the first task position (from the origin to first moving pivot);
W is the moving crank position for the output at the first task position (from the second gound pivot to the moving pivot);

The synthesis module below is very similar to that of the planar four bar synthesis method by khaustub. It follows the following steps

1.) Collect the input values and constants. (*note that there is the option to have a variable gamma angle which in the flapping wing mechanims is the wings flap while the output is the wing's pitch.)
2.) Substitute into the homogeneous transform matricies
3.) Calculate the Relative Transform Matricies which are relative to the location of the first point (transform matrix)
4.) Define the coordinates of the two moving pivots which are being solved for. (W and G)
5.) Set up the Constrain Equations. The link length between the two moving pivots must remain constant. Using the relative transform matricies the two defined coordiantes can be moved through space.
6.) Set up the Design Equations. Subtract the first constraint equation from the remaining constraint equations to remove the unknown link length variable.
7.) Solve the design equations. For a spatial four bar function generator there are seven sets of inputs and outputs, so there will be six design equations which coorrelate with the six unknown coordinates that need to be solved for.
8.) Keep only the real numbered solutions.
9.) Calculate the distance between the points. (I call this leg lengths, but this is not actually the leg lengths used in the Analysis. Only the legth between the points remains the same. The input crank and output are actually the distance the respective points are from the axis of rotation. I use the leg lengths as the "title" for each set of points.)
10.) The solutions of of the coordinates and "leg lengths" are stored.
11.) The solutions are plugged back into the constrain equations to ensure that all values are returned as zero.
12.) Calculate the input offset (θ offset). This finds the angle of the coordinates in the plane in which the point rotates from the initial 0 angle. This is especially important when coupling with a planar four bar for the flapping wing since both must be offset from the same initial crank position.
13.) Calculate the output offset (ϕ offset). Performed the same ways as the θ offeset

```
RSSRSynthesis[thetas_, phis_, gammas_, Const_] :=
 Module[{RSSRsubs, RSSRinput2useA, RSSRinput2useB, RSSRrelA, RSSRrelB, W,
   x, y, z, G, u, v, w, a, c, b, R, RSSRConstraintEqn, RSSRDesignEqn, numsols,
   realnumsols, numsols2use, leglengths, uvwxyz2use, test, thetatransform,
   thetaAdj, ThetaAdjFinal, phiAdj, PhiAdjFinal, phitransform, legsubs, subs},
  RSSRsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6, θ7}, {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7},
       {γ1, γ2, γ3, γ4, γ5, γ6, γ7}, {g, d, s, t, g1, η, ψ}}] →
     Flatten[{thetas, phis, gammas, Const}]];
  RSSRinput2useA = RSSRinputsA /. RSSRsubs;
  RSSRinput2useB = RSSRinputsB /. RSSRsubs;
  RSSRrelA = Table[RSSRinput2useA[[i]].Inverse[RSSRinput2useA[[1]]], {i, 7}];
  RSSRrelB = Table[RSSRinput2useB[[i]].Inverse[RSSRinput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  RSSRConstraintEqn = Table[Dot[RSSRrelA[[i]].G - RSSRrelB[[i]].W,
      RSSRrelA[[i]].G - RSSRrelB[[i]].W] - R^2, {i, 7}];
  RSSRDesignEqn = Table[Chop[Expand[RSSRConstraintEqn[[i + 1]] -
       RSSRConstraintEqn[[1]]]], {i, 6}];
  numsols = NSolve[RSSRDesignEqn == {0, 0, 0, 0, 0, 0}, {u, v, w, x, y, z},
    Method → "EndomorphismMatrix"];
  realnumsols = Flatten[Position[(u + v + w + x + y + z) /. numsols,
     val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*These are the lengths of {a,c,b}*)
  leglengths =
   Table[{Norm[{u, v, w} /. numsols2use[[i]]], Norm[({x, y, z} /. numsols2use[[i]]) -
       ((RSSRinput2useB[[1]].{0, 0, 0, 1} /. RSSRsubs)).
        Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}]],
     Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])]},
    {i, Length[numsols2use]}];
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{a, c, b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Table[((RSSRConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
    {i, Length[numsols2use]}];
  thetatransform = Round[Table[Inverse[(Zmat[{ψ, d}].Xmat[{η, t}]) /. RSSRsubs].
      ({u, v, w, 1} /. numsols2use[[i]]), {i, Length[numsols2use]}], 10^-10];
  thetaAdj = Quiet[N[Table[(ArcTan[thetatransform[[i, 1]], thetatransform[[i, 2]]] /.
        numsols2use[[i]]), {i, Length[numsols2use]}]]] - thetas[[1]];
  ThetaAdjFinal = Table[thetaAdj[[i]] - (2 * π) * Floor[thetaAdj[[i]] / (2 * π)],
    {i, Length[thetaAdj]}];
  phitransform = Round[Table[Inverse[(Xmat[{γ1, g}]) /. RSSRsubs].
      ({x, y, z, 1} /. numsols2use[[i]]), {i, Length[numsols2use]}], 10^-10];
  phiAdj = Quiet[N[Table[ArcTan[phitransform[[i, 1]], phitransform[[i, 2]]],
       {i, Length[numsols2use]}]] - phis[[1]]];
  PhiAdjFinal = Table[phiAdj[[i]] - (2 * π) * Floor[phiAdj[[i]] / (2 * π)],
    {i, Length[phiAdj]}];
  {leglengths, uvwxyz2use, test, ThetaAdjFinal, PhiAdjFinal}]
```

The analysis is performed the same was as the planar four bar with some additional rotations and translations implemented.

1.) Initial check to skip the analysis if there are no solutions from the synthesis.
2.) Import the linkage information from the synthesis
3.) Check for degernate solutions. This is if the input crank or output link have lenghts of 0. This means that in any position it is possible to satisfy the input/output parameters.
4.) Calculate the displacement and link length values given the imported coordinates.
5.) Check each branch by subtracting the input phi values from the calculated phi values based off the theta.
6.) The solutions that do not branch are saved.
7.) solutions that nearly do not branch saved. these are saved for potential future optomization, but are not used currently.
8.) The pertenent information is packaged together for the output.

```
RSSRAnalysis[acbin_, uvwxyzin_, thetas_,
  phis_, gammas_, Const_, thetasAdjust_, phiAdjust_] :=
 Module[{len, RSSRsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols,
   Sols2use, coord2use, phiAdj2use, thetaAdj2use, Branch2use,
   finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen},
  finalsols = {}; Altfinalsols = {};
  If[Length[acbin] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  len = Length[uvwxyzin];
  Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
  uvwxyzsub = Table[Thread[{u, v, w, x, y, z} → uvwxyzin[[i]]], {i, len}];
  legsubs = Table[
    Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub[[i]], ((Cnew /. Constsub) /.
          γ → gammas[[1]]) /. uvwxyzsub[[i]], acbin[[i, 3]]}], {i, len}];
  If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
       Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
     {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
   Goto[noSolutions]];
  RSSRsubs = Table[Thread[
     Flatten[{g, d, s, t, g1, η, ψ}] → {Const[[1]], (Dnew /. Constsub) /. uvwxyzsub[[j]],
       ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub[[j]], Const[[4]],
       (G1new /. Constsub) /. uvwxyzsub[[j]], Const[[6]], Const[[7]]}], {j, len}];
  (*Print[RSSRsubs];*)
  Branch1 =
   Quiet[ N[Table[Mod[Round[Table[ (ϕeqn1 /. Flatten[{legsubs[[j]], RSSRsubs[[j]],
              γ → gammas[[i]], Θ → (thetas[[i]] + thetasAdjust[[j]])}]) -
          (phis[[i]] + phiAdjust[[j]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
  Branch2 = Quiet[N[Table[Mod[Round[Table[ (ϕeqn2 /. Flatten[{legsubs[[j]], RSSRsubs[[
               j]], γ → gammas[[i]], Θ → (thetas[[i]] + thetasAdjust[[j]])}]) -
          (phis[[i]] + phiAdjust[[j]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
  Alen = N[Table[Table[Norm[ ((Acoord /. Flatten[{legsubs[[j]], RSSRsubs[[j]]}]) /.
         {γ → gammas[[i]], Θ → (thetas[[i]] + thetasAdjust[[j]]),
```

```
          ϕ → (phis[[i]] + phiAdjust[[j]])}), {i, 7}], {j, len}]];
  Clen = N[Table[Table[Norm[(((Bcoord - Ccoord) /. Flatten[{legsubs[[j]], RSSRsubs[[
              j]]}]) /. {γ → gammas[[i]], Θ → (thetas[[i]] + thetasAdjust[[j]]),
          ϕ → (phis[[i]] + phiAdjust[[j]])})], {i, 7}], {j, len}]];

  (*Print[legsubs,uvwxyzsub];*)
  (*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
    ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ gammas[[1]]}]*)
  Branch1adj = N[Branch1];
  Branch2adj = N[Branch2];
  Branch1vals =
   Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
      0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
  Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
       Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
     {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
  (*Print[Branch1,Branch2];
  Print[Branch1vals,Branch2vals];*)
  ValidSols =
   Cases[Table[If[Or[Total[Branch1vals[[i]]] == 7, Total[Branch2vals[[i]]] == 7], i],
     {i, Length[acbin]}], _Integer];
  AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
     {i, Length[acbin]}], _Integer];
  Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == 7, 1,
     If[Total[Branch2vals[[ValidSols[[i]]]]] == 7, 2, 0]], {i, Length[ValidSols]}];

  Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  phiAdj2use = Table[phiAdjust[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  thetaAdj2use = Table[thetasAdjust[[ValidSols[[i]]]], {i, Length[ValidSols]}];

  finalsols = Table[Thread[{a, c, b, u, v, w, x, y, z, Θadj, ϕadj, braval} →
      Flatten[{Sols2use[[i]], coord2use[[i]], thetaAdj2use[[i]],
        phiAdj2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

  Altfinalsols = If[Length[AltSols] > 0, {thetas, phis, gammas}, {}];
  Lastsols = {finalsols, Altfinalsols};
  Label[noSolutions];
  Lastsols]
```

This test checks the continuity of each of the results that succuessfully come out of the analysis.

It divides the range of into 200 points and checks if the link lengths change at any of these points.

```
RSSRTest[inputs_, Const_, thetas_, phis_, gammas_, eqninputs_] :=
 Module[{len, Constsub, legsubs, uvwxyzsub, RSSRsubs, limit, Alen,
   Blen, Clen, TestedSols, TableOfO, TableOfA, TableOfB, TableOfC},
  Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
  uvwxyzsub = inputs[[4 ;; 9]];
  legsubs = Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub,
```

```
    ((Cnew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, (b /. inputs)}];
RSSRsubs = Thread[Flatten[{g, d, s, t, g1, η, ψ}] →
   Flatten[{Const[[1]], (Dnew /. Constsub) /. uvwxyzsub,
     ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, Const[[4]],
     (G1new /. Constsub) /. uvwxyzsub, Const[[6]], Const[[7]]}]];
limit = 200;
TableOfO =
 Table[Ocoord /. Flatten[{legsubs, RSSRsubs, θ → (thetas[[i]] + θadj /. inputs),
     ϕ → phis[[i]], γ → gammas[[i]]}], {i, 7}];
TableOfA = Table[Acoord /. Flatten[{legsubs, RSSRsubs,
     θ → (thetas[[i]] + θadj /. inputs), ϕ → phis[[i]], γ → gammas[[i]]}], {i, 7}];
TableOfB = Table[((Bcoord) /. {ϕ → (If[(braval /. inputs) == 1, ϕeqn1, ϕeqn2])}) /.
   Flatten[{legsubs, RSSRsubs,
     θ → (thetas[[i]] + θadj /. inputs), γ → gammas[[i]]}], {i, 7}];
TableOfC = Table[Ccoord /. Flatten[{legsubs, RSSRsubs,
     θ → (thetas[[i]] + θadj /. inputs), ϕ → phis[[i]], γ → gammas[[i]]}], {i, 7}];

(*Print["O Coordinates"];
Print[TableOfO];
Print["A Coordinates"];
Print[TableOfA];
Print["B Coordinates"];
Print[TableOfB];
Print["C Coordinates"];
Print[TableOfC];*)

Alen = N[Table[Norm[(((Acoord - Ocoord) /. Flatten[{legsubs, RSSRsubs}]) /.
      {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))})], {n, limit}]];
Blen = N[Table[Norm[((Acoord /. Flatten[{legsubs, RSSRsubs}]) /.
       {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) -
     (((((Bcoord) /. {ϕ → (If[(braval /. inputs) == 1, ϕeqn1, ϕeqn2])}) /.
           Flatten[{legsubs, RSSRsubs}]) /.
         {γ → (eqninputs[[4]] /. x → (eqninputs[[3]] * (n / limit)))}) /.
       θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs)))], {n, limit}]];
Clen = N[Table[Norm[(((((Bcoord - Ccoord) /. {ϕ → (If[(braval /. inputs) == 1,
                ϕeqn1, ϕeqn2])}) /. Flatten[{legsubs, RSSRsubs}]) /.
        {γ → (eqninputs[[4]] /. x → (eqninputs[[3]] * (n / limit)))}) /.
      θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs)))], {n, limit}]];
(*Print[{Alen,Blen,Clen}];*)
TestedSols = {inputs[[1 ;; 3]], Chop[{Alen, Blen, Clen} - ({a, b, c} /. legsubs)]};
TestedSols]
```

The plotting module plots the input vs the output. It currently still needs some work as sometimes there are "jumps" of 2π in teh plotting.

```
PlottingSolutions[inputs_, Const_, thetas_, phis_, gammas_, eqninputs_] :=
 Module[{len, Constsub, legsubs, uvwxyzsub,
   RSSRsubs, plotϕeqn1, plotϕeqn2, Points2Plot, limit, test1},
  Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
  uvwxyzsub = inputs[[4 ;; 9]];
  legsubs = Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub,
      ((Cnew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, (b /. inputs)}];
  RSSRsubs = Thread[Flatten[{g, d, s, t, g1, η, ψ}] →
     Flatten[{Const[[1]], (Dnew /. Constsub) /. uvwxyzsub,
       ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, Const[[4]],
       (G1new /. Constsub) /. uvwxyzsub, Const[[6]], Const[[7]]}]];
  plotϕeqn1 = (ϕeqn1 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  plotϕeqn2 = (ϕeqn2 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  Points2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];
  limit = 500;
  ListPlot[{Table[{eqninputs[[3]] * (n / limit),
      eqninputs[[1]] /. x → (eqninputs[[3]] * (n / limit))}, {n, limit}],
    Round[Table[{eqninputs[[3]] * (n / limit), (((plotϕeqn1) /. {γ → (eqninputs[[4]] /.
                x → (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))}) /.
          {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}}) - (ϕadj /. inputs)},
      {n, limit}], 10^-4], Round[Table[{eqninputs[[3]] * (n / limit),
       (((plotϕeqn2) /. {γ → (eqninputs[[4]] /.
                x → (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))}) /.
          {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}}) -
        (ϕadj /. inputs) + 2 * Pi}, {n, limit}], 10^-4], Points2Plot},
   PlotLegends → {"Input Function", "Branch 1", "Branch 2",
     "Precision Points"},
   Joined → {True, True, True, False},
   PlotStyle →
    {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
   PlotLabel → inputs[[1 ;; 3]]]

 ]
```

```
PlottingSolutions[inputs_, Const_, thetas_, phis_, gammas_, eqninputs_] :=
 Module[{len, Constsub, legsubs, uvwxyzsub, RSSRsubs, plotϕeqn1, plotϕeqn2,
   Points2Plot, limit, test1, InputFunction, Branch1plot, Branch2plot},
  Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
  uvwxyzsub = inputs[[4 ;; 9]];
  legsubs = Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub,
      ((Cnew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, (b /. inputs)}];
  RSSRsubs = Thread[Flatten[{g, d, s, t, g1, η, ψ}] →
     Flatten[{Const[[1]], (Dnew /. Constsub) /. uvwxyzsub,
       ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, Const[[4]],
       (G1new /. Constsub) /. uvwxyzsub, Const[[6]], Const[[7]]}]];
  limit = 500;
  plotϕeqn1 = (ϕeqn1 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  plotϕeqn2 = (ϕeqn2 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  Points2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];
  InputFunction = Table[{eqninputs[[3]] * (n / limit),
     eqninputs[[1]] /. x → (eqninputs[[3]] * (n / limit))}, {n, limit}];
  Branch1plot = Round[Table[{eqninputs[[3]] * (n / limit),
      (((plotϕeqn1) /. {γ → (eqninputs[[4]] /. x →
                (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))}) /.
        {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) -
       (ϕadj /. inputs)}, {n, limit}], 10^-4];
  Branch2plot = Round[Table[{eqninputs[[3]] * (n / limit),
      (((plotϕeqn2) /. {γ → (eqninputs[[4]] /. x →
                (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))}) /.
        {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) -
       (ϕadj /. inputs) + 2 * Pi}, {n, limit}], 10^-4];

  ListPlot[{InputFunction, Branch1plot, Branch2plot, Points2Plot},
   PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
   Joined → {True, True, True, False},
   PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
   PlotLabel → inputs[[1 ;; 3]]]

 ]
```

```
CompareSolution[inputs_, Const_, thetas_, phis_, gammas_, eqninputs_] :=
  Module[{len, Constsub, legsubs, uvwxyzsub, RSSRsubs, plotϕeqn1,
    plotϕeqn2, Points2Plot, limit, test1, InputFunction, Branch1plot,
    Branch2plot, Branch1points, Branch2points, θpts, γpts},
   Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
   uvwxyzsub = inputs[[4 ;; 9]];
   legsubs = Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub,
       ((Cnew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, (b /. inputs)}];
   limit = 200;
   RSSRsubs =
    Thread[Flatten[{g, d, s, t, g1, η, ψ}] → Flatten[{Const[[1]], (Dnew /. Constsub) /.
         uvwxyzsub, ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub,
       Const[[4]], (G1new /. Constsub) /. uvwxyzsub, Const[[6]], Const[[7]]}]];
   plotϕeqn1 = (ϕeqn1 /. Flatten[{legsubs, RSSRsubs}]);
   plotϕeqn2 = (ϕeqn2 /. Flatten[{legsubs, RSSRsubs}]);
   Points2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];

   θpts = Table[eqninputs[[3]] * (n / limit) + (θadj /. inputs), {n, limit}];
   γpts = Table[(eqninputs[[4]] /. Flatten[{legsubs, RSSRsubs}]) /.
      {x → θpts[[n]]}, {n, limit}];
   InputFunction = Table[eqninputs[[1]] /. {x → θpts[[n]]}, {n, limit}];
   Branch1points =
    Table[(((plotϕeqn1) /. {γ → (eqninputs[[4]] /. x → (eqninputs[[3]] *
                (n / limit) (*+(θadj/.inputs)*)))}) /.
       {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) - (ϕadj /.
       inputs), {n, limit}];
   Branch2points = Table[(((plotϕeqn2) /. {γ → (eqninputs[[4]] /.
             x → (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*)))}) /.
       {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) -
      (ϕadj /. inputs) + 2 * Pi, {n, limit}];

   Total[MapThread[(#1 - #2) ^ 2 &,
     {If[(braval /. inputs) == 1, Branch1points, Branch2points], InputFunction}]]


  ];
```

The plotting module to comapare the SolidWorks data to the generated solution

```
PlottingSWCompare[inputs_, Const_, thetas_, phis_, gammas_,
  eqninputs_, SWin_, SWout_] := Module[{len, Constsub, legsubs,
   uvwxyzsub, RSSRsubs, plotϕeqn1, plotϕeqn2, Points2Plot, limit, test1},
  Constsub = Thread[{g, d, s, t, g1, η, ψ} → Const];
  uvwxyzsub = inputs[[4 ;; 9]];
  legsubs = Thread[{a, c, b} → {(Anew /. Constsub) /. uvwxyzsub,
      ((Cnew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, (b /. inputs)}];
  RSSRsubs = Thread[Flatten[{g, d, s, t, g1, η, ψ}] →
     Flatten[{Const[[1]], (Dnew /. Constsub) /. uvwxyzsub,
       ((Snew /. Constsub) /. γ → gammas[[1]]) /. uvwxyzsub, Const[[4]],
       (G1new /. Constsub) /. uvwxyzsub, Const[[6]], Const[[7]]}]];
  plotϕeqn1 = (ϕeqn1 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  plotϕeqn2 = (ϕeqn2 /. Flatten[{legsubs, RSSRsubs}])
  (*/.γ→ (eqninputs[[4]]/.x→ θ)*);
  Points2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];
  limit = 500;
  ListPlot[{Table[{SWin[[i]], SWout[[i]] (*+(ϕadj/.inputs)*)}, {i, Length[SWin]}],
    Table[{eqninputs[[3]] * (n / limit),
      eqninputs[[1]] /. x → (eqninputs[[3]] * (n / limit))}, {n, limit}],
    Round[Table[{eqninputs[[3]] * (n / limit), (((plotϕeqn1) /. {γ → (eqninputs[[4]] /.
                x → (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))} /.
          {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) - (ϕadj /. inputs)},
      {n, limit}], 10^-4], Round[Table[{eqninputs[[3]] * (n / limit),
       (((plotϕeqn2) /. {γ → (eqninputs[[4]] /.
                x → (eqninputs[[3]] * (n / limit) (*+(θadj/.inputs)*))))} /.
          {θ → (eqninputs[[3]] * (n / limit) + (θadj /. inputs))}) -
        (ϕadj /. inputs) + 2 * Pi}, {n, limit}], 10^-4], Points2Plot},
   PlotLegends → {"SolidWorks Output", "Ideal Wing Pitch", "Branch 1",
     "Branch 2", "Precision Points"},
   Joined → {True, True, True, True, False},
   PlotStyle →
    {Thick, Dashing[Medium], Dashing[Tiny], Dashing[Small],
     PointSize[Large]}, PlotLabel → inputs[[1 ;; 3]]]
  (*Print[FindMinimum[(((plotϕeqn2)/.{γ→ (eqninputs[[4]]/.x→ θ)})/.
       {θ→ (θ+(θadj/.inputs))})-(ϕadj/.inputs)+2*Pi,{θ,Pi/2}]]*)
 ]
```

The randomizing module randomizes the inputs giving 7 random numbers within a given tolerance

```
RandomizeTps[thetaAngles_, psiAngles_, ipTolerances_, opTolerances_] :=
  Module[{thetasNew, psisNew},

   thetasNew = psisNew = ConstantArray[0, 7];
   thetasNew = Table[RandomReal[{(thetaAngles[[i]] - ipTolerances[[i]]),
        (thetaAngles[[i]] + ipTolerances[[i]])}], {i, 7}];
   thetasNew[[1]] = If[thetasNew[[1]] < 0,
     RandomReal[{0, (thetaAngles[[1]] + ipTolerances[[1]])}], thetasNew[[1]]];
   thetasNew[[7]] = If[thetasNew[[7]] > 2 * Pi,
     RandomReal[{(thetaAngles[[7]] - ipTolerances[[7]]), 2 * Pi}], thetasNew[[7]]];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
         (psiAngles[[i]] + opTolerances[[i]])}], {i, 7}], 2 * Pi];

   {thetasNew, psisNew}];
```

The Chebyshev Spacing module chooses the precision points to design around.

```
ChebyshevSpacing[NoOfPrecisionPts_, x0_, xnplus1_] := Module[{PrecisionPts, xn, j},
  PrecisionPts = ConstantArray[0, NoOfPrecisionPts + 2];
  PrecisionPts[[1]] = x0;
  PrecisionPts[[-1]] = xnplus1;
  (*Print[PrecisionPts];*)
  xn = 0.5 * (x0 + xnplus1) -
    0.5 * (xnplus1 - x0) * Cos[(π * (2 * j - 1)) / (2 * NoOfPrecisionPts)];
  Table[
   PrecisionPts[[i + 1]] = xn /. j → i;
   (*Print[PrecisionPts];*)
   , {i, NoOfPrecisionPts}];

  PrecisionPts
 ]
```

User Inputs

Equations

Planar Crank Rocker Equation

This section derives the equations for a planar crank rocker to drive the second input of the spatial 5 bar.

NOTE: An the offset to the input angle must be negated to account for moving from the xy plane to the yz plane. The output is supposed to control the angle of the Z axis as the wing flaps it rotates about the x axis.

Inputs

```
PlanarTransAngle = 29.9 * Degree; (* Transmission Angle*)
PlanarSwingAngle = 120 * Degree; (* Total Swing Angle *)
PlanarStartAngle = 30 * Degree; (* Initial Position of the output*)
PlanarBaseLen = 5; (* Length of the base (g link) for the planar mechanism *)
PlanarCrankMax = 180 * Degree;
(* This is the desired input angle corresponding to the maximum output *)
```

LinkLength Ratios where {r1,r2,r3,r4} = {g,a,b,c} with g being the base length

```
r3r1 = Sqrt[(1 - Cos[PlanarSwingAngle]) / (2 * Cos[PlanarTransAngle]^2)];
r4r1 = Sqrt[(1 - r3r1^2) / (1 - r3r1^2 * Cos[PlanarTransAngle]^2)];
r2r1 = Sqrt[r3r1^2 + r4r1^2 - 1];
PlanarSubs = N[Thread[{g, a, b, c} → {1, r2r1, r3r1, r4r1} * PlanarBaseLen]]
```

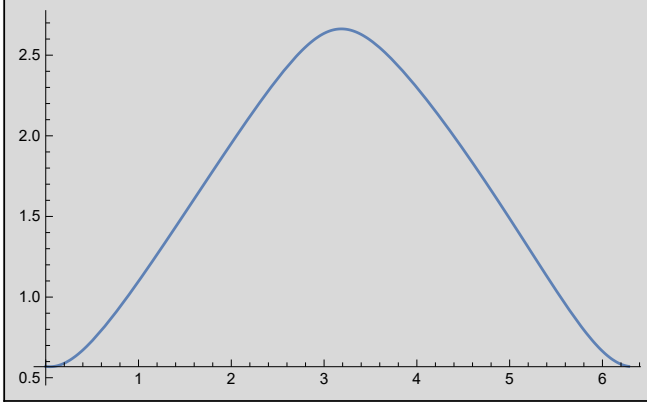
```
{g → 5., a → 0.388193, b → 4.99497, c → 0.448247}
```

Define the Initial γ equation through substitution

```
Initialγeqn = Planarϕeqn1 /. Flatten[{PlanarSubs, θ → x, η → 0, g1 → 0, ψ → 0}]
```

```
ArcCos[(1.11546 (-0.401851 + 3.88193 Cos[x])) /
    (√(25. - 3.88193 Cos[x] + 0.150694 Cos[x]^2 + 0.150694 Sin[x]^2))] +
 ArcTan[0.448247 (5. - 0.388193 Cos[x]), 0.448247 (0. - 0.388193 Sin[x])]
```

```
Plot[Initialγeqn, {x, 0, 2 * Pi}]
```



Find the Initial Maxs and Mins of the Planar Equation

```
InitialMax = FindMaximum[{Initialγeqn, 0 ≤ x ≤ 2 * Pi}, {x, Pi}]
```

```
{2.66283, {x → 3.18643}}
```

```
InitialMin = FindMinimum[{Initialγeqn, -Pi ≤ x ≤ Pi}, {x, 0}]
```

```
{0.568438, {x → 0.0448397}}
```

```
0.04483970872350103 * 180 / Pi
```

```
2.56913
```

```
InitialMin[[1]] - Pi / 6
```

```
0.0448397
```

Calculate the vertial/output (γ) offset to shift to the desired output values

```
Planarγoffset = (InitialMax[[1]] - (PlanarSwingAngle + PlanarStartAngle))
```

```
0.0448397
```

```
Planarγoffset * 180 / Pi
```

```
2.56913
```

Calculate the Horizontal/input (θ) Offset to shift to the desired corresponding input values

```
Planarθoffset = (x /. InitialMax[[2]]) - PlanarCrankMax
```

```
0.0448397
```

```
Planarθoffset * 180 / Pi
```

```
2.56913
```

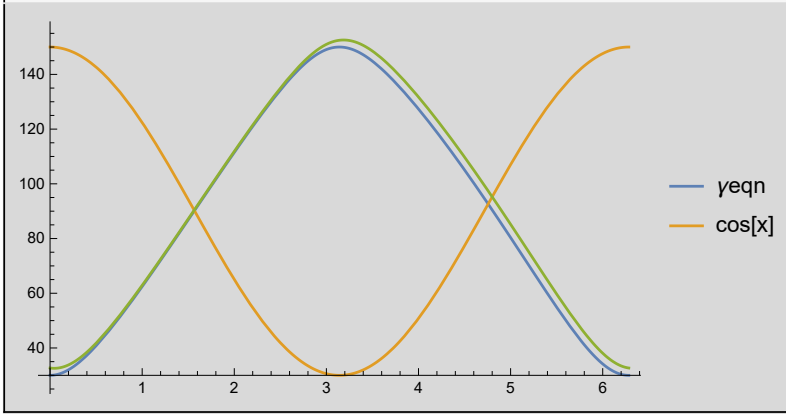
```
γeqn =
 (Planarϕeqn1 /. Flatten[{PlanarSubs, θ → (x + Planarθoffset), η → 0, g1 → 0, ψ → 0}]) -
  Planarγoffset
```

```
-0.0448397 + ArcCos[(1.11546 (-0.401851 + 3.88193 Cos[0.0448397 + x])) /
    (√(25. - 3.88193 Cos[0.0448397 + x] +
        0.150694 Cos[0.0448397 + x]^2 + 0.150694 Sin[0.0448397 + x]^2))] +
 ArcTan[0.448247 (5. - 0.388193 Cos[0.0448397 + x]),
  0.448247 (0. - 0.388193 Sin[0.0448397 + x])]
```

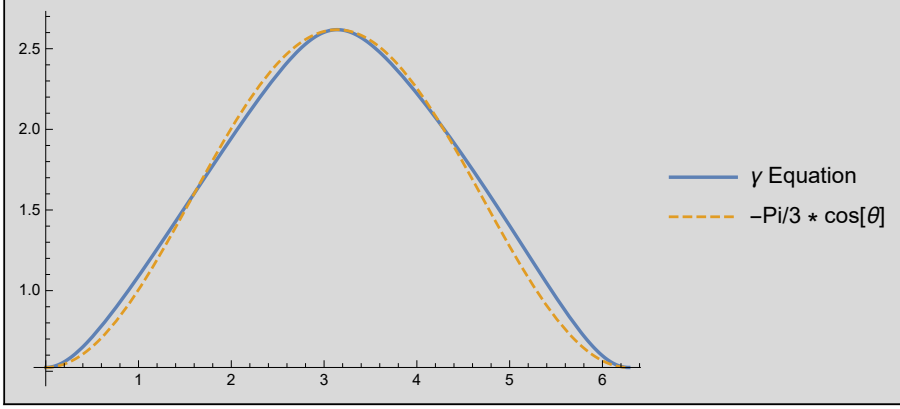
```
γnoshift = Planarϕeqn1 /. Flatten[{PlanarSubs, θ → (x), η → 0, g1 → 0, ψ → 0}]
```

```
ArcCos[(1.11546 (-0.401851 + 3.88193 Cos[x])) /
    (√(25. - 3.88193 Cos[x] + 0.150694 Cos[x]^2 + 0.150694 Sin[x]^2))] +
 ArcTan[0.448247 (5. - 0.388193 Cos[x]), 0.448247 (0. - 0.388193 Sin[x])]
```

```
PlanarPlotDeg =
 Plot[{γeqn * 180 / Pi, (60 * Degree * Cos[x] + 90 * Degree) * 180 / Pi, γnoshift * 180 / Pi},
  {x, 0, 2 * Pi}, PlotLegends → {"γeqn", "cos[x]"}]
```




```
PlanarPlotRad = Plot[{γeqn, (-60 * Degree * Cos[x] + 90 * Degree)}, {x, 0, 2 * Pi},
  PlotLegends → {"γ Equation", "-Pi/3 * cos[θ]"}, PlotStyle → {Thick, Dashed}]
```



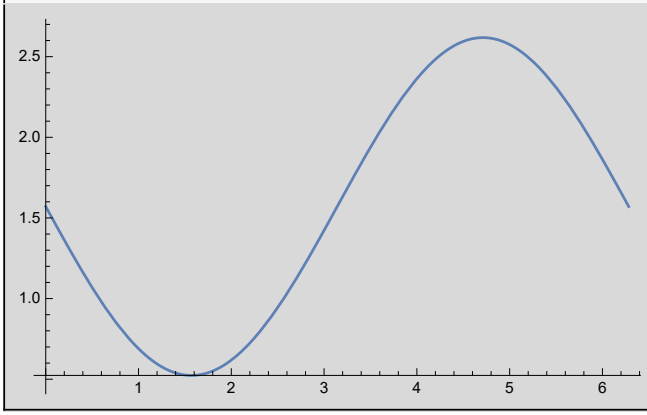
Other Inputs

If a Constant γ angle is desired setting a constant value here will set this.

```
(*γeqn = 60*Degree*Cos[x]; (*For wing stroke*)*)
```

```
yeqn = π / 2 - 60 * Degree * Sin[x]; (*for wing pitch*)
```

```
Plot[yeqn, {x, 0, 2 * Pi}]
```



Define the Lower and Upper Bounds for the input crank (assuming same input crank is driving γ as is driving ϕ

```
Lowerb = 0 * Degree; Upperb = 2 * Pi; (* for wing pitch*)
```

```
Eqnvals = {yeqn, Lowerb, Upperb, γeqn};
```

Applying Chebyshev Spacing for the x values of the function generation

```
precisionptsX = Chop[N[ChebyshevSpacing[5, Lowerb, Upperb]]]
```

```
{0, 0.15376, 1.29501, 3.14159, 4.98817, 6.12942, 6.28319}
```

Substituting the calculated x values into the Y equation to generate the 7 precision points

```
precisionptsY = Chop[N[yeqn /. x → precisionptsX]]
```

```
{1.5708, 1.41041, 0.563171, 1.5708, 2.57842, 1.73118, 1.5708}
```

```
precisionptsγ = Chop[N[γeqn /. x → precisionptsX]]
```

```
{0.523599, 0.544819, 1.33872, 2.61799, 1.41288, 0.545676, 0.523599}
```

```
Thetavals = precisionptsX
Phivals = precisionptsY
γvals = precisionptsγ
```

```
{0, 0.15376, 1.29501, 3.14159, 4.98817, 6.12942, 6.28319}
```

```
{1.5708, 1.41041, 0.563171, 1.5708, 2.57842, 1.73118, 1.5708}
```

```
{0.523599, 0.544819, 1.33872, 2.61799, 1.41288, 0.545676, 0.523599}
```

Set the Tolerances for the randomization. Set the beginning and end to 0 so that the function is forced to have continuous end points

```
(*ThetaTols =N[Array[15&,7]Degree];*)
ThetaTols = {0, 15, 15, 15, 15, 15, 0} * Degree
```

```
{0, 15 °, 15 °, 15 °, 15 °, 15 °, 0}
```

```
(*PhiTols=N[Array[15&,7]Degree]*)
PhiTols = {0, 15, 15, 15, 15, 15, 0} * Degree
```

```
{0, 15 °, 15 °, 15 °, 15 °, 15 °, 0}
```

Set the Constant Values

```
gval = 0;
dval = -5;
sval = 0;
tval = 0;
g1val = 0;
ηval = Pi / 2;
ψval = Pi / 2;
```

```
Constvals = {gval, dval, sval, tval, g1val, ηval, ψval}
```

$$\left\{0, -5, 0, 0, 0, \frac{\pi}{2}, \frac{\pi}{2}\right\}$$

Solver

Generation Code
Inputs: Number of Iterations
The first results tested are the initial input and output values
The associated γ values are calculated from the θ values
Synthesis and Analysis are run
If solutions come out of the Analysis they are saved into an array
The input (θ) and output (ϕ) are randomized and the For loop iterates

The zero and null values are removed from the Output array
The exact solutions array is built, the near solutions array is built

```
Iterations = 500;
Output = Array[0 &, Iterations];
NumberOfSols = 0;
RandInputs = Array[0 &, Iterations];
k = 1;
Randvals = {Thetavals, Phivals};
For[n = 0, n < Iterations, n++, Print[n];
 Randγ = Table[γeqn /. x → Randvals[[1, i]], {i, 7}];
 (*Calculate associated γ values for the given thetas*)
 Randsyn = Chop[RSSRSynthesis[Randvals[[1]], Randvals[[2]], Randγ, Constvals]];
 (*RSSR Synethesis*)
 NumberOfSols = NumberOfSols + Length[Randsyn[[1]]];
 Randsol =
  Chop[RSSRAnalysis[Randsyn[[1]], Randsyn[[2]], Randvals[[1]], Randvals[[2]],
    Randγ, Constvals, Randsyn[[4]], Randsyn[[5]]]]; (*RSSR Analysis*)
 If[Randsol == {{}, {}}, , Output[[k]] = Randsol;
  RandInputs[[k]] = Randvals;
  k++;]; (*Save values that Pass Analysis*)
 Randvals = RandomizeTps[Thetavals, Phivals, ThetaTols, PhiTols];];
(*Set new random values*)
Print[Output];
Output = Output /. {h__, (0 | 0. | Null) ...} :> {h};
(*RandInputs=RandInputs/.{h__,(0|0.|Null)...}:>{h}
     testa = Flatten[Table[Table[{Output[[i,j]],RandInputs[[i]]},
     {j,Length[Output[[i]]]}],{i,Length[Output]}],1]
*)
testa = DeleteCases[Output, {}];
ExactSols = Array[0 &, Length[testa]];
NearSols = Array[0 &, Length[testa]];
 For[i = 1, i < 1 + Length[testa], i++, Print[testa[[i, 1]]];
 If[testa[[i, 1]] == {}, NearSols[[i]] = testa[[i, 2]], ExactSols[[i]] = testa[[i]]];]
ExactSols = DeleteCases[ExactSols, 0];
NearSols = DeleteCases[NearSols, 0];
Print["Number of Solutions: ", NumberOfSols];
```

Plots only continuous results

1) TableofExactSols: Generates a table with the number of solutions per random input/output set (thetas and phis) there are a maximum of 20 solutions per set

2) TestedSols: Generates a table that shows the change in link length for 200 input angles which are linearly interpolated between the maximum and minimum angles

3) PassedTestTable: Generates a table showing which sets had zero change in link length

4) PassedSols: Generates a table with all the information for each continuous set

5) Plots the continuous results.

```
Table[Table[{{i, j}, PlottingSolutions[ExactSols[[i, 1, j]], Constvals,
    ExactSols[[i, 2, 1]], ExactSols[[i, 2, 2]], ExactSols[[i, 2, 3]], Eqnvals]},
  {j, TableofExactSols[[i]]}], {i, Length[ExactSols]}]
```

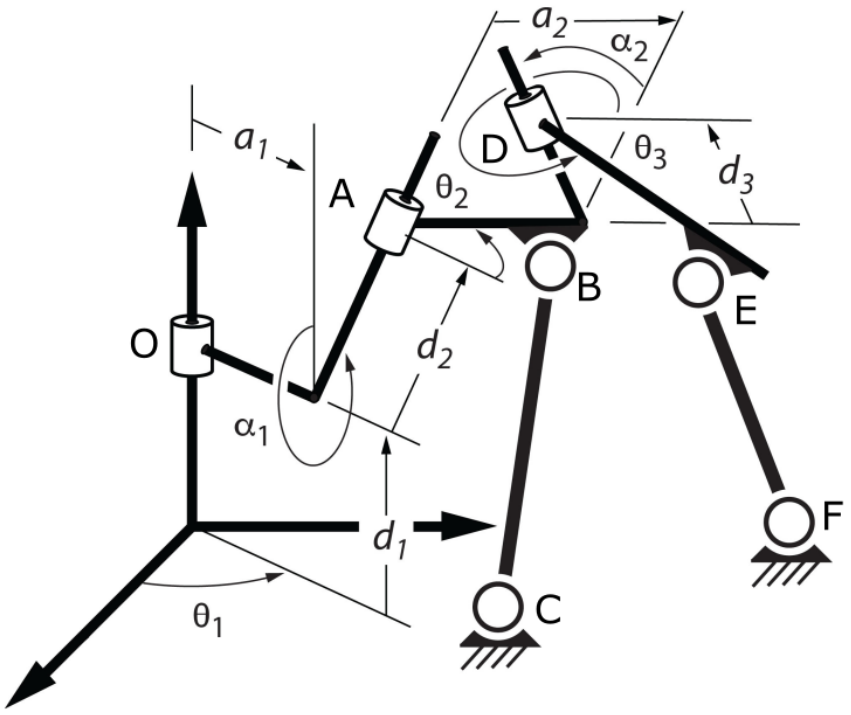
```
TableofExactSols = Table[Length[ExactSols[[i, 1]]], {i, Length[ExactSols]}];
TestedSols =
  Table[Table[{{i, j}, RSSRTest[ExactSols[[i, 1, j]], Constvals, ExactSols[[i, 2, 1]],
      ExactSols[[i, 2, 2]], ExactSols[[i, 2, 3]], Eqnvals]},
    {j, TableofExactSols[[i]]}], {i, Length[ExactSols]}];
PassedTestTable = DeleteCases[Flatten[
    Table[Table[If[Total[Flatten[TestedSols[[i, j, 2, 2]]]] == 0, {i, j}, 0],
      {j, TableofExactSols[[i]]}], {i, Length[ExactSols]}], 1], 0];
PassedSols = Table[{ExactSols[[PassedTestTable[[i, 1]], 1, PassedTestTable[[i, 2]]]],
    ExactSols[[PassedTestTable[[i, 1]], 2]]}, {i, Length[PassedTestTable]}];
Print["Number of Solultions that Passed Branching: ", Total[TableofExactSols]];
Print["Number of Solutions that Passed Continuity: ", Length[PassedSols]];
Table[PlottingSolutions[PassedSols[[i, 1]], Constvals, PassedSols[[i, 2, 1]],
  PassedSols[[i, 2, 2]], PassedSols[[i, 2, 3]], Eqnvals], {i, Length[PassedSols]}]
```

```
PassedSols
```

Appendix A

RRR-2SS Flapping Wing Mechanism Mathematica Code

Spatial Six Bar Stephenson



Multithreading SetUp

```
ParallelEvaluate[$KernelCount]
```

```
{16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16}
```

```
Kernels[]
```

```
{KernelObject[1, local], KernelObject[2, local],
 KernelObject[3, local], KernelObject[4, local], KernelObject[5, local],
 KernelObject[6, local], KernelObject[7, local], KernelObject[8, local],
 KernelObject[9, local], KernelObject[10, local], KernelObject[11, local],
 KernelObject[12, local], KernelObject[13, local],
 KernelObject[14, local], KernelObject[15, local], KernelObject[16, local]}
```

```
$ProcessorCount
```

```
16
```

Basic Functions and Constants

These equations are all calculated outside the module to reduce the number of repeated calculations

```
Zmat[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], 0, 0},
   {Sin[x[[1]]], Cos[x[[1]]], 0, 0}, {0, 0, 1, x[[2]]}, {0, 0, 0, 1}};
Xmat[x_] := {{1, 0, 0, x[[2]]}, {0, Cos[x[[1]]], -Sin[x[[1]]], 0},
   {0, Sin[x[[1]]], Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Ymat[x_] := {{Cos[x[[1]]], 0, Sin[x[[1]]], 0}, {0, 1, 0, x[[2]]},
   {-Sin[x[[1]]], 0, Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Disp[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], x[[2]]},
   {Sin[x[[1]]], Cos[x[[1]]], x[[3]]}, {0, 0, 1}};
rem = Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}];
```

Synthesis Equations

These are the equations that will be used in the synthesis.

θ: input values Equivalent to $\theta 1$

ϕ: output values equavalent to $\theta 2$

γ: moving revolute joint equivalent to $\theta 3$

```
RSSRin = {{θ1, θ2, θ3, θ4, θ5, θ6, θ7},
   {γ1, γ2, γ3, γ4, γ5, γ6, γ7}, {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7}};
```

Input Constants

```
d1 =.; d2 =.; d3 =.; a1 =.; a2 =.; α1 =.; α2 =.;
```

Homogeneous Transforms to get to points O, A, D

```
Omat = Table[Zmat[{RSSRin[[1, i]], d1}], {i, 7}];
```

```
Amat =
  Table[Zmat[{RSSRin[[1, i]], d1}].Xmat[{α1, a1}].Zmat[{RSSRin[[2, i]], d2}], {i, 7}];
```

```
Dmat = Table[Zmat[{RSSRin[[1, i]], d1}].Xmat[{α1, a1}].
    Zmat[{RSSRin[[2, i]], d2}].Xmat[{α2, a2}].Zmat[{RSSRin[[3, i]], d3}], {i, 7}];
```

Analysis Equations

These are the equations that will be used in the Analysis.

Analysis Equation 1 - first four bar with and revolute joints AB and SS joint BC

```
analysiseqn1 = ExpandAll[TrigExpand[Dot[{u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}, {u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}]] - b^2];
Acoeff1 = Coefficient[analysiseqn1, Cos[γ]];
Bcoeff1 = Coefficient[analysiseqn1, Sin[γ]];
Ccoeff1 = Simplify[analysiseqn1 - Acoeff1 * Cos[γ] - Bcoeff1 * Sin[γ]];
γeqn1 = ArcTan[Acoeff1, Bcoeff1] + ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
γeqn2 = ArcTan[Acoeff1, Bcoeff1] - ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
```

These are the equations for converting the initial coordinates of the spherical joint B to the frame of revolute joint A.

B = (x1,y1,z1)
C = (u1,v1,w1)

```
NewUVW1 = {u1, v1, w1};
```

```
NewXYZ1 = Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}]].{x1, y1, z1, 1}.rem;
```

Analysis Equation 2 - second four bar with revolute joints BD and SS joints EF

```
analysiseqn2 = ExpandAll[
   TrigExpand[Dot[{u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1},
      {u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1}]] - b^2];
Acoeff2 = Coefficient[analysiseqn2, Cos[ϕ]];
Bcoeff2 = Coefficient[analysiseqn2, Sin[ϕ]];
Ccoeff2 = Simplify[analysiseqn2 - Acoeff2 * Cos[ϕ] - Bcoeff2 * Sin[ϕ]];
ϕeqn1 = ArcTan[Acoeff2, Bcoeff2] + ArcCos[-Ccoeff2 / Sqrt[Acoeff2^2 + Bcoeff2^2]];
ϕeqn2 = ArcTan[Acoeff2, Bcoeff2] - ArcCos[-Ccoeff2 / Sqrt[Acoeff2^2 + Bcoeff2^2]];
```

These equations convert the coordinates of spherical joint E from the global frame to the frame of revolute joint D

E = (x2,y2,z2)
F = (u2,v2,w2)

```
NewUVW2 = {u2, v2, w2};
```

```
NewXYZ2 =
  Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}]].
   {x2, y2, z2, 1}.rem;
```

Continuity Test Equations

Location of each of the joints

```
Ocoord = Zmat[{θ, d1}].{0, 0, 0, 1};
Acoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{0, 0, 0, 1};
Bcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1};
Ccoord = {u1, v1, w1, 1};
Dcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].
   Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].{0, 0, 0, 1};
Ecoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
   Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1};
Fcoord = {u2, v2, w2, 1};
```

```
Dcoord
```

```
{a1 Cos[θ] + d2 Sin[α1] Sin[θ] + a2 (Cos[γ] Cos[θ] - Cos[α1] Sin[γ] Sin[θ]) +
   d3 (Cos[α2] Sin[α1] Sin[θ] - Sin[α2] (-Cos[θ] Sin[γ] - Cos[α1] Cos[γ] Sin[θ])),
 -d2 Cos[θ] Sin[α1] + a1 Sin[θ] + a2 (Cos[α1] Cos[θ] Sin[γ] + Cos[γ] Sin[θ]) +
   d3 (-Cos[α2] Cos[θ] Sin[α1] - Sin[α2] (Cos[α1] Cos[γ] Cos[θ] - Sin[γ] Sin[θ])),
 d1 + d2 Cos[α1] + d3 (Cos[α1] Cos[α2] - Cos[γ] Sin[α1] Sin[α2]) + a2 Sin[α1] Sin[γ], 1}
```

OA link length

```
OAlink = Norm[Ocoord.rem - Acoord.rem];
```

AB link length

```
ABlink = Norm[Acoord.rem - Bcoord.rem];
```

AD Link length

```
ADlink = Norm[Dcoord.rem - Acoord.rem];
```

DE Link length

```
DElink = Norm[Ecoord.rem - Dcoord.rem];
```

OC Link Length

```
OClink = Norm[Ocoord.rem - Ccoord.rem];
```

OF Link Length

```
OFlink = Norm[Ocoord.rem - Fcoord.rem];
```

BD Link Length

```
BDlink = Norm[Bcoord.rem - Dcoord.rem];
```

FC Link Length

```
FClink = Norm[Fcoord.rem - Ccoord.rem];
```

Functions

Other Modules

The Chebyshev Spacing module chooses the precision points to design around.

```
ChebyshevSpacing[NoOfPrecisionPts_, x0_, xnplus1_] := Module[{PrecisionPts, xn, j},
  PrecisionPts = ConstantArray[0, NoOfPrecisionPts + 2];
  PrecisionPts[[1]] = x0;
  PrecisionPts[[-1]] = xnplus1;
  (*Print[PrecisionPts];*)
  xn = 0.5 * (x0 + xnplus1) -
    0.5 * (xnplus1 - x0) * Cos[(π * (2 * j - 1)) / (2 * NoOfPrecisionPts)];
  Table[
   PrecisionPts[[i + 1]] = xn /. j → i;
   (*Print[PrecisionPts];*)
   , {i, NoOfPrecisionPts}];

  PrecisionPts
 ]
```

The randomizing module randomizes the inputs giving 7 random numbers within a given tolerance

```
RandomizeTps[thetaAngles_, gammaAngles_, psiAngles_, ipTolerances_,
   gammaTolerances_, opTolerances_] := Module[{thetasNew, gammasNew, psisNew},

   thetasNew = psisNew = gammasNew = ConstantArray[0, 7];
   thetasNew = Table[RandomReal[{(thetaAngles[[i]] - ipTolerances[[i]]),
       (thetaAngles[[i]] + ipTolerances[[i]])}], {i, 7}];
   thetasNew[[1]] = If[thetasNew[[1]] < 0,
     RandomReal[{0, (thetaAngles[[1]] + ipTolerances[[1]])}], thetasNew[[1]]];
   thetasNew[[7]] = If[thetasNew[[7]] > 2 * Pi,
     RandomReal[{(thetaAngles[[7]] - ipTolerances[[7]]), 2 * Pi}], thetasNew[[7]]];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
        (psiAngles[[i]] + opTolerances[[i]])}], {i, 7}], 2 * Pi];

   gammasNew = Mod[Table[RandomReal[{(gammaAngles[[i]] - gammaTolerances[[i]]),
        (gammaAngles[[i]] + gammaTolerances[[i]])}], {i, 7}], 2 * Pi];

   {thetasNew, gammasNew, psisNew}];
```

Randomizing only the phi angles

```
RandomizeTps1[psiAngles_, opTolerances_] := Module[{psisNew},

   psisNew = ConstantArray[0, 7];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
         (psiAngles[[i]] + opTolerances[[i]])}], {i, 7}], 2 * Pi];


   psisNew];
```

Calculate the Common Normal Lengths, Angular displacement and displacement along rotation axis find two d, one a, and one α.

Note that when the dot product is 0 the value of α is assigned to $\pi/2$.
ptc = p+tprime*S1
ptr = q+sprime*S2

```
CommonNorm[p_, S1_, q_, S2_] :=
  Module[{Cnorm, tprime, sprime, ptc, ptr, d1pt, d2pt, d1out, d2out, aout, αout},
   Cnorm = Normalize[Cross[S1, S2]];
   If[Cnorm == {0, 0, 0},
    d1out = 0;
    aout = 0;
    αout = 0;
    d2out = 0;
    d1pt = p;
    d2pt = q;
    Goto[SameLine];,
    tprime = Dot[Cross[(q - p), S2], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    sprime = Dot[Cross[(q - p), S1], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    ptc = p + tprime * S1;
    ptr = q + sprime * S2;
    d2pt = ptr;
    d1pt = ptc;
    d1out = (*Sign[Dot[S1,ptc-q]]**)Norm[p - ptc];
    d2out = (*Sign[Dot[S2,q-ptr]]**)Norm[ptr - q];
    aout = (*Sign[Dot[Cnorm,ptc-ptr]]**)Norm[ptc - ptr];
    If[Dot[S1, S2] == 0,
     αout = Pi / 2,
     αout = ArcTan[Dot[Cross[S1, S2], Cnorm] / Dot[S1, S2]]];];
   Label[SameLine];
   {d1out, aout, αout, d2out, d1pt, d2pt}];
```

Synthesis Modules

The synthesis modules below is very similar to that of the planar four bar synthesis method by khaus-tub. It follows the following steps

1.) Collect the input values and constants.
2.) Substitute into the homogeneous transform matricies
3.) Calculate the Relative Transform Matricies which are relative to the location of the first point (transform matrix)
4.) Define the coordinates of the moving pivots and stationary pivot which are being solved for. (W and G respectively)
5.) Set up the Constrain Equations. The link length between the two moving pivots must remain constant. Using the relative transform matricies the two defined coordiantes can be moved through space.
6.) Set up the Design Equations. Subtract the first constraint equation from the remaining constraint equations to remove the unknown link length variable.
7.) Solve the design equations. For a spatial four bar function generator there are seven sets of inputs and outputs, so there will be six design equations which coorrelate with the six unknown coordinates that need to be solved for.
8.) Keep only the real numbered solutions.
9.) Calculate the distance between the points. (I call this leg lengths, but this is not actually the leg lengths used in the Analysis. Only the legth between the points remains the same. The input crank and output are actually the distance the respective points are from the axis of rotation. I use the leg lengths as the "title" for each set of points.)
10.) The solutions of of the coordinates and "leg lengths" are stored.
11.) The solutions are plugged back into the constrain equations to ensure that all values are returned as zero.

```
FirstRSSRSynthesis[thetas_, gammas_, Const_] :=
 Module[{RSSRsubs, Ainput2useB, RSSRrelA, W, x, y, z, G, u, v, w,
   a, c, b, R, RSSRConstraintEqn, RSSRDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSRsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6, θ7}, {γ1, γ2, γ3, γ4, γ5, γ6, γ7},
       {d1, d2, d3, a1, a2, α1, α2}}] → Flatten[{thetas, gammas, Const}]];
  Ainput2useB = Amat /. RSSRsubs;
  RSSRrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  RSSRConstraintEqn =
   Table[Chop[Dot[G - RSSRrelA[[i]].W, G - RSSRrelA[[i]].W] - R^2], {i, 7}];
  RSSRDesignEqn = Table[Chop[Expand[
      RSSRConstraintEqn[[i + 1]] - RSSRConstraintEqn[[1]]]], {i, 6}];
  numsols = NSolve[RSSRDesignEqn == {0, 0, 0, 0, 0, 0},
    {u, v, w, x, y, z}, Method → "Legacy"];
  (*Print[Chop[RSSRrelA]];
  Print[RSSRConstraintEqn];
  Print[RSSRDesignEqn];
  Print[numsols];*)
  (*Print[RSSRDesignEqn/.Thread[{u,v,w,x,y,z}→ {u1,v1,w1,x1,y1,z1}]];*)
  (*Print[numsols];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {BC}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((RSSRConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
SecondRSSRSynthesis[thetas_, gammas_, phis_, Const_] :=
 Module[{RSSRsubs, Dinput2useB, RSSRrelD, W, x, y, z, G, u, v, w,
   a, c, b, R, RSSRConstraintEqn, RSSRDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSRsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6, θ7}, {γ1, γ2, γ3, γ4, γ5, γ6, γ7},
       {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7}, {d1, d2, d3, a1, a2, α1, α2}}] →
     Flatten[{thetas, gammas, phis, Const}]];
  Dinput2useB = Dmat /. RSSRsubs;
  RSSRrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  RSSRConstraintEqn =
   Table[Chop[Dot[G - RSSRrelD[[i]].W, G - RSSRrelD[[i]].W] - R^2], {i, 7}];
  RSSRDesignEqn = Table[Chop[Expand[
      RSSRConstraintEqn[[i + 1]] - RSSRConstraintEqn[[1]]]], {i, 6}];
  numsols = NSolve[RSSRDesignEqn == {0, 0, 0, 0, 0, 0},
    {u, v, w, x, y, z}, Method → "Legacy"];

  (*Print[Chop[RSSRrelA]];
  Print[RSSRConstraintEqn];
  Print[RSSRDesignEqn];
  Print[numsols];*)
  (*Print[RSSRDesignEqn/.Thread[{u,v,w,x,y,z}→ {u2,v2,w2,x2,y2,z2}]];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {EF}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[b → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((RSSRConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
DesignEquations[thetas_, gammas_, phis_, Const_] :=
 Module[{RSSRsubs, Dinput2useB, Ainput2useB, RSSRrelD, RSSRrelA, W, G, W2, G2, a, c,
   b, R, RSSRConstraintEqn, RSSRConstraintEqn2, RSSRDesignEqn, RSSRDesignEqn2,
   numsols, realnumsols, numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSRsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6, θ7}, {γ1, γ2, γ3, γ4, γ5, γ6, γ7},
       {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7}, {d1, d2, d3, a1, a2, α1, α2}}] →
     Flatten[{thetas, gammas, phis, Const}]];

  Ainput2useB = Amat /. RSSRsubs;
  RSSRrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  RSSRConstraintEqn =
   Table[Chop[Dot[G - RSSRrelA[[i]].W, G - RSSRrelA[[i]].W] - R^2], {i, 7}];
  RSSRDesignEqn = Table[Chop[Expand[
      RSSRConstraintEqn[[i + 1]] - RSSRConstraintEqn[[1]]]], {i, 6}];

  Dinput2useB = Dmat /. RSSRsubs;
  RSSRrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 7}];
  W2 = {m, n, o, 1};
  G2 = {p, q, r, 1};
  RSSRConstraintEqn2 =
   Table[Chop[Dot[G2 - RSSRrelD[[i]].W2, G2 - RSSRrelD[[i]].W2] - R^2], {i, 7}];
  RSSRDesignEqn2 = Table[Chop[
     Expand[RSSRConstraintEqn2[[i + 1]] - RSSRConstraintEqn2[[1]]]], {i, 6}];
  Print[{RSSRDesignEqn, RSSRDesignEqn2}];
  {NumberForm[RSSRDesignEqn, 2], NumberForm[RSSRDesignEqn2, 2]}
 ]
```

Analysis Modules

The analysis is performed in a similar way as the planar four bar with some additional rotations and translations implemented. The general procedures are essentially the same for the two analysis procedures.

1.) Initial check to skip the analysis if there are no solutions from the synthesis.

2.) Import the linkage information from the synthesis

3.) Calculate the displacement and link length values given the imported coordinates.

4.) Check each branch by subtracting the input phi values from the calculated phi values based off the input theta.

5.) The solutions that do not branch are saved.

6.) solutions that nearly do not branch saved. these are saved for potential future optimization, but are not used currently.

7.) The pertinent information is packaged together for the output.

```
FirstRSSRAnalysis[acbin_, uvwxyzin_, thetas_, gammas_, Const_] :=
 Module[{len, RSSRsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
```

```
 Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
 Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
finalsols = {};
Altfinalsols = {};
Lastsols = {finalsols, Altfinalsols};
If[Length[acbin] == 0, Print["No Solutions found!"];
 Goto[noSolutions]];
len = Length[uvwxyzin];
Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
uvwxyzsub = Table[Thread[{u1, v1, w1, x1, y1, z1} → uvwxyzin[[i]]], {i, len}];
NewABsub =
 Chop[Table[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub[[i]]),
        (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub[[i]], γ → (gammas[[1]]),
            θ → (thetas[[1]])}])}]], {i, len}]];
legsubs = Table[b → acbin[[i]], {i, len}];
If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
      Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
    {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
 Goto[noSolutions]];
(*Print[NewABsub];*)
Branch1 =
 Quiet[ N[Table[Mod[Round[Table[ (γeqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]],
             Constsub, θ → (thetas[[i]])}]) -
         (gammas[[i]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
Branch2 = Quiet[N[Table[Mod[Round[Table[ (γeqn2 /. Flatten[
           {legsubs[[j]], NewABsub[[j]], Constsub, θ → (thetas[[i]])}]) -
         (gammas[[i]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ gammas[[1]]}];*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols =
 Cases[Table[If[Or[Total[Branch1vals[[i]]] == 7, Total[Branch2vals[[i]]] == 7], i],
   {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == 7, 1,
   If[Total[Branch2vals[[ValidSols[[i]]]]] == 7, 2, 0]], {i, Length[ValidSols]}];
```

```
  Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

  finalsols = Table[Thread[{b, u1, v1, w1, x1, y1, z1, braval1} → Flatten[
       {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

  Altfinalsols = If[Length[AltSols] > 0, {thetas, gammas}, {}];
  Lastsols = {finalsols, Altfinalsols};
  Label[noSolutions];
  Lastsols]
```

```
SecondRSSRAnalysis[acbin_, uvwxyzin_, thetas_, gammas_, phis_, Const_] :=
 Module[{len, RSSRsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
   Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
  finalsols = {};
  Altfinalsols = {};
  Lastsols = {finalsols, Altfinalsols};
  If[Length[acbin] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  len = Length[uvwxyzin];
  Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = Table[Thread[{u2, v2, w2, x2, y2, z2} → uvwxyzin[[i]]], {i, len}];
  (*Print[uvwxyzsub];*)
  NewABsub =
   Chop[Table[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub[[i]]),
          (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub[[i]], ϕ → phis[[1]],
              γ → (gammas[[1]]), Θ → (thetas[[1]])}])}]], {i, len}]];
  legsubs = Table[b → acbin[[i]], {i, len}];
  If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
        Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
      {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
   Goto[noSolutions]];

  (*Print[NewABsub];*)
  Branch1 =
   Quiet[ N[Table[Mod[Round[Table[(ϕeqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]],
              Constsub, Θ → (thetas[[i]]), γ -> gammas[[i]]}]) -
           (phis[[i]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
  Branch2 = Quiet[N[Table[Mod[Round[Table[(ϕeqn2 /. Flatten[{legsubs[[j]],
              NewABsub[[j]], Constsub, Θ → (thetas[[i]]), γ -> gammas[[i]]}]) -
           (phis[[i]]), {i, 7}], π * 10^-4], 2 * Pi], {j, len}]]];
  (*Print[NewABsub];*)
  (*Print[legsubs];*)
  (*Print[legsubs,uvwxyzsub];*)
  (*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
    ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ gammas[[1]]}]*)
  Branch1adj = N[Branch1];
  Branch2adj = N[Branch2];
  Branch1vals =
```

```
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols =
 Cases[Table[If[Or[Total[Branch1vals[[i]]] == 7, Total[Branch2vals[[i]]] == 7], i],
   {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == 7, 1,
   If[Total[Branch2vals[[ValidSols[[i]]]]] == 7, 2, 0]], {i, Length[ValidSols]}];

Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

finalsols = Table[Thread[{b, u2, v2, w2, x2, y2, z2, braval2} → Flatten[
     {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

Altfinalsols = If[Length[AltSols] > 0, {thetas, gammas}, {}];
Lastsols = {finalsols, Altfinalsols};
Label[noSolutions];
Lastsols
```

Continuity Test

These check that the lengths of the links do not change through out its movements. The interval can be selected by setting the limit number. *** NOTE: The values all go from 0 to a ThetaMax... a Theta Min Value has not been implemented. ******

1.) Import the linkage information that passed through the analysis

2.) calculate the coordinates of the spherical joints in the correct reference. The moving joints should be in the frame of the revolute joint in which they rotate about.

*** In the second test tables of the γ and ϕ angles were calculated *****

3.) Calculate the initial link lengths

4.) Calculate the link lengths at the specified intervals

5.) Subtract the original link lengths from the subsequent link lengths

```
ContTest1[inputs_, Const_, thetas_, gammas_, ThetaMax_] :=
 Module[{len, Constsub, legsubs, uvwxyzsub, RSSRsubs, NewABsub, limit, OAlinklen,
   ABlinklen, OAlen, ABlen, BClen, TestedSols, TableOfO, TableOfD, TableOfE, TableOfS},
  If[Length[inputs] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = inputs[[2 ;; 7]];
  legsubs = {b → (b /. inputs)};
  RSSRsubs = Constsub;
  limit = 200;

  NewABsub = Chop[Thread[
     {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
           {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];

  OAlinklen = OAlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
  ABlinklen =
   ABlink /. Flatten[{Constsub, NewABsub, (γ → gammas[[1]]), (θ → thetas[[1]])}] ;

  OAlen = N[Table[Norm[(((Acoord - Ocoord) /. Flatten[{legsubs, NewABsub, RSSRsubs}]) /.
        {θ → (ThetaMax * (n / limit))})], {n, limit}]];
  ABlen = N[Table[Norm[((((Bcoord - Acoord) /.
            {γ → ((If[(braval1 /. inputs) == 1, γeqn1, γeqn2]) /. Flatten[
                 {legsubs, RSSRsubs, NewABsub, θ → (ThetaMax * (n / limit))}])}) /.
          Flatten[{legsubs, RSSRsubs, NewABsub}]) /. θ →
         (ThetaMax * (n / limit)))], {n, limit}]];
  BClen = N[Table[Norm[(((((Ccoord - Bcoord) /.
             {γ → ((If[(braval1 /. inputs) == 1, γeqn1, γeqn2]) /. Flatten[
                  {legsubs, RSSRsubs, NewABsub, θ → (ThetaMax * (n / limit))}])}) /.
           Flatten[{legsubs, RSSRsubs, NewABsub}])) /. θ → (ThetaMax *
           (n / limit)))], {n, limit}]];
  (*Print[({OAlinklen,ABlinklen,b}/.legsubs)];
  Print[{OAlen,ABlen,BClen}];*)
  TestedSols = {Chop[{OAlen, ABlen, BClen} - ({OAlinklen, ABlinklen, b} /. legsubs)]};
  Label[noSolutions];
  TestedSols]
```

```
ContTest2[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    ThetaTable, GammaTable, PhiTable, ADlinklen, DElinklen, limit, ADlen,
    DElen, EFlen, TestedSols, TableOfO, TableOfA, TableOfB, TableOfC},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 = Chop[Thread[
      {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
           {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];


   GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
        {legsubs1, Constsub, NewABsub1, θ → (ThetaMax * (n / limit))}]), {n, limit}];
   PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /.
       Flatten[{legsubs2, Constsub, NewABsub2,
         θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];

   ADlinklen = ADlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[
      {Constsub, NewABsub2, (γ → gammas[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];

   ADlen = N[Table[Norm[((Dcoord - Acoord) /. Flatten[{Constsub, (γ → GammaTable[[n]]),
            (ϕ → PhiTable[[n]])}]) /. {θ → (ThetaMax * (n / limit))}], {n, limit}]];
   (*Print[Alen];*)

   DElen = N[Table[
      Norm[(Ecoord - Dcoord) /. Flatten[{Constsub, NewABsub2, (γ → GammaTable[[n]]),
          (ϕ → PhiTable[[n]]), (θ → (ThetaMax * (n / limit)))}]], {n, limit}]];

   EFlen = N[Table[
      Norm[(Fcoord - Ecoord) /. Flatten[{Constsub, NewABsub2, (γ → GammaTable[[n]]),
          (ϕ → PhiTable[[n]]), (θ → (ThetaMax * (n / limit)))}]], {n, limit}]];



   (*Print[{Alen,Blen,Clen}];*)
   TestedSols = {(*inputs2[[1;;3]],*)
     Chop[{ADlen, DElen, EFlen} - ({ADlinklen, DElinklen, b /. legsubs2})]};
   (*Print[TestedSols];*)
   TestedSols];
```

Plotting Solutions

This is the module used for plotting the solutions.

```
PlottingSolutions[inputs1_, inputs2_, Const_,
  thetas_, gammas_, phis_, gammaeqn_, phieqn_, ThetaMax_] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, RSSRsubs, NewABsub1,
   NewABsub2, GammaTable, PhiTable, limit, plotγeqn1calc, plotγeqn2calc,
   plotϕeqn1calc, plotϕeqn2calc, plotγeqn1, plotγeqn2, plotϕeqn1, plotϕeqn2,
   plotΘ, plotϕ, plotγ, γPoints2Plot, ϕPoints2Plot, γplot, ϕplot},
  Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  RSSRsubs = Constsub;
  limit = 200;

  NewABsub1 = Chop[Thread[
     {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
           {Constsub, uvwxyzsub, (γ → gammas[[1]]), (Θ → thetas[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
       {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
            (ϕ → phis[[1]]), (γ → (gammas[[1]])), (Θ → thetas[[1]])}])}]]];

  plotγeqn1calc = Table[
    (γeqn1 /. Flatten[{legsubs1, Constsub, NewABsub1, Θ → (ThetaMax * (n / limit))}]),
    {n, limit}];
  plotγeqn2calc = Table[(γeqn2 /. Flatten[{legsubs1, Constsub,
       NewABsub1, Θ → (ThetaMax * (n / limit))}]), {n, limit}];

  GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
      {legsubs1, RSSRsubs, NewABsub1, Θ → (ThetaMax * (n / limit))}]), {n, limit}];
  plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
       Θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
  plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
       Θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
  plotΘ = Table[ThetaMax * (n / limit), {n, limit}];

  plotγ = N[(gammaeqn /. x → #) & /@ plotΘ];
  plotϕ = N[(phieqn /. x → #) & /@ plotΘ];

  γPoints2Plot = Table[{thetas[[i]], gammas[[i]]}, {i, Length[thetas]}];
  ϕPoints2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];

  plotγeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotγeqn1calc];
  plotγeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotγeqn2calc];
  plotϕeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn1calc];
  plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];
```

```
  γplot = ListPlot[{MapThread[{#1, #2} &, {plotθ, plotγ}], MapThread[{#1, #2} &,
      {plotθ, plotγeqn1}], MapThread[{#1, #2} &, {plotθ, plotγeqn2}], γPoints2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "γ Plot Comparison"];
  ϕplot = ListPlot[{MapThread[{#1, #2} &, {plotθ, plotϕ}], MapThread[{#1, #2} &,
      {plotθ, plotϕeqn1}], MapThread[{#1, #2} &, {plotθ, plotϕeqn2}], ϕPoints2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "ϕ Plot Comparison"];
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm, γplot, ϕplot}
 ]
```

```
FirstPosition3DPlot[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_] :=
  Module[{Constsub, uvwxyzsub, InitialPos, ptO,
    ptA, ptB, ptC, ptD, ptE, ptF, TubeDia, SphereDia},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   InitialPos = Thread[{θ, γ, ϕ} → {thetas[[1]], gammas[[1]], phis[[1]]}];
   ptO = Ocoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptA = Acoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptB = Bcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptC = Ccoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptD = Dcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptE = Ecoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptF = Fcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];

   Print[uvwxyzsub];

   TubeDia = 0.5;
   SphereDia = 1;
   Graphics3D[{Gray, Tube[{ptO, ptC, ptF}, TubeDia], Green,
     Tube[{ptO, ptA}, TubeDia], Blue, Tube[{ptB, ptA, ptD}, TubeDia],
     Yellow, Tube[{ptB, ptC}, TubeDia], Sphere[{ptB, ptC}, SphereDia],
     Red, Tube[{ptD, ptE}, TubeDia], Purple, Tube[{ptE, ptF}, TubeDia],
     Sphere[{ptE, ptF}, SphereDia], Orange, Tube[{{0, 0, 0}, ptO}, TubeDia/2]}]
  ];
```

```
Plot4Paper[inputs1_, inputs2_, Const_, thetas_,
  gammas_, phis_, gammaeqn_, phieqn_, ThetaMax_] := Module[
  {Constsub, legsubs1, legsubs2, uvwxyzsub, RSSRsubs, NewABsub1, NewABsub2, GammaTable,
   PhiTable, limit, plotγeqn1calc, plotγeqn2calc, plotϕeqn1calc, plotϕeqn2calc,
   plotγeqn1, plotγeqn2, plotϕeqn1, plotϕeqn2, plotθ, plotϕ, plotγ, γPoints2Plot,
   ϕPoints2Plot, γplot, ϕplot, γϕplot, γ4export, ϕ4export, SwingPitchPlot},
  Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
```

```
RSSRsubs = Constsub;
limit = 200;

NewABsub1 = Chop[Thread[
   {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
         {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

plotγeqn1calc = Table[
  (γeqn1 /. Flatten[{legsubs1, Constsub, NewABsub1, θ → (ThetaMax * (n / limit))}]),
  {n, limit}];
plotγeqn2calc = Table[(γeqn2 /. Flatten[{legsubs1, Constsub,
      NewABsub1, θ → (ThetaMax * (n / limit))}]), {n, limit}];

GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
     {legsubs1, RSSRsubs, NewABsub1, θ → (ThetaMax * (n / limit))}]), {n, limit}];
PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
plotθ = Table[ThetaMax * (n / limit), {n, limit}];

plotγ = N[(gammaeqn /. x → #) & /@ plotθ];
plotϕ = N[(phieqn /. x → #) & /@ plotθ];

γPoints2Plot = Table[{thetas[[i]], gammas[[i]]}, {i, Length[thetas]}];
ϕPoints2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];

plotγeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotγeqn1calc];
plotγeqn2 =
 Map[If[# < -Pi - .5, # + 2 * Pi, If[# > Pi - .5, # - 2 * Pi, #]] &, plotγeqn2calc];
plotϕeqn1 = Map[If[# < -Pi - .5, # + 2 * Pi, If[# > Pi - .5, # - 2 * Pi, #]] &,
  plotϕeqn1calc];
plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];

GammaTable = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, GammaTable];
PhiTable = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, PhiTable];

γplot = ListPlot[
  {MapThread[{#1, #2} &, {plotθ, plotγ}], MapThread[{#1, #2} &, {plotθ, plotγeqn1}],
   MapThread[{#1, #2} &, {plotθ, plotγeqn2}], γPoints2Plot}, PlotLegends →
   {"Desired Swing Angle", "Branch 1", "Branch 2", "Precision Points"},
  Joined → {True, True, True, False}, PlotStyle → {Thick, Dashing[Medium],
    Dashing[Tiny], PointSize[Large]}, PlotLabel → "γ Plot Comparison",
```

```
    LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
  ϕplot = ListPlot[{MapThread[{#1, #2} &, {plotθ, plotϕ}], MapThread[{#1, #2} &,
      {plotθ, plotϕeqn1}], MapThread[{#1, #2} &, {plotθ, plotϕeqn2}], ϕPoints2Plot},
    PlotLegends → {"Desired Pitch Angle", "Branch 1", "Branch 2",
      "Precision Points"}, Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "ϕ Plot Comparison",
    LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
  (*{{inputs1[[1]],inputs2[[1]]}//MatrixForm,γplot,ϕplot}*)

  γ4export = Show[γplot, ImageSize → Large,
    PlotLabel → "", FrameLabel → {Style["Crank Angle (Radians)", 18],
      Style["Swing Angle (Radians)", 18]}, Frame → {True, True, False, False},
    LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
    TicksStyle → Directive[FontSize → 20]];
  ϕ4export = Show[ϕplot, ImageSize → Large, PlotLabel → "",
    FrameLabel → {Style["Crank Angle (Radians)", 18],
      Style["Pitch Angle (Radians)", 18]}, Frame → {True, True, False, False},
    LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
    TicksStyle → Directive[FontSize → 20]];

  SwingPitchPlot = ListPlot[{MapThread[{#1, #2} &, {plotθ, plotγ}],
     MapThread[{#1, #2} &, {plotθ, plotϕ}], γPoints2Plot, ϕPoints2Plot}, PlotLegends →
     {"Desired Swing Angle", "Desired Pitch Angle", "Swing Precision Points",
      "Pitch Precision Points"}, Joined → {True, True, False, False},
    PlotStyle → {Thick, Dashing[Medium], PointSize[.015], PointSize[.02]},
    ImageSize → Large, FrameLabel → {Style["Crank Angle (Radians)", 18],
      Style["Pitch Angle (Radians)", 18]}, Frame → {True, True, False, False},
    LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

  γϕplot = ListPlot[
    {MapThread[{#1, #2} &, {plotθ, plotγ}], MapThread[{#1, #2} &, {plotθ, plotϕ}],
     γPoints2Plot, ϕPoints2Plot, MapThread[{#1, #2} &, {plotθ, PhiTable}],
     MapThread[{#1, #2} &, {plotθ, GammaTable}]}, PlotLegends →
     {"Desired Swing Angle", "Desired Pitch Angle", "Swing Precision Points",
      "Pitch Precision Points", "Pitch Angle", "Swing Angle"},
    Joined → {True, True, False, False, True, True}, PlotStyle → {{Thick}, {Medium},
      {PointSize[.015]}, {PointSize[.02]}, {Dashing[Medium]}, {Dashing[Tiny]}},
    ImageSize → Large, FrameLabel → {Style["Crank Angle (Radians)", 18],
      Style["Pitch Angle (Radians)", 18]}, Frame → {True, True, False, False},
    LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

  {{inputs1[[1]], inputs2[[1]]} // MatrixForm,
   γ4export, ϕ4export, SwingPitchPlot, γϕplot}
]
```

Ranking Solutions

The LinkRatio module calculates all of the link lengths and then calculates the ratio between the longest and shortest link.

```
LinkRatio[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 = Chop[Thread[
      {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
           {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   ABlinklen =
    ABlink /. Flatten[{Constsub, NewABsub1, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (γ → gammas[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2}];
   BDlinklen =
    BDlink /. Flatten[{Constsub, NewABsub1, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   Max[LinkLengths] / Min[LinkLengths]
  ];
```

The CurveComparison module find the root mean square difference between the γ and ϕ equations given in the input and the

```
CurveComparison[inputs1_, inputs2_, Const_,
   thetas_, gammas_, phis_, gammaeqn_, phieqn_, ThetaMax_] :=
  Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, RSSRsubs, NewABsub1, NewABsub2,
    GammaTable, PhiTable, limit, plotγeqn1calc, plotγeqn2calc, plotϕeqn1calc,
    plotϕeqn2calc, plotγeqn1, plotγeqn2, plotϕeqn1, plotϕeqn2, plotθ, plotϕ,
    plotγ, γPoints2Plot, ϕPoints2Plot, γplot, ϕplot, GammaError, PhiError},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   RSSRsubs = Constsub;
   limit = 200;
```

```
NewABsub1 = Chop[Thread[
   {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
         {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

plotγeqn1calc = Table[
  (γeqn1 /. Flatten[{legsubs1, Constsub, NewABsub1, θ → (ThetaMax * (n / limit))}]),
  {n, limit}];
plotγeqn2calc = Table[(γeqn2 /. Flatten[{legsubs1, Constsub,
      NewABsub1, θ → (ThetaMax * (n / limit))}]), {n, limit}];

GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
     {legsubs1, RSSRsubs, NewABsub1, θ → (ThetaMax * (n / limit))}]), {n, limit}];
PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];

plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];
plotθ = Table[ThetaMax * (n / limit), {n, limit}];

plotγ = N[(gammaeqn /. x → #) & /@ plotθ];
plotϕ = N[(phieqn /. x → #) & /@ plotθ];

γPoints2Plot = Table[{thetas[[i]], gammas[[i]]}, {i, Length[thetas]}];
ϕPoints2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];

plotγeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotγeqn1calc];
plotγeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotγeqn2calc];
plotϕeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn1calc];
plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];

GammaError = Sqrt[Total[MapThread[(#1 - #2)^2 &,
     {If[(braval1 /. inputs1) == 1, plotγeqn1, plotγeqn2], plotγ}]] / limit];
PhiError = Sqrt[Total[MapThread[(#1 - #2)^2 &,
     {If[(braval2 /. inputs2) == 1, plotϕeqn1, plotϕeqn2], plotϕ}]] / limit];
(*Print[{GammaError,PhiError}];*)
GammaError + PhiError
];
```

```
AllLinkLengths[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 = Chop[Thread[
      {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
           {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   ABlinklen =
    ABlink /. Flatten[{Constsub, NewABsub1, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (γ → gammas[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2}];
   BDlinklen =
    BDlink /. Flatten[{Constsub, NewABsub1, (γ → gammas[[1]]), (θ → thetas[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   LinkLengths
  ];
```

Finding Solutions

```
SolutionPath[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    ThetaTable, GammaTable, PhiTable, OutputTable, limit, GammaOut, PhiOut},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 = Chop[Thread[
      {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
            {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

   ThetaTable = Table[ThetaMax * (n / limit), {n, limit}];
   GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
        {legsubs1, Constsub, NewABsub1, θ → (ThetaMax * (n / limit))})]), {n, limit}];
   PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /.
       Flatten[{legsubs2, Constsub, NewABsub2,
         θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];


   GammaOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, GammaTable];
   PhiOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, PhiTable];

   OutputTable = {ThetaTable, GammaOut, PhiOut};
   OutputTable];
```

Inputs

Input Functions

These are the functions that are needed to drive the wing swing and pitch

```
WingSwingEqn = -60 * Degree * Cos[x] + 90 * Degree
```

```
90 ° - 60 ° Cos[x]
```

```
WingPitchEqn = π / 2 - 60 * Degree * Sin[x]
```

$$\frac{\pi}{2} - 60 \circ \text{Sin}[x]$$

```
Upperb = 2 * Pi; Lowerb = 0;
```

```
precisionptsθ8POINTS = Chop[N[ChebyshevSpacing[6, Lowerb, Upperb]]]
```

```
{0, 0.107047, 0.920151, 2.32849, 3.9547, 5.36303, 6.17614, 6.28319}
```

```
precisionptsθ8POINTS
```

```
{0, 0.107047, 0.920151, 2.32849, 3.9547, 5.36303, 6.17614, 6.28319}
```

```
(*precisionptsθ=precisionptsθ8POINTS[[1;;7]]*)
```

```
precisionptsθ = Table[2 * Pi * i / 7., {i, 0, 6}]
```

```
{0., 0.897598, 1.7952, 2.69279, 3.59039, 4.48799, 5.38559}
```

```
precisionptsγ = (WingSwingEqn /. x → #) & /@ precisionptsθ
```

```
{0.523599, 0.917879, 1.80382, 2.51429, 2.51429, 1.80382, 0.917879}
```

```
precisionptsϕ = (WingPitchEqn /. x → #) & /@ precisionptsθ
```

```
{1.5708, 0.752064, 0.549854, 1.11643, 2.02516, 2.59174, 2.38953}
```

```
Firstθ = precisionptsθ[[1]]
```

```
0.
```

```
Firstγ = precisionptsγ[[1]]
```

```
0.523599
```

```
Firstϕ = precisionptsϕ[[1]]
```

```
1.5708
```

Tolerances

These are the tolerances that are used for the randomization process

```
(*ThetaTols =N[Array[15&,7]Degree];*)
ThetaTols = {0, 5, 5, 5, 5, 5, 5} * Degree
```

```
{0, 5 °, 5 °, 5 °, 5 °, 5 °, 5 °}
```

```
(*PhiTols=N[Array[15&,7]Degree]*)
PhiTols = {0, 5, 5, 5, 5, 5, 5} * Degree
```

```
{0, 5 °, 5 °, 5 °, 5 °, 5 °, 5 °}
```

```
(*PhiTols=N[Array[15&,7]Degree]*)
GammaTols = {0, 5, 5, 5, 5, 5, 5} * Degree
```

```
{0, 5 °, 5 °, 5 °, 5 °, 5 °, 5 °}
```

Position of three revolute joints

Define the position at the start position. Give coordinates for the center of the joint and the vector of the axis of rotation. The three revolute joints should be able to be placed anywhere in their initial position. The code will determine the constant input values from there.

Revolute Joint 1 is used as the starting point

Original Test

```
Rev1pos = {3, -5, 0};
```

```
Rev1vec = {0, 1, 1};
```

```
Rev2pos = {0, 0, 0};
```

```
Rev2vec = {0, 0, 1};
```

```
Rev3pos = {1, 2, 0};
```

```
Rev3vec = {0, 1, 0};
```

Calculate the DH table for the three revolute joints

The following calculates the common normals between the lines it outputs:
d: the distance the joint is from the common normal along each of their respective axes
α: the angle between the two lines about the common normal
a: the length of the common normal

ptr, ptc: the coordinates on each line where the common normal is intersected

```
{d1val, a1val, α1val, d2pt1val, ptc1val, ptr1val} =
 CommonNorm[Rev1pos, Rev1vec, Rev2pos, Rev2vec]
```

$$\left\{5\sqrt{2},\ 3,\ \frac{\pi}{4},\ 5,\ \{3,\ 0,\ 5\},\ \{0,\ 0,\ 5\}\right\}$$

```
{d2pt2val, a2val, α2val, d3val, ptc2val, ptr2val} =
 CommonNorm[Rev2pos, Rev2vec, Rev3pos, Rev3vec]
```

$$\left\{0,\ 1,\ \frac{\pi}{2},\ 2,\ \{0,\ 0,\ 0\},\ \{1,\ 0,\ 0\}\right\}$$

Find the initial γ and ϕ values

```
{d1g2i, a1g2i, α1g2i, d2ptg2i, ptcg2i, ptrg2i} =
 CommonNorm[{0, 0, 0}, {0, 0, 1}, Rev1pos, Rev1vec]
```

$$\left\{5,\ 3,\ \frac{\pi}{4},\ 5\sqrt{2},\ \{0,\ 0,\ 5\},\ \{3,\ 0,\ 5\}\right\}$$

```
G2Iavec = ptrg2i - ptcg2i
```

$$\{3,\ 0,\ 0\}$$

```
G2IangleZ = ArcTan[G2Iavec[[1]], G2Iavec[[2]]] - ArcTan[1, 0]
```

$$0$$

```
a1g2isign = Sign[Dot[Zmat[{0, d1g2i}].{1, 0, 0, 1}.rem, ptrg2i - ptcg2i]]
```

$$1$$

```
a1g2i = a1g2i * a1g2isign
```

$$3$$

```
d2g2isign =
 Sign[Dot[Zmat[{0, d1g2i}].Xmat[{α1g2i, a1g2i}].{0, 0, 1, 1}.rem, Rev1pos - ptrg2i]]
```

$$-1$$

```
d2ptg2i = d2ptg2i * d2g2isign
```

$$-5\sqrt{2}$$

```
XaxisInitial =
 Zmat[{0, d1g2i}].Xmat[{α1g2i, a1g2i}].Zmat[{G2IangleZ, d2ptg2i}].{1, 0, 0, 0}.rem
```

```
{1, 0, 0}
```

```
γinitialvector = ptc1val - ptr1val
```

```
{3, 0, 0}
```

```
γinitial = ArcTan[γinitialvector[[1]], γinitialvector[[2]]] -
  ArcTan[XaxisInitial[[1]], XaxisInitial[[2]]]
```

```
0
```

a1 and its sign with respect to DH transformations

Sign is calculated by finding the positive direction of the X and transforming it to the same coordinate frame as the a link and comparing the direction to that of the ptc to ptr vector

```
a1valsign = Sign[Dot[Zmat[{0, d1val}].{1, 0, 0, 1}.rem, ptr1val - ptc1val]]
```

```
-1
```

```
a1vals = a1valsign * a1val
```

```
-3
```

d2 and its sign with respect to DH transformations
the d2 value is the distance along the axis of rotation of the second revolute joint between the intersection of the two common normals.

```
d2valsign =
 Sign[Dot[Zmat[{0, d1val}].Xmat[{α1val, a1vals}].{0, 0, 1, 1}.rem, ptc2val - ptr1val]]
```

```
-1
```

```
d2vals = d2valsign * Norm[ptc2val - ptr1val]
```

```
-5
```

a2 and its sign with respect to DH transformations

```
a2valsign = Sign[Dot[Zmat[{0, d1val}].Xmat[{α1val, a1vals}].
    Zmat[{γinitial, d2vals}].{1, 0, 0, 1}.rem, ptr2val - ptc2val]]
```

```
-1
```

```
a2vals = a2valsign * a2val
```

```
-1
```

d3 and its sign with respect to DH transformations

```
d3valsign = Sign[Dot[Zmat[{0, d1val}].Xmat[{α1val, a1vals}].Zmat[{γinitial, d2vals}].
    Xmat[{α2val, a2vals}].{0, 0, 1, 1}.rem, Rev3pos - ptr2val]]
```

```
1
```

```
d3vals = d3valsign * d3val
```

```
2
```

```
PrimePi[d3vals]
```

```
1
```

Const is the array that holds all of the constant values for the calculations.

```
Const = N[{d1val, d2vals, d3vals, a1vals, a2vals, α1val, α2val}]
```

```
{7.07107, -5., 2., -3., -1., 0.785398, 1.5708}
```

```
DHTable = {{d1, Θ, a1, α1}, {d2, γ, a2, α2}, {d3, ϕ, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} d1 & \Theta & a1 & \alpha 1 \\ d2 & \gamma & a2 & \alpha 2 \\ d3 & \phi & 0 & 0 \end{pmatrix}$$

```
DHTablePaper = {{Θ1, d1, α1, a1}, {Θ2, d2, α2, a2}, {Θ3, d3, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} \Theta 1 & d1 & \alpha 1 & a1 \\ \Theta 2 & d2 & \alpha 2 & a2 \\ \Theta 3 & d3 & 0 & 0 \end{pmatrix}$$

```
DHTable /. Thread[{d1, d2, d3, a1, a2, α1, α2} → Const]
```

$$\begin{pmatrix} 7.07107 & \Theta & -3. & 0.785398 \\ -5. & \gamma & -1. & 1.5708 \\ 2. & \phi & 0 & 0 \end{pmatrix}$$

```
DHTablePaper /. Thread[{d1, d2, d3, a1, a2, α1, α2} → Const]
```

$$\begin{pmatrix} \Theta 1 & 7.07107 & 0.785398 & -3. \\ \Theta 2 & -5. & 1.5708 & -1. \\ \Theta 3 & 2. & 0 & 0 \end{pmatrix}$$

```
Transpose[{{θ1, 7.07107, 0.785398, -3.}, {θ2, -5., 1.5708, -1.}, {θ3, 2., 0, 0}}]
```

```
{{θ1, θ2, θ3}, {7.07107, -5., 2.}, {0.785398, 1.5708, 0}, {-3., -1., 0}}
```

```
{{θ1, 5 * Sqrt[2], Pi / 4, -3}, {θ2, -5, Pi / 2, -1}, {θ3, 2, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} \theta 1 & 5\sqrt{2} & \frac{\pi}{4} & -3 \\ \theta 2 & -5 & \frac{\pi}{2} & -1 \\ \theta 3 & 2 & 0 & 0 \end{pmatrix}$$

Joint Coordinates and Rotation Axes in the New Frame

```
ConstSubstitutions = Thread[Flatten[{d1, d2, d3, a1, a2, α1, α2, θ, γ, ϕ}] →
   Flatten[{Const, precisionptsθ[[1]], precisionptsγ[[1]], precisionptsϕ[[1]]}]]
```

```
{d1 → 7.07107, d2 → -5., d3 → 2., a1 → -3., a2 → -1.,
 α1 → 0.785398, α2 → 1.5708, θ → 0., γ → 0.523599, ϕ → 1.5708}
```

```
OcoordAxis = ((Zmat[{θ, d1}].{0, 0, 1, 0}) /. ConstSubstitutions).rem
```

```
{0, 0, 1}
```

```
OcoordInitial = ((Ocoord) /. ConstSubstitutions).rem
```

```
{0., 0., 7.07107}
```

```
AcoordAxis =
  ((Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{0, 0, 1, 0}) /. ConstSubstitutions).rem
```

```
{0., -0.707107, 0.707107}
```

```
AcoordInitial = (Acoord /. ConstSubstitutions).rem
```

```
{-3., 3.53553, 3.53553}
```

```
AcoordInitial + AcoordAxis
```

```
{-3., 2.82843, 4.24264}
```

```
AcoordXAxis =
  ((Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{1, 0, 0, 0}) /. ConstSubstitutions).rem
```

```
{0.866025, 0.353553, 0.353553}
```

```
AcoordInitial + AcoordXAxis * .5
```

```
{-2.56699, 3.71231, 3.71231}
```

```
DcoordAxis =
 ((Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].
      {0, 0, 1, 0}) /. ConstSubstitutions).rem
```

```
{0.5, -0.612372, -0.612372}
```

```
DcoordInitial = (Dcoord /. ConstSubstitutions).rem
```

```
{-2.86603, 1.95724, 1.95724}
```

```
DcoordAxis + DcoordInitial
```

```
{-2.36603, 1.34486, 1.34486}
```

```
DcoordXAxis =
 ((Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].
      {1, 0, 0, 0}) /. ConstSubstitutions).rem
```

$$\{2.24126 \times 10^{-17}, -0.707107, 0.707107\}$$

```
DcoordXAxis * .5 + DcoordInitial
```

```
{-2.86603, 1.60368, 2.31079}
```

Solvers

Main Solver

This is the solver. The solver first randomizes the θ and γ values. Once a set of θ and γ values are found that pass analysis they are used repeatedly for sets of randomized ϕ values.

This solver utilizes multithreading. It will still work if multithreading is not initialized, but it will run single threaded.

```
θmax = 2 * Pi;
Iterations = 100;
k1 = 1;
SecondIterations = 100;
SolutionList = Array[0 &, Iterations];
```

```
Randvals = {precisionptsθ, precisionptsɣ, precisionptsϕ};
SolutionList = ParallelTable[
   Print[i];
   If[i > 1, Randvals = RandomizeTps[precisionptsθ,
       precisionptsɣ, precisionptsϕ, ThetaTols, GammaTols, PhiTols];,
    Randvals = {precisionptsθ, precisionptsɣ, precisionptsϕ};];
   (*First Four Bar Code *)
   test1 = Chop[FirstRSSRSynthesis[Randvals[[1]], Randvals[[2]], Const]];
   testanal1 =
    FirstRSSRAnalysis[test1[[1]], test1[[2]], Randvals[[1]], Randvals[[2]], Const];
   test1totals = If[Length[testanal1[[1]]] == 0, "No Solution",
     Table[Total[Flatten[ContTest1[testanal1[[1, i]], Const, Randvals[[1]],
         Randvals[[2]], θmax]]], {i, Length[testanal1[[1]]]}]];
   Print[test1totals];
   test1sols = Table[If[test1totals[[i]] == 0, {testanal1[[1, i]],
       Randvals[[1]], Randvals[[2]]}, 0], {i, Length[test1totals]}];
   test1sols = DeleteCases[test1sols, 0];
   Print[test1sols];
   (*Second Four Bar Code *)
   SecondSolutionList = Array[0 &, SecondIterations];
   (*Creates array of zeros to store solutions from second four bar*)
   If[Length[test1sols] == 0, 0,
    Theta1sols = test1sols[[1, 2]]; (*Saves the randomized theta values*)
    Gamma1sols = test1sols[[1, 3]]; (* Saves the randomized gamma values*)
    k2 = 1;
    RandPhi = precisionptsϕ;
    For[n2 = 0, n2 < SecondIterations, n2++,
     Print["Inside ", n2];
     test2 = Chop[SecondRSSRSynthesis[Theta1sols, Gamma1sols, RandPhi, Const]];
     testanal2 = SecondRSSRAnalysis[
       test2[[1]], test2[[2]], Theta1sols, Gamma1sols, RandPhi, Const];
     test2totals = If[Length[testanal2[[1]]] == 0, "No Solution",
       Table[Table[Total[Flatten[ContTest2[test1sols[[k, 1]],
            testanal2[[1, i]], Const, Theta1sols, Gamma1sols, RandPhi, θmax]]],
         {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}]];
     test2sols = Table[Table[If[test2totals[[k, i]] == 0,
         {test1sols[[k, 1]], testanal2[[1, i]], Theta1sols, Gamma1sols, RandPhi}, 0],
        {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}];
     test2sols = DeleteCases[Flatten[test2sols, 1], 0];
     If[test2sols == {}, , SecondSolutionList[[k2]] = test2sols; k2++;];
     RandPhi = RandomizeTps1[precisionptsϕ, PhiTols];
    ](*End of second four bar FOR loop*)
   ]; (* End of Second four bar IF statement*)
   SecondSolutionList = DeleteCases[SecondSolutionList, 0];
   SecondSolutionList
   , {i, Iterations}];(*End of First Four bar FOR loop*)
Print["Stop 1"];
SolutionList = DeleteCases[SolutionList, (0 | {})];
Print["Stop 2"];
SolutionList = Flatten[SolutionList, 2];
Print["Stop 3"];
```

```
SolutionList
```

Plotted Solutions

```
ParallelTable[PlottingSolutions[SolutionList[[i, 1]], SolutionList[[i, 2]],
  Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]],
  WingSwingEqn, WingPitchEqn, 2 * Pi], {i, Length[SolutionList]}]
```

Ranking Solutions

Calculates the link length ratios of each solution.

```
Length[SolutionList]
```

```
SolsRankLinkLen =
  Sort[ParallelTable[{LinkRatio[SolutionList[[i, 1]], SolutionList[[i, 2]],
      Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]]],
    SolutionList[[i]]}, {i, Length[SolutionList]}]];
```

```
MaxRatio = 10;
```

```
SolsRankLinkLen2use =
  DeleteCases[Table[If[SolsRankLinkLen[[i, 1]] < MaxRatio, SolsRankLinkLen[[i, 2]], 0],
    {i, Length[SolsRankLinkLen]}], 0];
```

```
SolsRankLinkLen
```

```
Length[SolsRankLinkLen2use]
```

Compares the RMS errors.

```
SortedSols = Sort[ParallelTable[
    {CurveComparison[SolsRankLinkLen2use[[i, 1]], SolsRankLinkLen2use[[i, 2]],
      Const, SolsRankLinkLen2use[[i, 3]], SolsRankLinkLen2use[[i, 4]],
      SolsRankLinkLen2use[[i, 5]], WingSwingEqn, WingPitchEqn, 2 * Pi],
     SolsRankLinkLen2use[[i]]}, {i, Length[SolsRankLinkLen2use]}]];
```

```
ParallelTable[PlottingSolutions[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
  Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]],
  WingSwingEqn, WingPitchEqn, 2 * Pi], {i, Length[SortedSols]}]
```

```
ParallelTable[FirstPosition3DPlot[SortedSols[[i, 2, 1]],
  SortedSols[[i, 2, 2]], Const, SortedSols[[i, 2, 3]],
  SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]]], {i, Length[SortedSols]}]
```

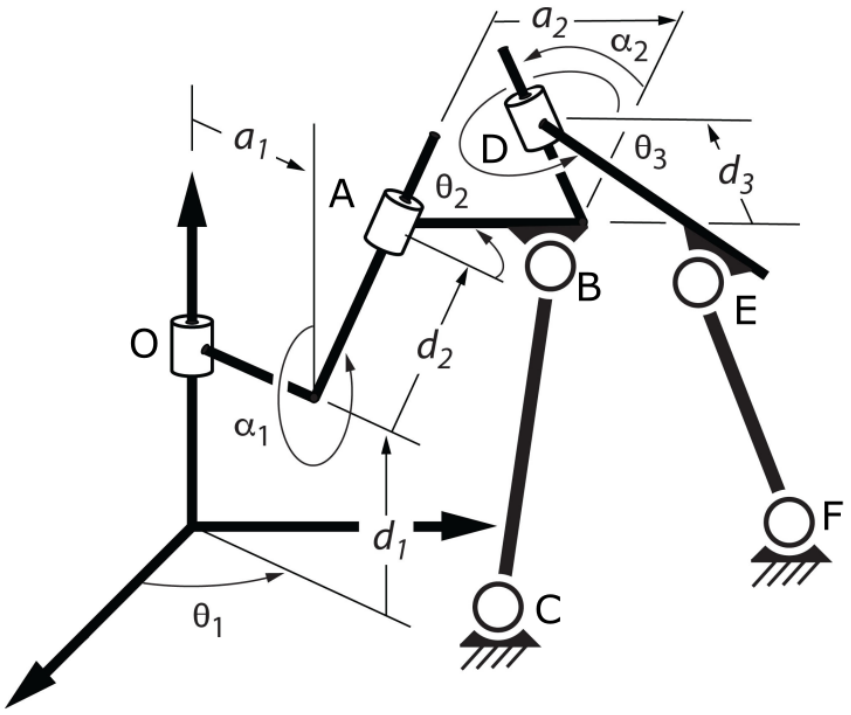
```
Length[SortedSols]
```

```
ParallelTable[{AllLinkLengths[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
    Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]]],
  Append[SortedSols[[i, 2]], Const]}, {i, Length[SortedSols]}]
```

Appendix B

RPR-2SS Valve Mechanism Mathematica Code

Spatial Six Bar RPR-2SS



Multithreading SetUp

```
ParallelEvaluate[$KernelCount]
```

```
{16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16}
```

```
Kernels[]
```

```
{KernelObject[1, local], KernelObject[2, local],
 KernelObject[3, local], KernelObject[4, local], KernelObject[5, local],
 KernelObject[6, local], KernelObject[7, local], KernelObject[8, local],
 KernelObject[9, local], KernelObject[10, local], KernelObject[11, local],
 KernelObject[12, local], KernelObject[13, local],
 KernelObject[14, local], KernelObject[15, local], KernelObject[16, local]}
```

```
$ProcessorCount
```

```
16
```

Basic Functions and Constants

These equations are all calculated outside the module to reduce the number of repeated calculations

```
Zmat[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], 0, 0},
   {Sin[x[[1]]], Cos[x[[1]]], 0, 0}, {0, 0, 1, x[[2]]}, {0, 0, 0, 1}};
Xmat[x_] := {{1, 0, 0, x[[2]]}, {0, Cos[x[[1]]], -Sin[x[[1]]], 0},
   {0, Sin[x[[1]]], Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Ymat[x_] := {{Cos[x[[1]]], 0, Sin[x[[1]]], 0}, {0, 1, 0, x[[2]]},
   {-Sin[x[[1]]], 0, Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Disp[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], x[[2]]},
   {Sin[x[[1]]], Cos[x[[1]]], x[[3]]}, {0, 0, 1}};
rem = Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}];
```

Synthesis Equations

These are the equations that will be used in the synthesis.

θ: input values Equivalent to $\theta 1$

ϕ: output values equavalent to $\theta 2$mobile

s:

γ: moving revolute joint equivalent to $\theta 3$

```
RSSPin = {{θ1, θ2, θ3, θ4, θ5, θ6, θ7},
   {s1, s2, s3, s4, s5, s6, s7}, {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6, ϕ7}};
```

Input Constants

```
d1 =.; γ1 =.; d3 =.; a1 =.; a2 =.; α1 =.; α2 =.;
```

Homogeneous Transforms to get to points O, A, D

```
Omat = Table[Zmat[{RSSPin[[1, i]], d1}], {i, 7}];
```

```
Amat =
  Table[Zmat[{RSSPin[[1, i]], d1}].Xmat[{α1, a1}].Zmat[{γ1, RSSPin[[2, i]]}], {i, 7}];
```

```
Dmat = Table[Zmat[{RSSPin[[1, i]], d1}].Xmat[{α1, a1}].
    Zmat[{γ1, RSSPin[[2, i]]}].Xmat[{α2, a2}].Zmat[{RSSPin[[3, i]], d3}], {i, 7}];
```

```
Amat2 = Table[Zmat[{0, 0}].Xmat[{α1, 0}].Zmat[{γ1, RSSPin[[2, i]]}], {i, 7}];
```

```
Bmat2 = Map[Zmat[{#1, d1}] &, RSSPin[[1]]];
```

Analysis Equations

These are the equations that will be used in the Analysis.

Analysis Equation 1 - first four bar with and revolute joints AB and SS joint BC

```
analysiseqn1 = ExpandAll[TrigExpand[Dot[{u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].{x1, y1, z1, 1}, {u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].{x1, y1, z1, 1}]] - b^2];
Acoeff1 = Coefficient[analysiseqn1, d2, 2];
Bcoeff1 = Coefficient[analysiseqn1, d2, 1];
Ccoeff1 = Coefficient[analysiseqn1, d2, 0];
seqn1 = (-Bcoeff1 + Sqrt[Bcoeff1^2 - 4 * Acoeff1 * Ccoeff1]) / (2 * Acoeff1);
seqn2 = (-Bcoeff1 - Sqrt[Bcoeff1^2 - 4 * Acoeff1 * Ccoeff1]) / (2 * Acoeff1);
```

These are the equations for converting the initial coordinates of the spherical joint B to the frame of revolute joint A.

B = (x1,y1,z1)
C = (u1,v1,w1)

```
NewUVW1 = {u1, v1, w1};
```

```
NewXYZ1 = Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}]].{x1, y1, z1, 1}.rem;
```

```
analysiseqn2 = ExpandAll[
   TrigExpand[Dot[{u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].
         Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1},
      {u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1}]] - b^2];
Acoeff2 = Coefficient[analysiseqn2, Cos[ϕ]];
Bcoeff2 = Coefficient[analysiseqn2, Sin[ϕ]];
Ccoeff2 = Simplify[analysiseqn2 - Acoeff2 * Cos[ϕ] - Bcoeff2 * Sin[ϕ]];
ϕeqn1 = ArcTan[Acoeff2, Bcoeff2] + ArcCos[-Ccoeff2 / Sqrt[Acoeff2^2 + Bcoeff2^2]];
ϕeqn2 = ArcTan[Acoeff2, Bcoeff2] - ArcCos[-Ccoeff2 / Sqrt[Acoeff2^2 + Bcoeff2^2]];
```

These equations convert the coordinates of spherical joint E from the global frame to the frame of revolute joint D

E = (x2,y2,z2)
F = (u2,v2,w2)

```
NewUVW2 = {u2, v2, w2};
```

```
NewXYZ2 =
  Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}]].
   {x2, y2, z2, 1}.rem;
```

Continuity Test Equations

Location of each of the joints

```
Ocoord = Zmat[{θ, d1}].{0, 0, 0, 1};
Acoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].{0, 0, 0, 1};
Bcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].{x1, y1, z1, 1};
Ccoord = {u1, v1, w1, 1};
Dcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].
   Zmat[{γ1, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].{0, 0, 0, 1};
Ecoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ1, d2}].
   Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1};
Fcoord = {u2, v2, w2, 1};
```

OA link length

```
OAlink = Norm[Ocoord.rem - Acoord.rem];
```

AB link length

```
ABlink = Norm[Acoord.rem - Bcoord.rem];
```

AD Link length

```
ADlink = Norm[Dcoord.rem - Acoord.rem];
```

DE Link length

```
DElink = Norm[Ecoord.rem - Dcoord.rem];
```

OC Link Length

```
OClink = Norm[Ocoord.rem - Ccoord.rem];
```

OF Link Length

```
OFlink = Norm[Ocoord.rem - Fcoord.rem];
```

BD Link Length

```
BDlink = Norm[Bcoord.rem - Dcoord.rem];
```

FC Link Length

```
FClink = Norm[Fcoord.rem - Ccoord.rem];
```

Functions

Other Modules

Synthesis Modules

The synthesis modules below is very similar to that of the planar four bar synthesis method by khaus-tub. It follows the following steps

1.) Collect the input values and constants.

2.) Substitute into the homogeneous transform matricies

3.) Calculate the Relative Transform Matricies which are relative to the location of the first point (transform matrix)

4.) Define the coordinates of the moving pivots and stationary pivot which are being solved for. (W and G respectively)

5.) Set up the Constrain Equations. The link length between the two moving pivots must remain constant. Using the relative transform matricies the two defined coordiantes can be moved through space.

6.) Set up the Design Equations. Subtract the first constraint equation from the remaining constraint equations to remove the unknown link length variable.

7.) Solve the design equations. For a spatial four bar function generator there are seven sets of inputs and outputs, so there will be six design equations which coorrelate with the six unknown coordinates that need to be solved for.

8.) Keep only the real numbered solutions.

9.) Calculate the distance between the points. (I call this leg lengths, but this is not actually the leg lengths used in the Analysis. Only the legth between the points remains the same. The input crank and output are actually the distance the respective points are from the axis of rotation. I use the leg lengths as the "title" for each set of points.)

10.) The solutions of of the coordinates and "leg lengths" are stored.

11.) The solutions are plugged back into the constrain equations to ensure that all values are returned as zero.

```
FirstRSSPSynthesis6[thetas_, ss_, Const_] :=
 Module[{RSSPsubs, Ainput2useB, RSSPrelA, W, (*x,y,z,*)G,
   (*u,v,w,*)a, c, b, R, RSSPConstraintEqn, RSSPDesignEqn, numsols,
   realnumsols, numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSPsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6}, {s1, s2, s3, s4, s5, s6},
       {d1, γ1, d3, a1, a2, α1, α2}}] → Flatten[{thetas, ss, Const}]];
  Ainput2useB = Amat /. RSSPsubs;
  RSSPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, 0, 1};
  RSSPConstraintEqn =
   Table[Chop[Dot[G - RSSPrelA[[i]].W, G - RSSPrelA[[i]].W] - R^2], {i, 6}];
  RSSPDesignEqn = Table[Chop[Expand[
      RSSPConstraintEqn[[i + 1]] - RSSPConstraintEqn[[1]]]], {i, 5}];
  numsols = NSolve[RSSPDesignEqn == {0, 0, 0, 0, 0}, {u, v, x, y, z}];

  (*Print[Chop[RSSPrelA]];
  Print[RSSPConstraintEqn];
  Print[RSSPDesignEqn];
  Print[numsols];*)
  (*Print[RSSPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u1,v1,w1,x1,y1,z1}]];*)
  (*Print[numsols];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + 0 + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {BC}*)
  leglengths =
   Table[Norm[({u, v, 0} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, 0, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((RSSPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
SecondRSSPSynthesis6[thetas_, ss_, phis_, Const_] :=
 Module[{RSSPsubs, Dinput2useB, RSSPrelD, W, x, y, z, G, u, v, w,
   a, c, b, R, RSSPConstraintEqn, RSSPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSPsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6}, {s1, s2, s3, s4, s5, s6},
       {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6}, {d1, γ1, d3, a1, a2, α1, α2}}] →
     Flatten[{thetas, ss, phis, Const}]];
  Dinput2useB = Dmat /. RSSPsubs;
  RSSPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, 0, 1};
  RSSPConstraintEqn =
   Table[Chop[Dot[G - RSSPrelD[[i]].W, G - RSSPrelD[[i]].W] - R^2], {i, 6}];
  RSSPDesignEqn = Table[Chop[Expand[
      RSSPConstraintEqn[[i + 1]] - RSSPConstraintEqn[[1]]]], {i, 5}];
  numsols = NSolve[RSSPDesignEqn == {0, 0, 0, 0, 0}, {u, v, x, y, z}];

  (*Print[Chop[RSSPrelA]];
  Print[RSSPConstraintEqn];
  Print[RSSPDesignEqn];
  Print[numsols];*)
  (*Print[RSSPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u2,v2,w2,x2,y2,z2}]];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + 0 + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {EF}*)
  leglengths =
   Table[Norm[({u, v, 0} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, 0, x, y, z} /. numsols2use;
  legsubs = Table[Thread[b → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((RSSPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
DesignEquations[thetas_, ss_, phis_, Const_] :=
 Module[{RSSPsubs, Dinput2useB, Ainput2useB, RSSPrelD, RSSPrelA, W, G, W2, G2, a, c,
   b, R, RSSPConstraintEqn, RSSPConstraintEqn2, RSSPDesignEqn, RSSPDesignEqn2,
   numsols, realnumsols, numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  RSSPsubs = Thread[Flatten[{{θ1, θ2, θ3, θ4, θ5, θ6}, {s1, s2, s3, s4, s5, s6},
       {ϕ1, ϕ2, ϕ3, ϕ4, ϕ5, ϕ6}, {d1, γ1, d3, a1, a2, α1, α2}}] →
     Flatten[{thetas, ss, phis, Const}]];

  Ainput2useB = Amat /. RSSPsubs;
  RSSPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  RSSPConstraintEqn =
   Table[Chop[Dot[G - RSSPrelA[[i]].W, G - RSSPrelA[[i]].W] - R^2], {i, 6}];
  RSSPDesignEqn = Table[Chop[Expand[
      RSSPConstraintEqn[[i + 1]] - RSSPConstraintEqn[[1]]]], {i, 5}];

  Dinput2useB = Dmat /. RSSPsubs;
  RSSPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W2 = {m, n, o, 1};
  G2 = {p, q, r, 1};
  RSSPConstraintEqn2 =
   Table[Chop[Dot[G2 - RSSPrelD[[i]].W2, G2 - RSSPrelD[[i]].W2] - R^2], {i, 6}];
  RSSPDesignEqn2 = Table[Chop[
     Expand[RSSPConstraintEqn2[[i + 1]] - RSSPConstraintEqn2[[1]]]], {i, 5}];
  Print[{RSSPDesignEqn, RSSPDesignEqn2}];
  {NumberForm[RSSPDesignEqn, 2], NumberForm[RSSPDesignEqn2, 2]}
 ]
```

Analysis Modules

The analysis is performed in a similar way as the planar four bar with some additional rotations and translations implemented. The general procedures are essentially the same for the two analysis procedures.

1.) Initial check to skip the analysis if there are no solutions from the synthesis.

2.) Import the linkage information from the synthesis

3.) Calculate the displacement and link length values given the imported coordinates.

4.) Check each branch by subtracting the input phi values from the calculated phi values based off the input theta.

5.) The solutions that do not branch are saved.

6.) solutions that nearly do not branch saved. these are saved for potential future optimization, but are not used currently.

7.) The pertinent information is packaged together for the output.

```
FirstRSSPAnalysis6[acbin_, uvwxyzin_, thetas_, ss_, Const_] :=
 Module[{len, InputLength, RSSPsubs, Constsub, legsubs, uvwxyzsub,
```

```
 Branch1, Branch2, Branch1adj, Branch2adj, Branch1vals, Branch2vals,
 ValidSols, AltSols, Sols2use, coord2use, Branch2use, finalsols,
 Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
finalsols = {};
Altfinalsols = {};
Lastsols = {finalsols, Altfinalsols};
If[Length[acbin] == 0, Print["No Solutions found!"];
 Goto[noSolutions]];
InputLength = Length[thetas];
len = Length[uvwxyzin];
Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
uvwxyzsub = Table[Thread[{u1, v1, w1, x1, y1, z1} → uvwxyzin[[i]]], {i, len}];
NewABsub =
 Chop[Table[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub[[i]]),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub[[i]], d2 → (ss[[1]]),
           θ → (thetas[[1]])}])}]], {i, len}]];
legsubs = Table[b → acbin[[i]], {i, len}];
If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
      Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
    {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
 Goto[noSolutions]];
(*Print[NewABsub];*)
Branch1 =
 Quiet[N[Table[Mod[Round[Table[(seqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]],
            Constsub, θ → (thetas[[i]])}]) - (ss[[i]]),
       {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
Branch2 = Quiet[N[Table[Mod[Round[Table[(seqn2 /. Flatten[
           {legsubs[[j]], NewABsub[[j]], Constsub, θ → (thetas[[i]])}]) -
        (ss[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ gammas[[1]]}]*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
     Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
   1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
```

```
    Length[ValidSols]}];

  Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

  finalsols = Table[Thread[{b, u1, v1, w1, x1, y1, z1, braval1} → Flatten[
       {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

  Altfinalsols = If[Length[AltSols] > 0, {thetas, ss}, {}];
  Lastsols = {finalsols, Altfinalsols};
  Label[noSolutions];
  Lastsols]
```

```
SecondRSSPAnalysis6[acbin_, uvwxyzin_, thetas_, ss_, phis_, Const_] :=
 Module[{len, InputLength, RSSPsubs, Constsub, legsubs, uvwxyzsub,
   Branch1, Branch2, Branch1adj, Branch2adj, Branch1vals, Branch2vals,
   ValidSols, AltSols, Sols2use, coord2use, Branch2use, finalsols,
   Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
  finalsols = {};
  Altfinalsols = {};
  Lastsols = {finalsols, Altfinalsols};
  If[Length[acbin] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  InputLength = Length[thetas];
  len = Length[uvwxyzin];
  Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = Table[Thread[{u2, v2, w2, x2, y2, z2} → uvwxyzin[[i]]], {i, len}];
  (*Print[uvwxyzsub];*)
  NewABsub =
   Chop[Table[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub[[i]]),
          (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub[[i]], ϕ → phis[[1]],
              d2 → (ss[[1]]), θ → (thetas[[1]])}])}]], {i, len}]];
  legsubs = Table[b → acbin[[i]], {i, len}];
  If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
        Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
      {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
   Goto[noSolutions]];

  (*Print[NewABsub];*)
  Branch1 =
   Quiet[N[Table[Mod[Round[Table[(ϕeqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]],
               Constsub, θ → (thetas[[i]]), d2 → ss[[i]]}]) - (phis[[i]]),
         {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
  Branch2 = Quiet[N[Table[Mod[Round[Table[(ϕeqn2 /. Flatten[{legsubs[[j]],
               NewABsub[[j]], Constsub, θ → (thetas[[i]]), d2 → ss[[i]]}]) -
          (phis[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
  (*Print[NewABsub];*)
  (*Print[legsubs];*)
  (*Print[legsubs,uvwxyzsub];*)
  (*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
```

```
   ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ gammas[[1]]}]*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
     Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
   1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
   Length[ValidSols]}];

Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

finalsols = Table[Thread[{b, u2, v2, w2, x2, y2, z2, braval2} → Flatten[
     {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

Altfinalsols = If[Length[AltSols] > 0, {thetas, ss , phis}, {}];
Lastsols = {finalsols, Altfinalsols};
Label[noSolutions];
Lastsols]
```

Continuity Test

These check that the lengths of the links do not change through out its movements. The interval can be selected by setting the limit number. *** NOTE: The values all go from 0 to a ThetaMax... a Theta Min Value has not been implemented. ******

1.) Import the linkage information that passed through the analysis

2.) calculate the coordinates of the spherical joints in the correct reference. The moving joints should be in the frame of the revolute joint in which they rotate about.

*** In the second test tables of the γ and ϕ angles were calculated *****

3.) Calculate the initial link lengths

4.) Calculate the link lengths at the specified intervals

5.) Subtract the original link lengths from the subsequent link lengths

```
ContTest16[inputs_, Const_, thetas_, ss_, ThetaMin_, ThetaMax_] := Module[
  {len, Constsub, legsubs, uvwxyzsub, RSSPsubs, ThetaTable, NewABsub, limit, OClinklen,
   ABlinklen, OClen, ABlen, BClen, TestedSols, TableOfO, TableOfD, TableOfE, TableOfS},
  If[Length[inputs] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = inputs[[2 ;; 7]];
  legsubs = {b → (b /. inputs)};
  RSSPsubs = Constsub;
  limit = 200;

  ThetaTable = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  NewABsub =
   Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
          Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];

  OClinklen =
   OClink /. Flatten[{Constsub, NewABsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
  ABlinklen = ABlink /. Flatten[{Constsub, NewABsub,
      (d2 → ss[[1]]), (θ → thetas[[1]])}] ;

  OClen = N[Table[Norm[(((Ocoord - Ccoord) /. Flatten[{legsubs, NewABsub, RSSPsubs}]) /.
       {θ → ThetaTable[[n]]})], {n, limit}]];
  ABlen = N[Table[Norm[((((Bcoord - Acoord) /.
           {d2 → ((If[(braval1 /. inputs) == 1, seqn1, seqn2]) /.
               Flatten[{legsubs, RSSPsubs, NewABsub, θ → ThetaTable[[n]]}])}) /.
         Flatten[{legsubs, RSSPsubs, NewABsub}]) /. θ → ThetaTable[[
         n]])], {n, limit}]];
  BClen = N[Table[Norm[(((((Ccoord - Bcoord) /. {d2 →
              ((If[(braval1 /. inputs) == 1, seqn1, seqn2]) /.
                Flatten[{legsubs, RSSPsubs, NewABsub, θ → ThetaTable[[n]]}])}) /.
          Flatten[{legsubs, RSSPsubs, NewABsub}])) /. θ →
        ThetaTable[[n]])], {n, limit}]];
  (*Print[({OAlinklen,ABlinklen,b}/.legsubs)];
  Print[{OAlen,ABlen,BClen}];*)
  TestedSols = {Chop[{OClen, ABlen, BClen} - ({OClinklen, ABlinklen, b} /. legsubs)]};
  Label[noSolutions];
  TestedSols]
```

```
ContTest26[inputs1_, inputs2_, Const_, thetas_, ss_, phis_, ThetaMin_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1,
    NewABsub2, ThetaTable, STable, PhiTable, ADlinklen, DElinklen, limit,
    ADlen, DElen, EFlen, TestedSols, TableOfO, TableOfA, TableOfB, TableOfC},
   Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 =
    Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
          Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (ϕ → phis[[1]]), (d2 → (ss[[1]])), (θ → thetas[[1]])}])}]]];

   ThetaTable = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];
   STable = Table[(If[(braval1 /. inputs1) == 1, seqn1, seqn2] /.
       Flatten[{legsubs1, Constsub, NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];
   PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /. Flatten[{legsubs2,
         Constsub, NewABsub2, θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];

   ADlinklen = ADlink /. Flatten[{Constsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[
      {Constsub, NewABsub2, (d2 → ss[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];

   ADlen = N[Table[Norm[((Dcoord - Acoord) /. Flatten[{Constsub, (d2 → STable[[n]]),
            (ϕ → PhiTable[[n]])}]) /. {θ → ThetaTable[[n]]}], {n, limit}]];
   (*Print[Alen];*)

   DElen =
    N[Table[Norm[(Ecoord - Dcoord) /. Flatten[{Constsub, NewABsub2, (d2 → STable[[n]]),
          (ϕ → PhiTable[[n]]), (θ → ThetaTable[[n]])}]], {n, limit}]];

   EFlen =
    N[Table[Norm[(Fcoord - Ecoord) /. Flatten[{Constsub, NewABsub2, (d2 → STable[[n]]),
          (ϕ → PhiTable[[n]]), (θ → ThetaTable[[n]])}]], {n, limit}]];



   (*Print[{Alen,Blen,Clen}];*)
   TestedSols = {(*inputs2[[1;;3]],*)
     Chop[{ADlen, DElen, EFlen} - ({ADlinklen, DElinklen, b /. legsubs2})]};
   (*Print[TestedSols];*)
   TestedSols];
```

Plotting Solutions

This is the module used for plotting the solutions.

```
PlottingSolutions6[inputs1_, inputs2_, Const_,
  thetas_, ss_, phis_, Seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, RSSPsubs, NewABsub1,
   NewABsub2, ThetaTable, STable, PhiTable, limit, plotSeqn1calc, plotSeqn2calc,
   plotϕeqn1calc, plotϕeqn2calc, plotSeqn1, plotSeqn2, plotϕeqn1, plotϕeqn2,
   plotϴ, plotϕ, plotS, SPoints2Plot, ϕPoints2Plot, Splot, ϕplot},
  Constsub = Thread[{d1, ɣ1, d3, a1, a2, α1, α2} → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  RSSPsubs = Constsub;
  limit = 200;

  NewABsub1 =
   Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
          Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (ϴ → thetas[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} →
      Flatten[{(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (ϕ → phis[[1]]), (d2 → (ss[[1]])), (ϴ → thetas[[1]])}])}]]];

  ThetaTable = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  plotSeqn1calc =
   Table[(seqn1 /. Flatten[{legsubs1, Constsub, NewABsub1, ϴ → ThetaTable[[n]]}]),
    {n, limit}];
  plotSeqn2calc = Table[(seqn2 /. Flatten[
      {legsubs1, Constsub, NewABsub1, ϴ → ThetaTable[[n]]}]), {n, limit}];

  STable = Table[(If[(braval1 /. inputs1) == 1, seqn1, seqn2] /.
     Flatten[{legsubs1, RSSPsubs, NewABsub1, ϴ → ThetaTable[[n]]}]), {n, limit}];
  plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
       ϴ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
  plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
       ϴ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
  plotϴ = ThetaTable;

  plotS = N[(Seqn /. x → #) & /@ plotϴ];
  plotϕ = N[(phieqn /. x → #) & /@ plotϴ];

  SPoints2Plot = Table[{thetas[[i]], ss[[i]]}, {i, Length[thetas]}];
  ϕPoints2Plot = Table[
    {thetas[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, phis][[i]]},
    {i, Length[thetas]}];

  plotSeqn1 = plotSeqn1calc;
```

```
  plotSeqn2 = plotSeqn2calc;
  plotϕeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn1calc];
  plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];

  Splot = ListPlot[{MapThread[{#1, #2} &, {plotϴ, plotS}], MapThread[{#1, #2} &,
      {plotϴ, plotSeqn1}], MapThread[{#1, #2} &, {plotϴ, plotSeqn2}], SPoints2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "S Plot Comparison"];
  ϕplot = ListPlot[{MapThread[{#1, #2} &, {plotϴ, plotϕ}], MapThread[{#1, #2} &,
      {plotϴ, plotϕeqn1}], MapThread[{#1, #2} &, {plotϴ, plotϕeqn2}], ϕPoints2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "ϕ Plot Comparison"];
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm, Splot, ϕplot}
 ]
```

```
FirstPosition3DPlot[inputs1_, inputs2_, Const_, thetas_, ss_, phis_] :=
  Module[{Constsub, uvwxyzsub, InitialPos, ptO,
    ptA, ptB, ptC, ptD, ptE, ptF, TubeDia, SphereDia},
   Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub =  Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   InitialPos = Thread[{ϴ, d2, ϕ} → {thetas[[1]], ss[[1]], phis[[1]]}];
   ptO = Ocoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptA = Acoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptB = Bcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptC = Ccoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptD = Dcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptE = Ecoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptF = Fcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];

   Print[uvwxyzsub];

   TubeDia = 0.125;
   SphereDia = .25;
   Graphics3D[{Gray, Tube[{ptO, ptC, ptF}, TubeDia], Green,
     Tube[{ptO, ptA}, TubeDia], Blue, Tube[{ptB, ptA, ptD}, TubeDia],
     Yellow, Tube[{ptB, ptC}, TubeDia], Sphere[{ptB, ptC}, SphereDia],
     Red, Tube[{ptD, ptE}, TubeDia], Purple, Tube[{ptE, ptF}, TubeDia],
     Sphere[{ptE, ptF}, SphereDia], Orange, Tube[{{0, 0, 0}, ptO}, TubeDia / 2]}]
  ];
```

```
Plot4Paper[inputs1_, inputs2_, Const_,
  thetas_, ss_, phis_, Seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, RSSPsubs, NewABsub1, NewABsub2,
   ThetaTable, STable, PhiTable, limit, plotSeqn1calc, plotSeqn2calc,
   plotϕeqn1calc, plotϕeqn2calc, plotSeqn1, plotSeqn2, plotϕeqn1, plotϕeqn2,
   plotϴ, plotϕ, plotS, SPoints2Plot, ϕPoints2Plot, Splot, ϕplot, S4export,
```

```
 ϕ4export, RotateRetractPlot, Sϕplot, DesiredRotateRetractPlot},
Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
legsubs1 = {b → (b /. inputs1)};
legsubs2 = {b → (b /. inputs2)};
RSSPsubs = Constsub;
limit = 200;

NewABsub1 =
 Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
        Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} →
    Flatten[{(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (ϕ → phis[[1]]), (d2 → (ss[[1]])), (θ → thetas[[1]])}])}]]];

ThetaTable = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

plotSeqn1calc =
 Table[(seqn1 /. Flatten[{legsubs1, Constsub, NewABsub1, θ → ThetaTable[[n]]}]),
  {n, limit}];
plotSeqn2calc = Table[(seqn2 /. Flatten[
     {legsubs1, Constsub, NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];

STable = Table[(If[(braval1 /. inputs1) == 1, seqn1, seqn2] /.
    Flatten[{legsubs1, RSSPsubs, NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];
PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /. Flatten[{legsubs2,
      Constsub, NewABsub2, θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
plotθ = ThetaTable;

plotS = N[(Seqn /. x → #) & /@ plotθ];
plotϕ = N[(phieqn /. x → #) & /@ plotθ];

SPoints2Plot = Table[{thetas[[i]], ss[[i]]}, {i, Length[thetas]}];
ϕPoints2Plot = Table[
  {thetas[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, phis][[i]]},
  {i, Length[thetas]}];

plotSeqn1 = plotSeqn1calc;
plotSeqn2 = plotSeqn2calc;
plotϕeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn1calc];
plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];

Splot = ListPlot[{MapThread[{#1, #2} &, {plotθ, plotS}], MapThread[{#1, #2} &,
    {plotθ, plotSeqn1}], MapThread[{#1, #2} &, {plotθ, plotSeqn2}], SPoints2Plot},
  PlotLegends → {"Desired Slide", "Branch 1", "Branch 2", "Precision Points"},
```

```
  Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "S Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
ϕplot = ListPlot[{MapThread[{#1, #2} &, {plotϴ, plotϕ}], MapThread[{#1, #2} &,
    {plotϴ, plotϕeqn1}], MapThread[{#1, #2} &, {plotϴ, plotϕeqn2}], ϕPoints2Plot},
  PlotLegends → {"Desired Rotation Angle", "Branch 1", "Branch 2",
    "Precision Points"}, Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "ϕ Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
(*{{inputs1[[1]],inputs2[[1]]}//MatrixForm,γplot,ϕplot}*)

S4export = Show[Splot, ImageSize → Large,
  PlotLabel → "", FrameLabel → {Style["Crank Angle (Radians)", 18],
    Style["Slide Distance (in)", 18]}, Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];
ϕ4export = Show[ϕplot, ImageSize → Large, PlotLabel → "",
  FrameLabel → {Style["Crank Angle (Radians)", 18],
    Style["Rotation Angle (Radians)", 18]}, Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];

RotateRetractPlot = ListPlot[{MapThread[{#1, #2} &, {plotϴ, plotS}],
   MapThread[{#1, #2} &, {plotϴ, plotϕ}], SPoints2Plot, ϕPoints2Plot}, PlotLegends →
   {"Desired Slide Distance", "Desired Rotation Angle", "Slide Precision Points",
    "Rotation Precision Points"}, Joined → {True, True, False, False},
  PlotStyle → {Thick, Dashing[Medium], PointSize[.015], PointSize[.02]},
  ImageSize → Large, FrameLabel →
   {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
    {Style["Crank Angle (Radians)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

Sϕplot = ListPlot[{MapThread[{#1, #2} &, {plotϴ, plotS}],
   MapThread[{#1, #2} &, {plotϴ, plotϕ}], SPoints2Plot, ϕPoints2Plot,
   MapThread[{#1, #2} &, {plotϴ, PhiTable}], MapThread[{#1, #2} &, {plotϴ, STable}]},
  PlotLegends → {"Desired Slide Distance", "Desired Rotation Angle",
    "Slide Precision Points", "Rotation Precision Points", "Rotation Angle",
    "Slide Distance"}, Joined → {True, True, False, False, True, True},
  PlotStyle → {{Thick}, {Medium}, {PointSize[.015]}, {PointSize[.02]},
    {Dashing[Medium]}, {Dashing[Tiny]}}, ImageSize → Large, FrameLabel →
   {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
    {Style["Crank Angle (Radians)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

DesiredRotateRetractPlot = ListPlot[
   {MapThread[{#1, #2} &, {plotϴ, plotS}], MapThread[{#1, #2} &, {plotϴ, plotϕ}]},
   PlotLegends → {"Desired Slide Distance", "Desired Rotation Angle"},
```

```
    Joined → {True, True}, PlotStyle → {Thick, Dashing[Medium]},
    ImageSize → Large, FrameLabel →
     {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
      {Style["Crank Angle (Radians)", 18], None}},
    FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
    LabelStyle → Directive[FontFamily → "Times New Roman", 15] ]
  ×
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm,
   S4export, ϕ4export, RotateRetractPlot, Sϕplot}
]
```

Ranking Solutions

The LinkRatio module calculates all of the link lengths and then calculates the ratio between the longest and shortest link.

```
LinkRatio[inputs1_, inputs2_, Const_, thetas_, ss_, phis_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 =
    Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
          Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
            (ϕ → phis[[1]]), (d2 → (ss[[1]])), (θ → thetas[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   ABlinklen =
    ABlink /. Flatten[{Constsub, NewABsub1, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (d2 → ss[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2}];
   BDlinklen =
    BDlink /. Flatten[{Constsub, NewABsub1, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   Max[LinkLengths] / Min[LinkLengths]
  ];
```

The CurveComparison module find the root mean square difference between the γ and ϕ equations given in the input and the

```
CurveComparison[inputs1_, inputs2_, Const_,
   thetas_, ss_, phis_, seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
  Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, RSSPsubs, ThetaTable,
    NewABsub1, NewABsub2, STable, PhiTable, limit, plotseqn1calc, plotseqn2calc,
    plotϕeqn1calc, plotϕeqn2calc, plotseqn1, plotseqn2, plotϕeqn1, plotϕeqn2, plotθ,
    plotϕ, plots, sPoints2Plot, ϕPoints2Plot, splot, ϕplot, SError, PhiError},
   Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   RSSPsubs = Constsub;
   limit = 200;
```

```
ThetaTable = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

NewABsub1 =
 Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
        Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
      {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (ϕ → phis[[1]]), (d2 → (ss[[1]])), (θ → thetas[[1]])}])}]]];

plotseqn1calc = Table[(seqn1 /.
    Flatten[{legsubs1, Constsub, NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];
plotseqn2calc = Table[(seqn2 /. Flatten[{legsubs1, Constsub,
      NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];

STable = Table[(If[(braval1 /. inputs1) == 1, seqn1, seqn2] /.
    Flatten[{legsubs1, RSSPsubs, NewABsub1, θ → ThetaTable[[n]]}]), {n, limit}];
PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /. Flatten[{legsubs2,
      Constsub, NewABsub2, θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];

plotϕeqn1calc = Table[(ϕeqn1 /. Flatten[{legsubs2, Constsub,
      NewABsub2, θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
plotϕeqn2calc = Table[(ϕeqn2 /. Flatten[{legsubs2, Constsub, NewABsub2,
      θ → ThetaTable[[n]], d2 → STable[[n]]}]), {n, limit}];
plotθ = ThetaTable;

plots = N[(seqn /. x → #) & /@ plotθ];
plotϕ = N[(phieqn /. x → #) & /@ plotθ];

sPoints2Plot = Table[{thetas[[i]], ss[[i]]}, {i, Length[thetas]}];
ϕPoints2Plot = Table[{thetas[[i]], phis[[i]]}, {i, Length[thetas]}];

plotseqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotseqn1calc];
plotseqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotseqn2calc];
plotϕeqn1 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn1calc];
plotϕeqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotϕeqn2calc];

SError = Sqrt[Total[MapThread[(#1 - #2)^2 &,
     {If[(braval1 /. inputs1) == 1, plotseqn1, plotseqn2], plots}]] / limit];
PhiError = Sqrt[Total[MapThread[(#1 - #2)^2 &,
     {If[(braval2 /. inputs2) == 1, plotϕeqn1, plotϕeqn2], plotϕ}]] / limit];
(*Print[{GammaError,PhiError}];*)
SError + PhiError
];
```

```
AllLinkLengths[inputs1_, inputs2_, Const_, thetas_, ss_, phis_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[{d1, γ1, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 =
    Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /.
          Flatten[{Constsub, uvwxyzsub, (d2 → ss[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
            (ϕ → phis[[1]]), (d2 → (ss[[1]])), (θ → thetas[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   ABlinklen =
    ABlink /. Flatten[{Constsub, NewABsub1, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (d2 → ss[[1]]), (θ → thetas[[1]]), (ϕ → phis[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2}];
   BDlinklen =
    BDlink /. Flatten[{Constsub, NewABsub1, (d2 → ss[[1]]), (θ → thetas[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   LinkLengths
  ];
```

Finding Solutions

```
SolutionPath[inputs1_, inputs2_, Const_, thetas_, gammas_, phis_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    ThetaTable, GammaTable, PhiTable, OutputTable, limit, GammaOut, PhiOut},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 = Chop[Thread[
      {u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[
            {Constsub, uvwxyzsub, (γ → gammas[[1]]), (θ → thetas[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
              (ϕ → phis[[1]]), (γ → (gammas[[1]])), (θ → thetas[[1]])}])}]]];

   ThetaTable = Table[ThetaMax * (n / limit), {n, limit}];
   GammaTable = Table[(If[(braval1 /. inputs1) == 1, γeqn1, γeqn2] /. Flatten[
        {legsubs1, Constsub, NewABsub1, θ → (ThetaMax * (n / limit))})]), {n, limit}];
   PhiTable = Table[(If[(braval2 /. inputs2) == 1, ϕeqn1, ϕeqn2] /.
       Flatten[{legsubs2, Constsub, NewABsub2,
         θ → (ThetaMax * (n / limit)), γ → GammaTable[[n]]}]), {n, limit}];


   GammaOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, GammaTable];
   PhiOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, PhiTable];

   OutputTable = {ThetaTable, GammaOut, PhiOut};
   OutputTable];
```

Inputs

Input Functions/Values

Polynomial Fitting

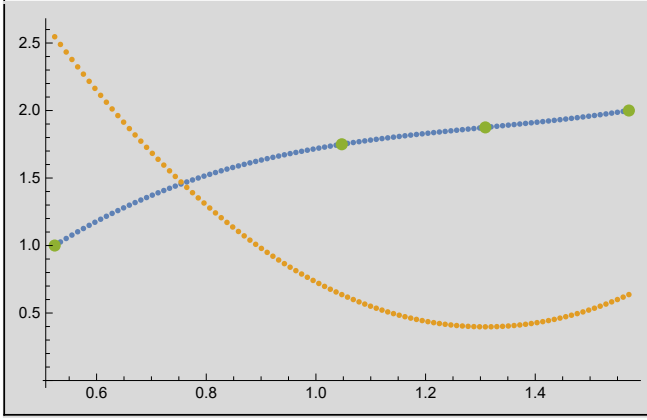
```
SposFunction = Fit[{{30 * Degree, 1}, {60 * Degree, 1.75},
   {75 * Degree, 1.875}, {90 * Degree, 2}}, {1, x, x^2, x^3}, x]
```

$$-1.25 + 6.3662\,x - 4.55945\,x^2 + 1.16106\,x^3$$

```
Sfitplot =
  Table[({x, SposFunction} /. (x → 30 * Degree + 60 * Degree * n / 100)), {n, 0, 100}];
```

```
dSfitplot =
  Table[({x, D[SposFunction, x]} /. (x → 30 * Degree + 60 * Degree * n / 100)), {n, 0, 100}];
```

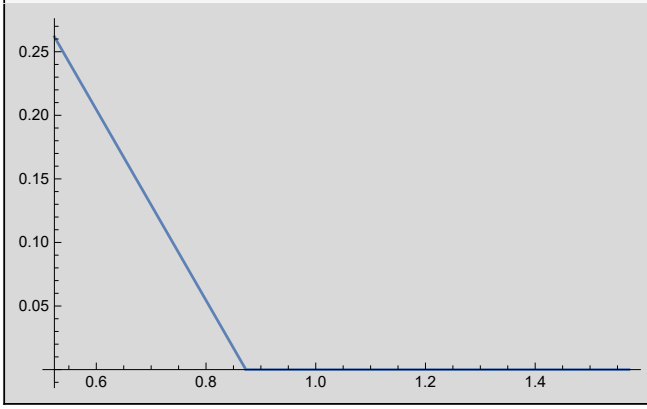
```
ListPlot[{Sfitplot, dSfitplot,
  {{30 * Degree, 1}, {60 * Degree, 1.75}, {75 * Degree, 1.875}, {90 * Degree, 2}}},
 PlotStyle → {Thick, Thick, PointSize[Large]}]
```



Piece Wise function for Phi Angle

```
PhiAngleFunction =
  Piecewise[{{-15 / 20 * x + (75 / 2) * Degree, x < 50 * Degree}, {0, x ≥ 50 * Degree}}];
```

```
Plot[PhiAngleFunction, {x, 30 * Degree, 90 * Degree}]
```



Precision Points

These are the functions that are needed to drive the slide and rotation

```
precisionptsθ = {90, 75, 60, 50, 45, 30} * Degree
```

```
{90 °, 75 °, 60 °, 50 °, 45 °, 30 °}
```

```
precisionptsS = Map[(SposFunction /. x → #) &, precisionptsθ]
```

```
{2., 1.875, 1.75, 1.60494, 1.5, 1.}
```

```
precisionptsϕ = Map[(PhiAngleFunction /. x → #) &, precisionptsθ]
```

$$\left\{0, 0, 0, 0, \frac{15\ ^\circ}{4}, 15\ ^\circ\right\}$$

Tolerances

These are the tolerances that are used for the randomization process

```
ThetaTols = {0, 5, 5, 5, 5, 5, 5} * Degree
```

```
{0, 5 °, 5 °, 5 °, 5 °, 5 °, 5 °}
```

```
STols = {.05, .05, .05, .05, .05, .05, .05}
```

```
{0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05}
```

```
PhiTols = {0, 0, 0, 0, 5, 5, 5} * Degree
```

```
{0, 0, 0, 0, 5 °, 5 °, 5 °}
```

D-H Table

```
d1val = 0;
γ1val = 0;
d3vals = 1;
a1vals = 1;
a2vals = 0;
α1val = Pi / 4.;
α2val = 0;
```

```
Const = N[{d1val, γ1val, d3vals, a1vals, a2vals, α1val, α2val}]
```

```
{0., 0., 1., 1., 0., 0.785398, 0.}
```

```
DHTable = {{θ, d1, α1, a1}, {γ1, d2, α2, a2}, {ϕ, d3, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} \theta & d1 & \alpha1 & a1 \\ \gamma1 & d2 & \alpha2 & a2 \\ \phi & d3 & 0 & 0 \end{pmatrix}$$

```
DHTable /. Thread[
  Flatten[{{d1, γ1, d3, a1, a2, α1, α2}, θ, d2, ϕ}] → Flatten[{Const, θ1, d2, θ3}]]
```

$$\begin{pmatrix} \theta_1 & 0. & 0.785398 & 1. \\ 0. & d_2 & 0. & 0. \\ \theta_3 & 1. & 0 & 0 \end{pmatrix}$$

```
TeXForm[DHTable /. Thread[
   Flatten[{{d1, γ1, d3, a1, a2, α1, α2}, θ, d2, ϕ}] → Flatten[{Const, θ1, d2, θ3}]]]
```

```
\left(
\begin{array}{cccc}
 \theta _1 & 0. & 0.785398 & 1. \\
 0. & d_2 & 0. & 0. \\
 \theta _3 & 1. & 0 & 0 \\
\end{array}
\right)
```

Position of Points

```
ConstSubs = Thread[Flatten[{{d1, γ1, d3, a1, a2, α1, α2}, {θ1, s1, ϕ1}}] →
   Flatten[{Const, precisionptsθ[[1]], precisionptsS[[1]], precisionptsϕ[[1]]}]]
```

```
{d1 → 0., γ1 → 0., d3 → 1., a1 → 1., a2 → 0.,
 α1 → 0.785398, α2 → 0., θ1 → 90 °, s1 → 2., ϕ1 → 0}
```

O Position and Vector

```
Opos = (Omat[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{0., 0., 0.}
```

```
Ovec = (Omat[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{0., 0., 1.}
```

```
Oxpos = (Omat[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{0., 1., 0.}
```

A Position and Vector

```
Apos = (Amat[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{1.41421, 1., 1.41421}
```

```
Avec = (Amat[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{2.12132, 1., 2.12132}
```

```
Axpos = (Amat[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{1.41421, 2., 1.41421}
```

D Postition and Vector

```
Dpos = (Dmat[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{2.12132, 1., 2.12132}
```

```
Dvec = (Dmat[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{2.82843, 1., 2.82843}
```

```
Dxpos = (Dmat[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{2.12132, 2., 2.12132}
```

Solvers

Main Solver

This is the solver. The solver first randomizes the θ and γ values. Once a set of θ and γ values are found that pass analysis they are used repeatedly for sets of randomized ϕ values.

This solver utilizes multithreading. It will still work if multithreading is not initialized, but it will run single threaded.

```
θstart = Pi / 2;
θend = Pi / 6;
Iterations = 500;
SetSharedVariable[IterationCounter];
IterationCounter = 0;
k1 = 1;
SecondIterations = 100;
SolutionList = Array[0 &, Iterations];
```

```
Randvals = {precisionptsθ, precisionptsS, precisionptsϕ};
SolutionList = ParallelTable[
   Print[i];
   If[i > 1, Randvals = RandomizeTps[precisionptsθ,
       precisionptsS, precisionptsϕ, ThetaTols, STols, PhiTols];,
    Randvals = {precisionptsθ, precisionptsS, precisionptsϕ};];
   (*First Four Bar Code *)
   test1 = Chop[FirstRSSPSynthesis6[Randvals[[1]], Randvals[[2]], Const]];
   testanal1 =
    FirstRSSPAnalysis6[test1[[1]], test1[[2]], Randvals[[1]], Randvals[[2]], Const];
   test1totals = If[Length[testanal1[[1]]] == 0, "No Solution",
     Table[Total[Flatten[ContTest16[testanal1[[1, i]], Const, Randvals[[1]],
         Randvals[[2]], θstart, θend]]], {i, Length[testanal1[[1]]]}]];
   Print[test1totals];
   test1sols = Table[If[test1totals[[i]] == 0, {testanal1[[1, i]],
       Randvals[[1]], Randvals[[2]]}, 0], {i, Length[test1totals]}];
   test1sols = DeleteCases[test1sols, 0];
   Print[test1sols];
   (*Second Four Bar Code *)
   SecondSolutionList = Array[0 &, SecondIterations];
   (*Creates array of zeros to store solutions from second four bar*)
   If[Length[test1sols] == 0, 0,
    IterationCounter++;
    Theta1sols = test1sols[[1, 2]]; (*Saves the randomized theta values*)
    S1sols = test1sols[[1, 3]]; (* Saves the randomized gamma values*)
    k2 = 1;
    RandPhi = precisionptsϕ;
    For[n2 = 0, n2 < SecondIterations, n2++,
     (*Print["Inside ", n2];*)
     test2 = Chop[SecondRSSPSynthesis6[Theta1sols, S1sols, RandPhi, Const]];
     testanal2 = SecondRSSPAnalysis6[
       test2[[1]], test2[[2]], Theta1sols, S1sols, RandPhi, Const];
     test2totals = If[Length[testanal2[[1]]] == 0, "No Solution",
       Table[Table[Total[Flatten[ContTest26[test1sols[[k, 1]], testanal2[[1, i]],
             Const, Theta1sols, S1sols, RandPhi, θstart, θend]]],
         {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}]];
     test2sols = Table[Table[If[test2totals[[k, i]] == 0,
         {test1sols[[k, 1]], testanal2[[1, i]], Theta1sols, S1sols, RandPhi}, 0],
        {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}];
     test2sols = DeleteCases[Flatten[test2sols, 1], 0];
     If[test2sols == {}, , SecondSolutionList[[k2]] = test2sols; k2++;];
     RandPhi = RandomizeTps1[precisionptsϕ, PhiTols];
    ](*End of second four bar FOR loop*)
   ]; (* End of Second four bar IF statement*)
   SecondSolutionList = DeleteCases[SecondSolutionList, 0];
   SecondSolutionList
   , {i, Iterations}];(*End of First Four bar FOR loop*)
Print["Stop 1"];
SolutionList = DeleteCases[SolutionList, (0 | {})];
Print["Stop 2"];
SolutionList = Flatten[SolutionList, 2];
```

```
Print["Stop 3"];
Print["Iterations: ",
  (Iterations - IterationCounter + IterationCounter * SecondIterations)];
```

```
SolutionList
```

```
Length[SolutionList]
```

Plotted Solutions

```
ParallelTable[PlottingSolutions6[SolutionList[[i, 1]], SolutionList[[i, 2]],
  Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]],
  SposFunction, PhiAngleFunction, Pi / 2, Pi / 6], {i, Length[SolutionList]}]
```

Ranking Solutions

Calculates the link length ratios of each solution.

```
Length[SolutionList]
```

```
SolsRankLinkLen =
  Sort[ParallelTable[{LinkRatio[SolutionList[[i, 1]], SolutionList[[i, 2]],
      Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]]],
    SolutionList[[i]]}, {i, Length[SolutionList]}]];
```

```
MaxRatio = 10;
```

```
SolsRankLinkLen2use =
  DeleteCases[Table[If[SolsRankLinkLen[[i, 1]] < MaxRatio, SolsRankLinkLen[[i, 2]], 0],
    {i, Length[SolsRankLinkLen]}], 0];
```

```
SolsRankLinkLen[[1]]
```

```
Append[SolsRankLinkLen2use[[1]], SolsRankLinkLen[[1, 1]]]
```

```
Length[SolsRankLinkLen2use]
```

Compares the RMS errors.

```
SortedSols = Sort[ParallelTable[
    {CurveComparison[SolsRankLinkLen2use[[i, 1]], SolsRankLinkLen2use[[i, 2]],
      Const, SolsRankLinkLen2use[[i, 3]], SolsRankLinkLen2use[[i, 4]],
      SolsRankLinkLen2use[[i, 5]], SposFunction, PhiAngleFunction, θstart, θend],
     SolsRankLinkLen2use[[i]]}, {i, Length[SolsRankLinkLen2use]}]];
```

```
SortedSols[[1]]
```

```
Length[SortedSols]
```

```
Min[Length[SortedSols], 100]
```

```
SortedSols[[1]]
```

```
ParallelTable[PlottingSolutions6[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
  Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]],
  SposFunction, PhiAngleFunction, θstart, θend], {i, Min[Length[SortedSols], 200]}]
```

```
ParallelTable[FirstPosition3DPlot[SortedSols[[i, 2, 1]],
  SortedSols[[i, 2, 2]], Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]],
  SortedSols[[i, 2, 5]]], {i, Min[Length[SortedSols], 200]}]
```

```
Length[SortedSols]
```

```
ParallelTable[{AllLinkLengths[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
   Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]]],
  Append[SortedSols[[i, 2]], Const]}, {i, Length[SortedSols]}]
```

Appendix C

PRP-2SS Valve Mechanism Mathematica Code

Spatial Six-Bar PRP

```
ParallelEvaluate[$KernelCount]
```

```
{16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16}
```

```
Kernels[]
```

```
{KernelObject[1, local], KernelObject[2, local],
 KernelObject[3, local], KernelObject[4, local], KernelObject[5, local],
 KernelObject[6, local], KernelObject[7, local], KernelObject[8, local],
 KernelObject[9, local], KernelObject[10, local], KernelObject[11, local],
 KernelObject[12, local], KernelObject[13, local],
 KernelObject[14, local], KernelObject[15, local], KernelObject[16, local]}
```

```
$ProcessorCount
```

```
16
```

Basic Functions and Constants

These equations are all calculated outside the module to reduce the number of repeated calculations

```
Zmat[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], 0, 0},
   {Sin[x[[1]]], Cos[x[[1]]], 0, 0}, {0, 0, 1, x[[2]]}, {0, 0, 0, 1}};
Xmat[x_] := {{1, 0, 0, x[[2]]}, {0, Cos[x[[1]]], -Sin[x[[1]]], 0},
   {0, Sin[x[[1]]], Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Ymat[x_] := {{Cos[x[[1]]], 0, Sin[x[[1]]], 0}, {0, 1, 0, x[[2]]},
   {-Sin[x[[1]]], 0, Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Disp[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], x[[2]]},
   {Sin[x[[1]]], Cos[x[[1]]], x[[3]]}, {0, 0, 1}};
rem = Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}];
```

Synthesis Equations

These are the equations that will be used in the synthesis.

θ: input values Equivalent to $\theta 1$

ϕ: output values equavalent to $\theta 2$mobile

s:

γ: moving revolute joint equivalent to $\theta 3$

```
PRPin = {{r1, r2, r3, r4, r5, r6}, {γ1, γ2, γ3, γ4, γ5, γ6}, {s1, s2, s3, s4, s5, s6}};
```

Input Constants

```
ConstSym = {θ, d2, ϕ, a1, a2, α1, α2}
```

```
{θ, d2, ϕ, a1, a2, α1, α2}
```

```
VariableSym = {d1, γ, d3}
```

```
{d1, γ, d3}
```

Homogeneous Transforms to get to points O, A, D

```
OmatPRP = Table[Zmat[{θ, PRPin[[1, i]]}], {i, 6}];
```

```
AmatPRP =
  Table[Zmat[{θ, PRPin[[1, i]]}].Xmat[{α1, a1}].Zmat[{PRPin[[2, i]], d2}], {i, 6}];
```

```
DmatPRP = Table[Zmat[{θ, PRPin[[1, i]]}].Xmat[{α1, a1}].
    Zmat[{PRPin[[2, i]], d2}].Xmat[{α2, a2}].Zmat[{ϕ, PRPin[[3, i]]}], {i, 6}];
```

Analysis Equations

These are the equations that will be used in the Analysis.

Analysis Equation 1 - first four bar with and revolute joints AB and ss joint BC

```
analysiseqn1 = ExpandAll[TrigExpand[Dot[{u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}, {u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}]] - b^2];
Acoeff1 = Coefficient[analysiseqn1, Cos[γ]];
Bcoeff1 = Coefficient[analysiseqn1, Sin[γ]];
Ccoeff1 = Simplify[analysiseqn1 - Acoeff1 * Cos[γ] - Bcoeff1 * Sin[γ]];
Anal1eqn1 =
  ArcTan[Acoeff1, Bcoeff1] + ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
Anal1eqn2 = ArcTan[Acoeff1, Bcoeff1] - ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
```

```
Simplify[Anal1eqn1]
```

```
ArcCos[ (-a1^2 + b^2 - d1^2 - d2^2 - u1^2 - v1^2 + 2 d1 w1 - w1^2 - x1^2 - y1^2 - 2 d2 z1 -
      z1^2 - 2 (d1 - w1) (d2 + z1) Cos[α1] + 2 Cos[θ] (a1 u1 - v1 (d2 + z1) Sin[α1]) +
      2 a1 v1 Sin[θ] + 2 d2 u1 Sin[α1] Sin[θ] + 2 u1 z1 Sin[α1] Sin[θ]) /
    (2 √((-a1 y1 + (u1 y1 - v1 x1 Cos[α1]) Cos[θ] + (d1 - w1) x1 Sin[α1] + v1 y1 Sin[θ] +
            u1 x1 Cos[α1] Sin[θ])^2 + (a1 x1 - (u1 x1 + v1 y1 Cos[α1]) Cos[θ] +
            (d1 - w1) y1 Sin[α1] - v1 x1 Sin[θ] + u1 y1 Cos[α1] Sin[θ])^2))] +
 ArcTan[a1 x1 - (u1 x1 + v1 y1 Cos[α1]) Cos[θ] + (d1 - w1) y1 Sin[α1] -
   v1 x1 Sin[θ] +
   u1 y1 Cos[α1] Sin[θ],
  -a1 y1 + (u1 y1 - v1 x1 Cos[α1]) Cos[θ] +
   (d1 - w1) x1 Sin[α1] + v1 y1 Sin[θ] +
   u1 x1 Cos[α1] Sin[θ]]
```

These are the equations for converting the initial coordinates of the spherical joint B to the frame of revolute joint A.

B = (x1,y1,z1)
C = (u1,v1,w1)

```
NewUVW1 = {u1, v1, w1};
```

```
NewXYZ1 = Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}]].{x1, y1, z1, 1}.rem;
```

Analysis Equation 2 - second four bar with revolute joints BD and joint2 joints EF

```
analysiseqn2 = ExpandAll[
   TrigExpand[Dot[{u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1},
      {u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1}]] - b^2];
Acoeff2 = Coefficient[analysiseqn2, d3, 2];
Bcoeff2 = Coefficient[analysiseqn2, d3, 1];
Ccoeff2 = Coefficient[analysiseqn2, d3, 0];
Anal2eqn1 = (-Bcoeff2 + Sqrt[Bcoeff2^2 - 4 * Acoeff2 * Ccoeff2]) / (2 * Acoeff2);
Anal2eqn2 = (-Bcoeff2 - Sqrt[Bcoeff2^2 - 4 * Acoeff2 * Ccoeff2]) / (2 * Acoeff2);
```

These equations convert the coordinates of spherical joint E from the global frame to the frame of revolute joint D

E = (x2,y2,z2)
F = (u2,v2,w2)

```
NewUVW2 = {u2, v2, w2};
```

```
NewXYZ2 =
  Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}]].
   {x2, y2, z2, 1}.rem;
```

Continuity Test Equations

Location of each of the joints

```
Ocoord = Zmat[{θ, d1}].{0, 0, 0, 1};
Acoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{0, 0, 0, 1};
Bcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1};
Ccoord = {u1, v1, w1, 1};
Dcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].
   Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].{0, 0, 0, 1};
Ecoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
   Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1};
Fcoord = {u2, v2, w2, 1};
```

```
Dcoord
```

$$\{a1\,\text{Cos}[\theta] + d2\,\text{Sin}[\alpha1]\,\text{Sin}[\theta] + a2\,(\text{Cos}[\gamma]\,\text{Cos}[\theta] - \text{Cos}[\alpha1]\,\text{Sin}[\gamma]\,\text{Sin}[\theta]) + d3\,(\text{Cos}[\alpha2]\,\text{Sin}[\alpha1]\,\text{Sin}[\theta] - \text{Sin}[\alpha2]\,(-\text{Cos}[\theta]\,\text{Sin}[\gamma] - \text{Cos}[\alpha1]\,\text{Cos}[\gamma]\,\text{Sin}[\theta])),\ -d2\,\text{Cos}[\theta]\,\text{Sin}[\alpha1] + a1\,\text{Sin}[\theta] + a2\,(\text{Cos}[\alpha1]\,\text{Cos}[\theta]\,\text{Sin}[\gamma] + \text{Cos}[\gamma]\,\text{Sin}[\theta]) + d3\,(-\text{Cos}[\alpha2]\,\text{Cos}[\theta]\,\text{Sin}[\alpha1] - \text{Sin}[\alpha2]\,(\text{Cos}[\alpha1]\,\text{Cos}[\gamma]\,\text{Cos}[\theta] - \text{Sin}[\gamma]\,\text{Sin}[\theta])),\ d1 + d2\,\text{Cos}[\alpha1] + d3\,(\text{Cos}[\alpha1]\,\text{Cos}[\alpha2] - \text{Cos}[\gamma]\,\text{Sin}[\alpha1]\,\text{Sin}[\alpha2]) + a2\,\text{Sin}[\alpha1]\,\text{Sin}[\gamma],\ 1\}$$

OA link length

```
OAlink = Norm[Ocoord.rem - Acoord.rem];
```

AB link length

```
ABlink = Norm[Acoord.rem - Bcoord.rem];
```

AD Link length

```
ADlink = Norm[Dcoord.rem - Acoord.rem];
```

DE Link length

```
DElink = Norm[Ecoord.rem - Dcoord.rem];
```

OC Link Length

```
OClink = Norm[Ocoord.rem - Ccoord.rem];
```

OF Link Length

```
OFlink = Norm[Ocoord.rem - Fcoord.rem];
```

BD Link Length

```
BDlink = Norm[Bcoord.rem - Dcoord.rem];
```

FC Link Length

```
FClink = Norm[Fcoord.rem - Ccoord.rem];
```

Functions

Other Modules

The Chebyshev Spacing module chooses the precision points to design around.

```
ChebyshevSpacing[NoOfPrecisionPts_, x0_, xnplus1_] := Module[{PrecisionPts, xn, j},
  PrecisionPts = ConstantArray[0, NoOfPrecisionPts + 2];
  PrecisionPts[[1]] = x0;
  PrecisionPts[[-1]] = xnplus1;
  (*Print[PrecisionPts];*)
  xn = 0.5 * (x0 + xnplus1) -
    0.5 * (xnplus1 - x0) * Cos[(π * (2 * j - 1)) / (2 * NoOfPrecisionPts)];
  Table[
   PrecisionPts[[i + 1]] = xn /. j → i;
   (*Print[PrecisionPts];*)
   , {i, NoOfPrecisionPts}];

  PrecisionPts
 ]
```

The randomizing module randomizes the inputs giving 7 random numbers within a given tolerance

```
RandomizeTps[thetaAngles_, Slength_, psiAngles_, ipTolerances_,
   sTolerances_, opTolerances_] := Module[{thetasNew, sNew, psisNew},

   thetasNew = psisNew = sNew = ConstantArray[0, 7];
   thetasNew = Table[RandomReal[{(thetaAngles[[i]] - ipTolerances[[i]]),
       (thetaAngles[[i]] + ipTolerances[[i]])}], {i, 7}];
   thetasNew[[1]] = If[thetasNew[[1]] < 0,
     RandomReal[{0, (thetaAngles[[1]] + ipTolerances[[1]])}], thetasNew[[1]]];
   thetasNew[[7]] = If[thetasNew[[7]] > 2 * Pi,
     RandomReal[{(thetaAngles[[7]] - ipTolerances[[7]]), 2 * Pi}], thetasNew[[7]]];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
       (psiAngles[[i]] + opTolerances[[i]])}], {i, 6}], 2 * Pi];

   sNew = Table[RandomReal[{(Slength[[i]] - sTolerances[[i]]),
       (Slength[[i]] + sTolerances[[i]])}], {i, 6}];

   {thetasNew, sNew, psisNew}];
```

Randomizing only revolute joint angles

```
RandomizeTps1[psiAngles_, opTolerances_, Num_] := Module[{psisNew},

   psisNew = ConstantArray[0, Num];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
       (psiAngles[[i]] + opTolerances[[i]])}], {i, Num}], 2 * Pi];
   psisNew = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, psisNew];

   psisNew];
```

Randomizing only prismatic joint displacements

```
RandomizeTps2[psiAngles_, opTolerances_, Num_] := Module[{psisNew},

   psisNew = ConstantArray[0, Num];

   psisNew = Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
       (psiAngles[[i]] + opTolerances[[i]])}], {i, Num}];
   (*psisNew=Map[If[#<-Pi,#+2*Pi,If[#>Pi,#-2*Pi,#]]&,psisNew];*)

   psisNew];
```

Calculate the Common Normal Lengths, Angular displacement and displacement along rotation axis find two d, one a, and one α.

Note that when the dot product is 0 the value of α is assigned to $\pi/2$.
ptc = p+tprime*S1

ptr = q+sprime*S2

```
CommonNorm[p_, S1_, q_, S2_] :=
  Module[{Cnorm, tprime, sprime, ptc, ptr, d1pt, d2pt, d1out, d2out, aout, αout},
   Cnorm = Normalize[Cross[S1, S2]];
   If[Cnorm == {0, 0, 0},
    d1out = 0;
    aout = 0;
    αout = 0;
    d2out = 0;
    d1pt = p;
    d2pt = q;
    Goto[SameLine];,
    tprime = Dot[Cross[(q - p), S2], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    sprime = Dot[Cross[(q - p), S1], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    ptc = p + tprime * S1;
    ptr = q + sprime * S2;
    d2pt = ptr;
    d1pt = ptc;
    d1out = (*Sign[Dot[S1,ptc-q]]**)Norm[p - ptc];
    d2out = (*Sign[Dot[S2,q-ptr]]**)Norm[ptr - q];
    aout = (*Sign[Dot[Cnorm,ptc-ptr]]**)Norm[ptc - ptr];
    If[Dot[S1, S2] == 0,
     αout = Pi / 2,
     αout = ArcTan[Dot[Cross[S1, S2], Cnorm] / Dot[S1, S2]]];];
   Label[SameLine];
   {d1out, aout, αout, d2out, d1pt, d2pt}];
```

Synthesis Modules

The synthesis modules below is very similar to that of the planar four bar synthesis method by khaus-tub. It follows the following steps

1.) Collect the input values and constants.

2.) Substitute into the homogeneous transform matricies

3.) Calculate the Relative Transform Matricies which are relative to the location of the first point (transform matrix)

4.) Define the coordinates of the moving pivots and stationary pivot which are being solved for. (W and G respectively)

5.) Set up the Constrain Equations. The link length between the two moving pivots must remain constant. Using the relative transform matricies the two defined coordiantes can be moved through space.

6.) Set up the Design Equations. Subtract the first constraint equation from the remaining constraint equations to remove the unknown link length variable.

7.) Solve the design equations. For a spatial four bar function generator there are seven sets of inputs and outputs, so there will be six design equations which coorrelate with the six unknown coordinates

that need to be solved for.
8.) Keep only the real numbered solutions.
9.) Calculate the distance between the points. (I call this leg lengths, but this is not actually the leg lengths used in the Analysis. Only the legth between the points remains the same. The input crank and output are actually the distance the respective points are from the axis of rotation. I use the leg lengths as the "title" for each set of points.)
10.) The solutions of of the coordinates and "leg lengths" are stored.
11.) The solutions are plugged back into the constrain equations to ensure that all values are returned as zero.

```
FirstPRPSynthesis[joint1_, joint2_, Const_] :=
 Module[{PRPsubs, Ainput2useB, PRPrelA, W, (*x,y,z,*)G, (*u,v,
   w,*)a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, test2, legsubs, subs, resub},
  PRPsubs = Thread[Flatten[{PRPin[[1 ;; 2]], ConstSym}] →
     Flatten[{joint1, joint2, Const}]];
  Ainput2useB = AmatPRP /. PRPsubs;
  PRPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelA[[i]].W, G - PRPrelA[[i]].W] - R^2], {i, 7}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 6}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0, 0},
    {u, v, w, x, y, z}(*,WorkingPrecision→ 40*)];

  resub = Table[PRPDesignEqn /. numsols[[i]], {i, Length[numsols]}];
  Print[resub];
  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];
  Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u1,v1,w1,x1,y1,z1}]];*)
  (*Print[numsols];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {BC}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  test2 = Chop[Table[((PRPDesignEqn) /. numsols2use[[i]]),
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test, test2}]
```

```
FirstPRPSynthesis6[joint1_, joint2_, Const_] :=
 Module[{PRPsubs, Ainput2useB, PRPrelA, W, (*x,y,z,*)G, (*u,v,
   w,*)a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, test2, legsubs, subs, resub},
  PRPsubs = Thread[Flatten[{PRPin[[1 ;; 2]], ConstSym}] →
     Flatten[{joint1, joint2, Const}]];
  Ainput2useB = AmatPRP /. PRPsubs;
  PRPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  z = 0;
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelA[[i]].W, G - PRPrelA[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0}, {u, v, w, x, y}
    (*,WorkingPrecision→ 40*)];
  resub = Table[PRPDesignEqn /. numsols[[i]], {i, Length[numsols]}];
  (*Print[resub];*)
  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];
  Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u1,v1,w1,x1,y1,z1}]];*)
  (*Print[numsols];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {BC}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  test2 = Chop[Table[((PRPDesignEqn) /. numsols2use[[i]]),
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test, test2}]
```

```
SecondPRPSynthesis[joint1_, joint2_, joint3_, Const_] :=
 Module[{PRPsubs, Dinput2useB, PRPrelD, W, x, y, z, G, u, v, w,
   a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  PRPsubs = Thread[Flatten[{PRPin, ConstSym}] →
     Flatten[{joint1, joint2, joint3, Const}]];
  Dinput2useB = DmatPRP /. PRPsubs;
  PRPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 7}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelD[[i]].W, G - PRPrelD[[i]].W] - R^2], {i, 7}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 6}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0, 0}, {u, v, w, x, y, z}];

  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];*)
  (*Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u2,v2,w2,x2,y2,z2}]];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {EF}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[b → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
SecondPRPSynthesis6[joint1_, joint2_, joint3_, Const_] :=
 Module[{PRPsubs, Dinput2useB, PRPrelD, W, x, y, z, G, u, v, w,
   a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  PRPsubs = Thread[Flatten[{PRPin, ConstSym}] →
     Flatten[{joint1, joint2, joint3, Const}]];
  Dinput2useB = DmatPRP /. PRPsubs;
  PRPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  z = 0;
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelD[[i]].W, G - PRPrelD[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0}, {u, v, w, x, y}];

  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];*)
  (*Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u2,v2,w2,x2,y2,z2}]];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {EF}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[b → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

```
DesignEquations[joint1_, joint2_, joint3_, Const_] :=
 Module[{PRPsubs, Dinput2useB, Ainput2useB, PRPrelD, PRPrelA, W, G, W2, G2, a, c, b,
   R, PRPConstraintEqn, PRPConstraintEqn2, PRPDesignEqn, PRPDesignEqn2, numsols,
   realnumsols, numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  PRPsubs = Thread[Flatten[{PRPin, ConstSym}] →
     Flatten[{joint1, joint2, joint3, Const}]];

  Ainput2useB = AmatPRP /. PRPsubs;
  PRPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelA[[i]].W, G - PRPrelA[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];

  Dinput2useB = DmatPRP /. PRPsubs;
  PRPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W2 = {m, n, o, 1};
  G2 = {p, q, r, 1};
  PRPConstraintEqn2 =
   Table[Chop[Dot[G2 - PRPrelD[[i]].W2, G2 - PRPrelD[[i]].W2] - R^2], {i, 6}];
  PRPDesignEqn2 = Table[Chop[Expand[
      PRPConstraintEqn2[[i + 1]] - PRPConstraintEqn2[[1]]]], {i, 5}];
  Print[{PRPDesignEqn, PRPDesignEqn2}];
  {NumberForm[PRPDesignEqn, 2], NumberForm[PRPDesignEqn2, 2]}
 ]
```

Analysis Modules

The analysis is performed in a similar way as the planar four bar with some additional rotations and translations implemented. The general procedures are essentially the same for the two analysis procedures.

1.) Initial check to skip the analysis if there are no solutions from the synthesis.

2.) Import the linkage information from the synthesis

3.) Calculate the displacement and link length values given the imported coordinates.

4.) Check each branch by subtracting the input phi values from the calculated phi values based off the input theta.

5.) The solutions that do not branch are saved.

6.) solutions that nearly do not branch saved. these are saved for potential future optimization, but are not used currently.

7.) The pertinent information is packaged together for the output.

```
FirstPRPAnalysis6[acbin_, uvwxyzin_, joint1_, joint2_, Const_] := Module[
  {len, InputLength, PRPsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
```

```
 Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
finalsols = {};
Altfinalsols = {};
Lastsols = {finalsols, Altfinalsols};
If[Length[acbin] == 0, Print["No Solutions found!"];
 Goto[noSolutions]];
InputLength = Length[joint1];
len = Length[uvwxyzin];
Constsub = Thread[ConstSym → Const];
uvwxyzsub = Table[Thread[{u1, v1, w1, x1, y1, z1} → uvwxyzin[[i]]], {i, len}];
NewABsub =
 Chop[Table[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub[[i]]),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub[[i]], VariableSym[[2]] →
            (joint2[[1]]), VariableSym[[1]] → (joint1[[1]])}])}]], {i, len}]];
legsubs = Table[b → acbin[[i]], {i, len}];
If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
      Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
    {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
 Goto[noSolutions]];
(*Print[NewABsub];*)
If[ToString[VariableSym[[2]]] == "γ",
 Branch1 =
  Quiet[N[Table[Mod[Round[Table[(Anal1eqn1 /. Flatten[{legsubs[[j]], NewABsub[[
              j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
         (joint2[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
 Branch2 = Quiet[N[Table[Mod[Round[Table[(Anal1eqn2 /. Flatten[{legsubs[[
              j]], NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
         (joint2[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];,
 Branch1 = Quiet[N[Table[Round[Table[(Anal1eqn1 /. Flatten[{legsubs[[j]],
            NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
        (joint2[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
 Branch2 = Quiet[N[Table[Round[Table[(Anal1eqn2 /. Flatten[{legsubs[[j]],
            NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
        (joint2[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ joint2[[1]]}]*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
```

```
      Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
  AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
     {i, Length[acbin]}], _Integer];
  Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
     1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
     Length[ValidSols]}];
  Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

  finalsols = Table[Thread[{b, u1, v1, w1, x1, y1, z1, braval1} → Flatten[
       {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

  Altfinalsols = If[Length[AltSols] > 0, {joint1, joint2}, {}];
  Lastsols = {finalsols, Altfinalsols};
  Label[noSolutions];
  Lastsols]
```

```
SecondPRPAnalysis[acbin_, uvwxyzin_, joint1_, joint2_, joint3_, Const_] := Module[
  {len, InputLength, PRPsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
   Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
  finalsols = {};
  Altfinalsols = {};
  Lastsols = {finalsols, Altfinalsols};
  If[Length[acbin] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  InputLength = Length[joint1];
  (*Print[InputLength];*)
  len = Length[uvwxyzin];
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Table[Thread[{u2, v2, w2, x2, y2, z2} → uvwxyzin[[i]]], {i, len}];
  (*Print[uvwxyzsub];*)
  NewABsub =
   Chop[Table[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub[[i]]),
         (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub[[i]], VariableSym[[3]] →
              joint3[[1]], VariableSym[[2]] → (joint2[[1]]),
            VariableSym[[1]] → (joint1[[1]])}])}]], {i, len}]];
  legsubs = Table[b → acbin[[i]], {i, len}];
  If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
       Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
     {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
   Goto[noSolutions]];
  (*Print[ToString[VariableSym[[3]]]=="ϕ"];*)
  (*Print[NewABsub];*)
  If[ToString[VariableSym[[3]]] == "ϕ",
   Branch1 = Quiet[N[Table[
       Mod[Round[Table[(Anal2eqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
              VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
           (joint3[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
   Branch2 = Quiet[N[Table[Mod[Round[Table[(Anal2eqn2 /.
```

```
            Flatten[{legsubs[[j]], NewABsub[[j]], Constsub, VariableSym[[1]] →
                (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) - (joint3[[i]]),
        {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];,
 Branch1 = Quiet[ N[Table[Round[Table[ (Anal2eqn1 /.
          Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
            VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
        (joint3[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
 Branch2 = Quiet[N[Table[Round[Table[ (Anal2eqn2 /.
          Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
            VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
        (joint3[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ joint2[[1]]}]*)
(*Print[Branch1];*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
     Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
   1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
   Length[ValidSols]}];

Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

finalsols = Table[Thread[{b, u2, v2, w2, x2, y2, z2, braval2} → Flatten[
     {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

Altfinalsols = If[Length[AltSols] > 0, {joint1, joint2 , joint3}, {}];
Lastsols = {finalsols, Altfinalsols};
Label[noSolutions];
Lastsols]
```

Continuity Test

These check that the lengths of the links do not change through out its movements. The interval can be selected by setting the limit number. *** NOTE: The values all go from 0 to a ThetaMax... a Theta Min Value has not been implemented. ******

1.) Import the linkage information that passed through the analysis

2.) calculate the coordinates of the spherical joints in the correct reference. The moving joints should be in the frame of the revolute joint in which they rotate about.

*** In the second test tables of the γ and ϕ angles were calculated *****

3.) Calculate the initial link lengths

4.) Calculate the link lengths at the specified intervals

5.) Subtract the original link lengths from the subsequent link lengths

```
ContTest16[inputs_, Const_, joint1_, joint2_, ThetaMin_, ThetaMax_] := Module[
  {len, Constsub, legsubs, uvwxyzsub, PRPsubs, Joint1Table, NewABsub, limit, OAlinklen,
   ABlinklen, OAlen, ABlen, BClen, TestedSols, TableOfO, TableOfD, TableOfE, TableOfS},
  If[Length[inputs] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = inputs[[2 ;; 7]];
  legsubs = {b → (b /. inputs)};
  PRPsubs = Constsub;
  limit = 200;

  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  NewABsub = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
  Print[NewABsub];
  OAlinklen = OAlink /. Flatten[{Constsub, NewABsub,
     (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
  ABlinklen = ABlink /. Flatten[{Constsub, NewABsub,
     (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}] ;

  OAlen = N[Table[Norm[(((Ocoord - Acoord) /. Flatten[{legsubs, NewABsub, PRPsubs}]) /.
       {VariableSym[[1]] → Joint1Table[[n]]})], {n, limit}]];
  ABlen = N[Table[Norm[((((Bcoord - Acoord) /. {VariableSym[[2]] → ((If[
                  (braval1 /. inputs) == 1, Anal1eqn1, Anal1eqn2]) /. Flatten[{legsubs,
                 PRPsubs, NewABsub, VariableSym[[1]] → Joint1Table[[n]]}])}) /.
         Flatten[{legsubs, PRPsubs, NewABsub}]) /. VariableSym[[1]] →
        Joint1Table[[n]])], {n, limit}]];
  BClen = N[Table[Norm[(((((Ccoord - Bcoord) /. {VariableSym[[2]] → ((If[(braval1 /.
                     inputs) == 1, Anal1eqn1, Anal1eqn2]) /. Flatten[{legsubs,
                  PRPsubs, NewABsub, VariableSym[[1]] → Joint1Table[[n]]}])}) /.
          Flatten[{legsubs, PRPsubs, NewABsub}])) /. VariableSym[[
         1]] → Joint1Table[[n]])], {n, limit}]];
  (*Print[({OAlinklen,ABlinklen,b}/.legsubs)];
  Print[{OAlen,ABlen,BClen}];*)
  TestedSols = {Chop[{OAlen, ABlen, BClen} - ({OAlinklen, ABlinklen, b} /. legsubs)]};
  Label[noSolutions];
  TestedSols]
```

```
ContTest26[inputs1_, inputs2_, Const_,
   joint1_, joint2_, joint3_, ThetaMin_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    Joint1Table, Joint2Table, Joint3Table, FClinklen, DElinklen, limit,
    FClen, DElen, EFlen, TestedSols, TableOfO, TableOfA, TableOfB, TableOfC},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
```

```
legsubs1 = {b → (b /. inputs1)};
legsubs2 = {b → (b /. inputs2)};
limit = 200;

NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
      (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
          (VariableSym[[1]] → joint1[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
          (VariableSym[[1]] → joint1[[1]])}])}]]];

Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];
Joint2Table =
 Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
      Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];

FClinklen = FClink /. Flatten[{Constsub, NewABsub2, NewABsub1,
    (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
    (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
    (VariableSym[[3]] → joint3[[1]])}];

FClen = N[Table[Norm[
    ((Fcoord - Ccoord) /. Flatten[{Constsub, NewABsub2, NewABsub1, (VariableSym[[
            2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]])}]) /.
     {VariableSym[[1]] → Joint1Table[[n]]}], {n, limit}]];
(*Print[Alen];*)

DElen =
 N[Table[Norm[(Ecoord - Dcoord) /. Flatten[{Constsub, NewABsub2, (VariableSym[[
          2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]]),
       (VariableSym[[1]] → Joint1Table[[n]])}]], {n, limit}]];

EFlen =
 N[Table[Norm[(Fcoord - Ecoord) /. Flatten[{Constsub, NewABsub2, (VariableSym[[
          2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]]),
       (VariableSym[[1]] → Joint1Table[[n]])}]], {n, limit}]];



(*Print[{Alen,Blen,Clen}];*)
TestedSols = {(*inputs2[[1;;3]],*)
  Chop[{FClen, DElen, EFlen} - ({FClinklen, DElinklen, b /. legsubs2})]};
(*Print[TestedSols];*)
TestedSols];
```

Plotting Solutions

```
PlottingSolutions6[inputs1_, inputs2_, Const_,
  joint1_, joint2_, joint3_, Seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
   Joint1Table, Joint2Table, Joint3Table, limit, plotJoint2eqn1calc,
   plotJoint2eqn2calc, plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1,
   plotJoint2eqn2, plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3,
   plotJoint2, Joint2Points2Plot, Joint3Points2Plot, Joint2plot, Joint3plot},
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  limit = 200;

  NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
      {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
           (VariableSym[[1]] → joint1[[1]])}])}]]];

  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  Joint2Table =
   Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
       Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
  plotJoint2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
  plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];

  plotJoint1 = Joint1Table;

  plotJoint3eqn1calc =
   Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
       Joint1Table[[n]], VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];
  plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[{legsubs2, Constsub,
       NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
       VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];

  plotJoint3 = N[(Seqn /. x → #) & /@ plotJoint1];
  plotJoint2 = N[(phieqn /. x → #) & /@ plotJoint1];

  Joint2Points2Plot = Table[{joint1[[i]], joint2[[i]]}, {i, Length[joint1]}];
  Joint3Points2Plot = Table[
```

```
    {joint1[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, joint3][[i]]},
    {i, Length[joint1]}];

  plotJoint3eqn1 = plotJoint3eqn1calc;
  plotJoint3eqn2 = plotJoint3eqn2calc;
  plotJoint2eqn1 =
   Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotJoint2eqn1calc];
  plotJoint2eqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &,
    plotJoint2eqn2calc];

  (*Print[plotJoint3eqn2calc];*)

  Joint2plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
     MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn1}],
     MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn2}], Joint2Points2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "γ Plot Comparison"];
  Joint3plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint3}],
     MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn1}],
     MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn2}], Joint3Points2Plot},
    PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
    Joined → {True, True, True, False},
    PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
    PlotLabel → "d3 Plot Comparison"];
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm, Joint2plot, Joint3plot}
 ]
```

```
FirstPosition3DPlot[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
  Module[{Constsub, uvwxyzsub, InitialPos, ptO,
    ptA, ptB, ptC, ptD, ptE, ptF, TubeDia, SphereDia},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub =  Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   InitialPos = Thread[VariableSym → {joint1[[1]], joint2[[1]], joint3[[1]]}];
   ptO = Ocoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptA = Acoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptB = Bcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptC = Ccoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptD = Dcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptE = Ecoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptF = Fcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];

   Print[uvwxyzsub];

   TubeDia = 0.125;
   SphereDia = .25;
   Graphics3D[{Gray, Tube[{ptO, ptC, ptF}, TubeDia], Green,
     Tube[{ptO, ptA}, TubeDia], Blue, Tube[{ptB, ptA, ptD}, TubeDia],
     Yellow, Tube[{ptB, ptC}, TubeDia], Sphere[{ptB, ptC}, SphereDia],
     Red, Tube[{ptD, ptE}, TubeDia], Purple, Tube[{ptE, ptF}, TubeDia],
     Sphere[{ptE, ptF}, SphereDia], Orange, Tube[{{0, 0, 0}, ptO}, TubeDia / 2]}]
  ];
```

```
Plot4Paper[inputs1_, inputs2_, Const_, joint1_,
  joint2_, joint3_, Seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, PRPsubs, NewABsub1,
   NewABsub2, Joint1Table, Joint2Table, Joint3Table, limit, plotJoin2eqn1calc,
   plotJoint2eqn2calc, plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1,
   plotJoint2eqn2, plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3,
   plotJoint2, Joint2Points2Plot, Joint3Points2Plot, Joint2plot, Joint3plot,
   S4export, ϕ4export, RotateRetractPlot, Sϕplot, DesiredRotateRetractPlot},
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  PRPsubs = Constsub;
  limit = 200;

  NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
      {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
           (VariableSym[[1]] → joint1[[1]])}])}]]];

  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];
```

```
plotJoin2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
      NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
      NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];

Joint2Table =
 Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
      PRPsubs, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      d2 → Joint2Table[[n]]}]), {n, limit}];
plotJoint3eqn1calc = Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2,
      VariableSym[[1]] → Joint1Table[[n]], d2 → Joint2Table[[n]]}]), {n, limit}];
plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[
     {legsubs2, Constsub, NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];
plotJoint1 = Joint1Table;

plotJoint2 = N[(Seqn /. x → #) & /@ plotJoint1];
plotJoint3 = N[(phieqn /. x → #) & /@ plotJoint1];

Joint2Points2Plot = Table[{joint1[[i]], joint2[[i]]}, {i, Length[joint1]}];
Joint3Points2Plot = Table[
  {joint1[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, joint3][[i]]},
  {i, Length[joint1]}];

plotJoint2eqn1 = plotJoin2eqn1calc;
plotJoint2eqn2 = plotJoint2eqn2calc;
plotJoint3eqn1 =
 Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotJoint3eqn1calc];
plotJoint3eqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &,
  plotJoint3eqn2calc];

Joint2plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn2}], Joint2Points2Plot},
  PlotLegends → {"Desired Slide", "Branch 1", "Branch 2", "Precision Points"},
  Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "S Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
Joint3plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint3}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn2}], Joint3Points2Plot},
  PlotLegends → {"Desired Rotation Angle", "Branch 1", "Branch 2",
    "Precision Points"}, Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "ϕ Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
```

```
(*{{inputs1[[1]],inputs2[[1]]}//MatrixForm,γplot,Joint3plot}*)

S4export = Show[Joint2plot, ImageSize → Large,
  PlotLabel → "", FrameLabel → {Style["Crank Angle (Radians)", 18],
    Style["Slide Distance (in)", 18]}, Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];
ϕ4export = Show[Joint3plot, ImageSize → Large, PlotLabel → "",
  FrameLabel → {Style["Crank Angle (Radians)", 18],
    Style["Rotation Angle (Radians)", 18]}, Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];

RotateRetractPlot =
 ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}], MapThread[{#1, #2} &,
    {plotJoint1, plotJoint3}], Joint2Points2Plot, Joint3Points2Plot}, PlotLegends →
   {"Desired Slide Distance", "Desired Rotation Angle", "Slide Precision Points",
    "Rotation Precision Points"}, Joined → {True, True, False, False},
  PlotStyle → {Thick, Dashing[Medium], PointSize[.015], PointSize[.02]},
  ImageSize → Large, FrameLabel →
   {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
    {Style["Crank Angle (Radians)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

Sϕplot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3}], Joint2Points2Plot,
   Joint3Points2Plot, MapThread[{#1, #2} &, {plotJoint1, Joint3Table}],
   MapThread[{#1, #2} &, {plotJoint1, Joint2Table}]}, PlotLegends →
   {"Desired Slide Distance", "Desired Rotation Angle", "Slide Precision Points",
    "Rotation Precision Points", "Rotation Angle", "Slide Distance"},
  Joined → {True, True, False, False, True, True},
  PlotStyle → {{Thick}, {Medium}, {PointSize[.015]}, {PointSize[.02]},
    {Dashing[Medium]}, {Dashing[Tiny]}}, ImageSize → Large, FrameLabel →
   {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
    {Style["Crank Angle (Radians)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

DesiredRotateRetractPlot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
    MapThread[{#1, #2} &, {plotJoint1, plotJoint3}]}, PlotLegends →
    {"Desired Slide Distance", "Desired Rotation Angle"}, Joined → {True, True},
   PlotStyle → {Thick, Dashing[Medium]}, ImageSize → Large, FrameLabel →
    {{Style["Slide Distance (in)", 18], Style["Rotation Angle (Radians)", 18]},
     {Style["Crank Angle (Radians)", 18], None}},
   FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
   LabelStyle → Directive[FontFamily → "Times New Roman", 15] ]
  ×
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm,
   S4export, ϕ4export, RotateRetractPlot, Sϕplot}
]
```

Ranking Solutions

The LinkRatio module calculates all of the link lengths and then calculates the ratio between the longest and shortest link.

```
LinkRatio[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
         (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
             (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
       {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
             (VariableSym[[1]] → joint1[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (VariableSym[[2]] → joint2[[1]]),
       (VariableSym[[1]] → joint1[[1]]), (VariableSym[[3]] → joint3[[1]])}];
   ABlinklen = ABlink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
       (VariableSym[[3]] → joint3[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[1]] → joint1[[1]])}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2,
       (VariableSym[[1]] → joint1[[1]])}];
   BDlinklen = BDlink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
       (VariableSym[[3]] → joint3[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   Max[LinkLengths] / Min[LinkLengths]
  ];
```

The CurveComparison module find the root mean square difference between the γ and ϕ equations

given in the input and the

```
CurveComparison[inputs1_, inputs2_, Const_,
    joint1_, joint2_, joint3_, seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
  Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, PRPsubs, Joint1Table,
    NewABsub1, NewABsub2, Joint2Table, Joint3Table, limit, plotJoint2eqn1calc,
    plotJoint2eqn2calc, plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1,
    plotJoint2eqn2, plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3, plotJoint2,
    Joint2Points2Plot, Joint3Points2Plot, splot, Joint3plot, SError, PhiError},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   PRPsubs = Constsub;
   limit = 200;

   Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
         (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
       {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
           (VariableSym[[1]] → joint1[[1]])}])}]]];

   Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

   Joint2Table =
    Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
       Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
   plotJoint2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
   plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];

   plotJoint1 = Joint1Table;

   plotJoint3eqn1calc =
    Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
        Joint1Table[[n]], VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];
   plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[{legsubs2, Constsub,
       NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
       VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];

   plotJoint3 = N[(seqn /. x → #) & /@ plotJoint1];
   plotJoint2 = N[(phieqn /. x → #) & /@ plotJoint1];

   Joint2Points2Plot = Table[{joint1[[i]], joint2[[i]]}, {i, Length[joint1]}];
```

```
Joint3Points2Plot = Table[
  {joint1[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, joint3][[i]]},
  {i, Length[joint1]}];

plotJoint3eqn1 = plotJoint3eqn1calc;
plotJoint3eqn2 = plotJoint3eqn2calc;
plotJoint2eqn1 =
 Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotJoint2eqn1calc];
plotJoint2eqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &,
  plotJoint2eqn2calc];

SError =
 Sqrt[Total[MapThread[(#1 - #2)^2 &, {If[(braval1 /. inputs1) == 1, plotJoint2eqn1,
       plotJoint2eqn2], plotJoint2}]] / limit];
PhiError = Sqrt[Total[MapThread[(#1 - #2)^2 &, {If[(braval2 /. inputs2) == 1,
       plotJoint3eqn1, plotJoint3eqn2], plotJoint3}]] / limit];
(*Print[{GammaError,PhiError}];*)
SError + PhiError
];
```

```
AllLinkLengths[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen, OClinklen,
    OFlinklen, BDlinklen, FClinklen, LinkLengths, Sublist},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
         (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
             (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
       {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
             (VariableSym[[1]] → joint1[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   Sublist =
    Flatten[{Constsub, NewABsub1, NewABsub2, (VariableSym[[2]] → joint2[[1]]),
      (VariableSym[[1]] → joint1[[1]]), (VariableSym[[3]] → joint3[[1]])}];
   OAlinklen = OAlink /. Sublist;
   ADlinklen = ADlink /. Sublist;
   ABlinklen = ABlink /. Sublist;
   DElinklen = DElink /. Sublist;
   OClinklen = OClink /. Sublist;
   OFlinklen = OFlink /. Sublist;
   BDlinklen = BDlink /. Sublist;
   FClinklen = FClink /. Sublist;

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   LinkLengths
  ];
```

Finding Solutions

```
SolutionPath[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    Joint1Table, Joint2Table, Joint3Table, OutputTable, limit, GammaOut, PhiOut},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} →
       Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub,
             (γ → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub),
        (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[3]] → joint3[[1]]),
            (γ → (joint2[[1]])), (VariableSym[[1]] → joint1[[1]])}])}]]];

   Joint1Table = Table[ThetaMax * (n / limit), {n, limit}];
   Joint2Table = Table[
     (If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1, Constsub,
         NewABsub1, VariableSym[[1]] → (ThetaMax * (n / limit))}]), {n, limit}];
   Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
       Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
          (ThetaMax * (n / limit)), γ → Joint2Table[[n]]}]), {n, limit}];


   GammaOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, Joint2Table];
   PhiOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, Joint3Table];

   OutputTable = {Joint1Table, GammaOut, PhiOut};
   OutputTable];
```

Inputs

Input Functions/Values

Polynomial Fitting

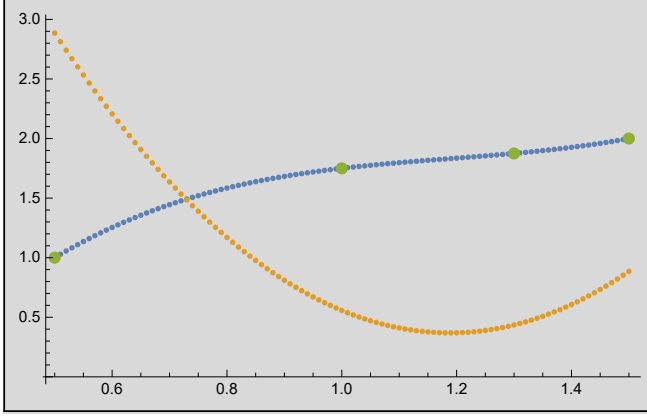
```
SposFunction = Fit[{{.5, 1}, {1, 1.75}, {1.3, 1.875}, {1.5, 2}}, {1, x, x^2, x^3}, x]
```

$$-1.57812 + 7.86979\,x - 6.3125\,x^2 + 1.77083\,x^3$$

```
Sfitplot = Table[({x, SposFunction} /. (x → .5 + 1 * n / 100)), {n, 0, 100}];
```

```
dSfitplot = Table[({x, D[SposFunction, x]} /. (x → .5 + 1 * n / 100)), {n, 0, 100}];
```

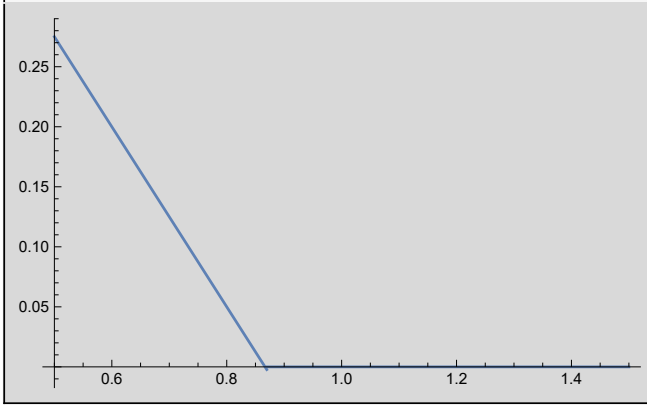
```
ListPlot[{Sfitplot, dSfitplot, {{.5, 1}, {1, 1.75}, {1.3, 1.875}, {1.5, 2}}},
 PlotStyle → {Thick, Thick, PointSize[Large]}]
```



Piece Wise function for Phi Angle

```
PhiAngleFunction = Piecewise[{{-15 / 20 * x + .65, x < .87}, {0, x ≥ .87}}];
```

```
Plot[PhiAngleFunction, {x, .5, 1.5}]
```



Precision Points

These are the functions that are needed to drive the slide and rotation

```
precisionptsd1 = {1.5, 1.3, 1, .9, .8, .5};
```

```
precisionptsd3 = Map[(SposFunction /. x → #) &, precisionptsd1]
```

```
{2., 1.875, 1.75, 1.6825, 1.58437, 1.}
```

```
precisionptsγ = Map[(PhiAngleFunction /. x → #) &, precisionptsd1]
```

```
{0, 0, 0, 0, 0.05, 0.275}
```

Tolerances

These are the tolerances that are used for the randomization process

```
d1Tols = {.05, .05, .05, .05, .05, .05, .05}
```

```
{0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05}
```

```
d3Tols = {.05, .05, .05, .05, .05, .05, .05}
```

```
{0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05}
```

```
GammaTols = N[{1, 1, 1, 15, 15, 15, 15} * Degree]
```

```
{0.0174533, 0.0174533, 0.0174533, 0.261799, 0.261799, 0.261799, 0.261799}
```

D-H Table

```
ConstSym
```

```
{θ, d2, ϕ, a1, a2, α1, α2}
```

```
θval = 0;
d2val = 1;
ϕvals = 0;
a1vals = 1;
a2vals = 0;
α1val = Pi / 4.;
α2val = 0;
```

```
(*θval=0;d2val=5;ϕvals=Pi/3;a1vals=1;a2vals=0;α1val=Pi/4.;α2val=0;*)
```

```
Const = N[{θval, d2val, ϕvals, a1vals, a2vals, α1val, α2val}]
```

```
{0., 1., 0., 1., 0., 0.785398, 0.}
```

```
DHTable = {{θ, d1, α1, a1}, {γ, d2, α2, a2}, {ϕ, d3, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} \theta & d1 & \alpha 1 & a1 \\ \gamma & d2 & \alpha 2 & a2 \\ \phi & d3 & 0 & 0 \end{pmatrix}$$

```
DHTable /. Thread[Flatten[{ConstSym, θ, γ, ϕ}] → Flatten[{Const, θ1, θ2, θ3}]]
```

$$\begin{pmatrix} 0. & d1 & 0.785398 & 1. \\ \theta_2 & 1. & 0. & 0. \\ 0. & d3 & 0 & 0 \end{pmatrix}$$

```
TeXForm[DHTable /. Thread[Flatten[{ConstSym, θ, γ, ϕ}] → Flatten[{Const, θ1, θ2, θ3}]]]
```

```
\left(
\begin{array}{cccc}
 0. & \text{d1} & 0.785398 & 1. \\
 \theta _2 & 1. & 0. & 0. \\
 0. & \text{d3} & 0 & 0 \\
\end{array}
\right)
```

Position of Points

```
ConstSubs = Thread[Flatten[{ConstSym, r1, γ1, s1}] →
   Flatten[{Const, precisionptsd1[[1]], precisionptsγ[[1]], precisionptsd3[[1]]}]]
```

```
{θ → 0., d2 → 1., ϕ → 0., a1 → 1., a2 → 0., α1 → 0.785398, α2 → 0., r1 → 1.5, γ1 → 0, s1 → 2.}
```

O Position and Vector

```
Opos = (OmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{0., 0., 1.5}
```

```
Ovec = (OmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{0., 0., 2.5}
```

```
Oxpos = (OmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{1., 0., 1.5}
```

A Position and Vector

```
Apos = (AmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{1., -0.707107, 2.20711}
```

```
Avec = (AmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{1., -1.41421, 2.91421}
```

```
Axpos = (AmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{2., -0.707107, 2.20711}
```

D Postition and Vector

```
Dpos = (DmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{1., -2.12132, 3.62132}
```

```
Dvec = (DmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{1., -2.82843, 4.32843}
```

```
Dxpos = (DmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{2., -2.12132, 3.62132}
```

Solvers

Main Solver

This is the solver. The solver first randomizes the θ and γ values. Once a set of θ and γ values are found that pass analysis they are used repeatedly for sets of randomized ϕ values.

This solver utilizes multithreading. It will still work if multithreading is not initialized, but it will run single threaded.

```
(*{precisionptsd1,precisionptsγ,precisionptsd3}={testRandValsGood1[[1,1;;6]],
   testRandValsGood1[[2,1;;6]],testRandValsGood1[[3,1;;6]]};*)
Iterations = 10000;
SetSharedVariable[IterationCounter];
IterationCounter = 0;
k1 = 1;
SecondIterations = 100;
NumInputs = 6;
```

```
SolutionList = Array[0 &, Iterations];
Randvals = {precisionptsd1, precisionptsγ, precisionptsd3};
SolutionList = ParallelTable[
   (*Print[i];*)
   If[i > 1, Randvals = {RandomizeTps2[precisionptsd1, d1Tols, NumInputs],
       RandomizeTps1[precisionptsγ, GammaTols, NumInputs],
       RandomizeTps2[precisionptsd3, d3Tols, NumInputs]};,
    Randvals = {precisionptsd1, precisionptsγ, precisionptsd3};];
   (*First Four Bar Code *)
   test1 = Chop[FirstPRPSynthesis6[Randvals[[1]], Randvals[[2]], Const]];
   testanal1 =
    FirstPRPAnalysis6[test1[[1]], test1[[2]], Randvals[[1]], Randvals[[2]], Const];
   test1totals = If[Length[testanal1[[1]]] == 0, "No Solution", Table[Total[
       Flatten[ContTest16[testanal1[[1, i]], Const, Randvals[[1]], Randvals[[2]],
         Randvals[[1, 1]], Randvals[[1, 6]]]]], {i, Length[testanal1[[1]]]}]];
   (*Print[test1totals];*)
   test1sols = Table[If[test1totals[[i]] == 0, {testanal1[[1, i]],
       Randvals[[1]], Randvals[[2]]}, 0], {i, Length[test1totals]}];
   test1sols = DeleteCases[test1sols, 0];
   (*Print[test1sols];*)
   (*Second Four Bar Code *)
   SecondSolutionList = Array[0 &, SecondIterations];
   (*Creates array of zeros to store solutions from second four bar*)
   If[Length[test1sols] == 0, 0,
    IterationCounter++;
    Theta1sols = test1sols[[1, 2]]; (*Saves the randomized theta values*)
    S1sols = test1sols[[1, 3]]; (* Saves the randomized gamma values*)
    k2 = 1;
    Randd3 = precisionptsd3;
    For[n2 = 0, n2 < SecondIterations, n2++,
     (*Print["Inside ", n2];*)
     test2 = Chop[SecondPRPSynthesis6[Theta1sols, S1sols, Randd3, Const]];
     testanal2 =
      SecondPRPAnalysis[test2[[1]], test2[[2]], Theta1sols, S1sols, Randd3, Const];
     test2totals = If[Length[testanal2[[1]]] == 0, "No Solution",
       Table[Table[Total[Flatten[ContTest26[test1sols[[k, 1]], testanal2[[1, i]],
            Const, Theta1sols, S1sols, Randd3, Randvals[[1, 1]], Randvals[[1, 6]]]]],
         {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}]];
     test2sols = Table[Table[If[test2totals[[k, i]] == 0,
         {test1sols[[k, 1]], testanal2[[1, i]], Theta1sols, S1sols, Randd3}, 0],
        {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}];
     test2sols = DeleteCases[Flatten[test2sols, 1], 0];
     If[test2sols == {}, , SecondSolutionList[[k2]] = test2sols; k2++;];
     Randd3 = RandomizeTps2[precisionptsd3, d3Tols, NumInputs];
    ](*End of second four bar FOR loop*)
   ]; (* End of Second four bar IF statement*)
   SecondSolutionList = DeleteCases[SecondSolutionList, 0];
   SecondSolutionList
   , {i, Iterations}];(*End of First Four bar FOR loop*)
Print["Stop 1"];
SolutionList = DeleteCases[SolutionList, (0 | {})];
```

```
Print["Stop 2"];
SolutionList = Flatten[SolutionList, 2];
Print["Stop 3"];
Print["Iterations: ",
  (Iterations - IterationCounter + IterationCounter * SecondIterations)];
```

```
SolutionList
```

```
Length[SolutionList]
```

Ranking Solutions

Calculates the link length ratios of each solution.

```
Length[SolutionList]
```

```
SolsRankLinkLen =
  Sort[ParallelTable[{LinkRatio[SolutionList[[i, 1]], SolutionList[[i, 2]],
      Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]]],
    SolutionList[[i]]}, {i, Length[SolutionList]}]];
```

```
MaxRatio = 5;
```

```
SolsRankLinkLen2use =
  DeleteCases[Table[If[SolsRankLinkLen[[i, 1]] < MaxRatio, SolsRankLinkLen[[i, 2]], 0],
    {i, Length[SolsRankLinkLen]}], 0];
```

```
SolsRankLinkLen[[1]]
```

```
Append[SolsRankLinkLen2use[[1]], SolsRankLinkLen[[1, 1]]]
```

```
Length[SolsRankLinkLen2use]
```

Compares the RMS errors.

```
SortedSols = Sort[ParallelTable[
    {CurveComparison[SolsRankLinkLen2use[[i, 1]], SolsRankLinkLen2use[[i, 2]],
      Const, SolsRankLinkLen2use[[i, 3]], SolsRankLinkLen2use[[i, 4]],
      SolsRankLinkLen2use[[i, 5]], SposFunction, PhiAngleFunction,
      Min[SolsRankLinkLen2use[[i, 3]]], Max[SolsRankLinkLen2use[[i, 3]]]],
     SolsRankLinkLen2use[[i]]}, {i, Length[SolsRankLinkLen2use]}]];
```

```
SortedSols[[1]]
```

```
Length[SortedSols]
```

```
Min[Length[SortedSols], 100]
```

```
2. * Pi
```

```
Mod[SortedSols[[1, 2, 4]], 2 * Pi]
```

```
ListPlot[MapThread[{#1, #2} &, {SortedSols[[1, 2, 3]], SortedSols[[1, 2, 4]]}]]
```

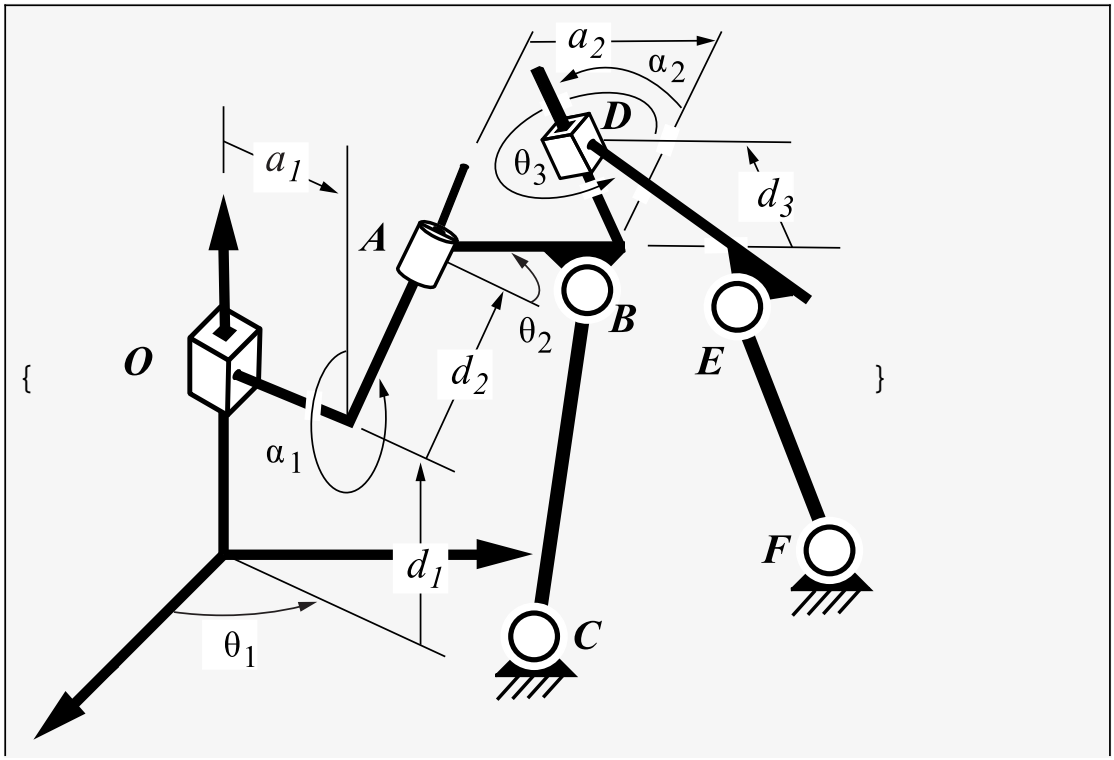
```
ParallelTable[PlottingSolutions6[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
  Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]],
  SposFunction, PhiAngleFunction, .5, 1.5], {i, Min[Length[SortedSols], 200]}]
```

```
ParallelTable[FirstPosition3DPlot[SortedSols[[i, 2, 1]],
  SortedSols[[i, 2, 2]], Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]],
  SortedSols[[i, 2, 5]]], {i, Min[Length[SortedSols], 200]}]
```

```
ParallelTable[{AllLinkLengths[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
   Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]]],
  Append[SortedSols[[i, 2]], Const]}, {i, Length[SortedSols]}]
```

Appendix D

PRP-2SS Flying Squirrel Mechanism Mathematica Code



Spatial Six-Bar PRP

```
ParallelEvaluate[$KernelCount]
```

```
{16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16}
```

```
Kernels[]
```

```
{KernelObject[1, local], KernelObject[2, local],
 KernelObject[3, local], KernelObject[4, local], KernelObject[5, local],
 KernelObject[6, local], KernelObject[7, local], KernelObject[8, local],
 KernelObject[9, local], KernelObject[10, local], KernelObject[11, local],
 KernelObject[12, local], KernelObject[13, local],
 KernelObject[14, local], KernelObject[15, local], KernelObject[16, local]}
```

```
$ProcessorCount
```

```
16
```

Basic Functions and Constants

These equations are all calculated outside the module to reduce the number of repeated calculations

In[1]:=
```
Zmat[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], 0, 0},
   {Sin[x[[1]]], Cos[x[[1]]], 0, 0}, {0, 0, 1, x[[2]]}, {0, 0, 0, 1}};
Xmat[x_] := {{1, 0, 0, x[[2]]}, {0, Cos[x[[1]]], -Sin[x[[1]]], 0},
   {0, Sin[x[[1]]], Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Ymat[x_] := {{Cos[x[[1]]], 0, Sin[x[[1]]], 0}, {0, 1, 0, x[[2]]},
   {-Sin[x[[1]]], 0, Cos[x[[1]]], 0}, {0, 0, 0, 1}};
Disp[x_] := {{Cos[x[[1]]], -Sin[x[[1]]], x[[2]]},
   {Sin[x[[1]]], Cos[x[[1]]], x[[3]]}, {0, 0, 1}};
rem = Transpose[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}}];
```

Synthesis Equations

These are the equations that will be used in the synthesis.

θ: input values Equivalent to $\theta 1$

ϕ: output values equavalent to $\theta 2$mobile

s:

γ: moving revolute joint equivalent to $\theta 3$

In[6]:=
```
PRPin = {{r1, r2, r3, r4, r5, r6}, {γ1, γ2, γ3, γ4, γ5, γ6}, {s1, s2, s3, s4, s5, s6}};
```

Input Constants

In[7]:=
```
ConstSym = {θ, d2, ϕ, a1, a2, α1, α2}
```

Out[7]=
```
{θ, d2, ϕ, a1, a2, α1, α2}
```

In[8]:=
```
VariableSym = {d1, γ, d3}
```

Out[8]=
```
{d1, γ, d3}
```

Homogeneous Transforms to get to points O, A, D

In[9]:=
```
OmatPRP = Table[Zmat[{θ, PRPin[[1, i]]}], {i, 6}];
```

In[10]:=
```
AmatPRP =
  Table[Zmat[{θ, PRPin[[1, i]]}].Xmat[{α1, a1}].Zmat[{PRPin[[2, i]], d2}], {i, 6}];
```

In[11]:=
```
DmatPRP = Table[Zmat[{θ, PRPin[[1, i]]}].Xmat[{α1, a1}].
    Zmat[{PRPin[[2, i]], d2}].Xmat[{α2, a2}].Zmat[{ϕ, PRPin[[3, i]]}], {i, 6}];
```

Analysis Equations

These are the equations that will be used in the Analysis.

Analysis Equation 1 - first four bar with and revolute joints AB and ss joint BC

In[12]:=
```
analysiseqn1 = ExpandAll[TrigExpand[Dot[{u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}, {u1, v1, w1, 1} -
       Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1}]] - b^2];
Acoeff1 = Coefficient[analysiseqn1, Cos[γ]];
Bcoeff1 = Coefficient[analysiseqn1, Sin[γ]];
Ccoeff1 = Simplify[analysiseqn1 - Acoeff1 * Cos[γ] - Bcoeff1 * Sin[γ]];
Anal1eqn1 =
  ArcTan[Acoeff1, Bcoeff1] + ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
Anal1eqn2 = ArcTan[Acoeff1, Bcoeff1] - ArcCos[-Ccoeff1 / Sqrt[Acoeff1^2 + Bcoeff1^2]];
```

These are the equations for converting the initial coordinates of the spherical joint B to the frame of revolute joint A.

B = (x1,y1,z1)

C = (u1,v1,w1)

In[19]:=
```
NewUVW1 = {u1, v1, w1};
```

In[20]:=
```
NewXYZ1 = Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}]].{x1, y1, z1, 1}.rem;
```

Analysis Equation 2 - second four bar with revolute joints BD and joint2 joints EF

In[21]:=
```
analysiseqn2 = ExpandAll[
   TrigExpand[Dot[{u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1},
      {u2, v2, w2, 1} - Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
        Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1}]] - b^2];
Acoeff2 = Coefficient[analysiseqn2, d3, 2];
Bcoeff2 = Coefficient[analysiseqn2, d3, 1];
Ccoeff2 = Coefficient[analysiseqn2, d3, 0];
Anal2eqn1 = (-Bcoeff2 + Sqrt[Bcoeff2^2 - 4 * Acoeff2 * Ccoeff2]) / (2 * Acoeff2);
Anal2eqn2 = (-Bcoeff2 - Sqrt[Bcoeff2^2 - 4 * Acoeff2 * Ccoeff2]) / (2 * Acoeff2);
```

These equations convert the coordinates of spherical joint E from the global frame to the frame of revolute joint D

E = (x2,y2,z2)

F = (u2,v2,w2)

In[27]:=
```
NewUVW2 = {u2, v2, w2};
```

In[28]:=
```
NewXYZ2 =
  Inverse[Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}]].
   {x2, y2, z2, 1}.rem;
```

Continuity Test Equations

Location of each of the joints

In[29]:=
```
Ocoord = Zmat[{θ, d1}].{0, 0, 0, 1};
Acoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{0, 0, 0, 1};
Bcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].{x1, y1, z1, 1};
Ccoord = {u1, v1, w1, 1};
Dcoord = Zmat[{θ, d1}].Xmat[{α1, a1}].
   Zmat[{γ, d2}].Xmat[{α2, a2}].Zmat[{ϕ, d3}].{0, 0, 0, 1};
Ecoord = Zmat[{θ, d1}].Xmat[{α1, a1}].Zmat[{γ, d2}].
   Xmat[{α2, a2}].Zmat[{ϕ, d3}].{x2, y2, z2, 1};
Fcoord = {u2, v2, w2, 1};
```

OA link length

In[37]:=
```
OAlink = Norm[Ocoord.rem - Acoord.rem];
```

AB link length

In[38]:=
```
ABlink = Norm[Acoord.rem - Bcoord.rem];
```

AD Link length

In[39]:=
```
ADlink = Norm[Dcoord.rem - Acoord.rem];
```

DE Link length

In[40]:=
```
DElink = Norm[Ecoord.rem - Dcoord.rem];
```

OC Link Length

In[41]:=
```
OClink = Norm[Ocoord.rem - Ccoord.rem];
```

OF Link Length

In[42]:=
```
OFlink = Norm[Ocoord.rem - Fcoord.rem];
```

BD Link Length

In[43]:=
```
BDlink = Norm[Bcoord.rem - Dcoord.rem];
```

FC Link Length

In[44]:=
```
FClink = Norm[Fcoord.rem - Ccoord.rem];
```

Functions

Other Modules

The Chebyshev Spacing module chooses the precision points to design around.

In[45]:=
```
ChebyshevSpacing[NoOfPrecisionPts_, x0_, xnplus1_] := Module[{PrecisionPts, xn, j},
  PrecisionPts = ConstantArray[0, NoOfPrecisionPts + 2];
  PrecisionPts[[1]] = x0;
  PrecisionPts[[-1]] = xnplus1;
  (*Print[PrecisionPts];*)
  xn = 0.5 * (x0 + xnplus1) -
    0.5 * (xnplus1 - x0) * Cos[(π * (2 * j - 1)) / (2 * NoOfPrecisionPts)];
  Table[
   PrecisionPts[[i + 1]] = xn /. j → i;
   (*Print[PrecisionPts];*)
   , {i, NoOfPrecisionPts}];

  PrecisionPts
 ]
```

The randomizing module randomizes the inputs giving 7 random numbers within a given tolerance

In[46]:=
```
RandomizeTps[thetaAngles_, Slength_, psiAngles_, ipTolerances_,
   sTolerances_, opTolerances_] := Module[{thetasNew, sNew, psisNew},

   thetasNew = psisNew = sNew = ConstantArray[0, 7];
   thetasNew = Table[RandomReal[{(thetaAngles[[i]] - ipTolerances[[i]]),
       (thetaAngles[[i]] + ipTolerances[[i]])}], {i, 7}];
   thetasNew[[1]] = If[thetasNew[[1]] < 0,
     RandomReal[{0, (thetaAngles[[1]] + ipTolerances[[1]])}], thetasNew[[1]]];
   thetasNew[[7]] = If[thetasNew[[7]] > 2 * Pi,
     RandomReal[{(thetaAngles[[7]] - ipTolerances[[7]]), 2 * Pi}], thetasNew[[7]]];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
        (psiAngles[[i]] + opTolerances[[i]])}], {i, 6}], 2 * Pi];

   sNew = Table[RandomReal[{(Slength[[i]] - sTolerances[[i]]),
       (Slength[[i]] + sTolerances[[i]])}], {i, 6}];

   {thetasNew, sNew, psisNew}];
```

Randomizing only revolute joint angles

In[47]:=
```
RandomizeTps1[psiAngles_, opTolerances_, Num_] := Module[{psisNew},

   psisNew = ConstantArray[0, Num];

   psisNew = Mod[Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
        (psiAngles[[i]] + opTolerances[[i]])}], {i, Num}], 2 * Pi];
   psisNew = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, psisNew];

   psisNew];
```

Randomizing only prismatic joint displacements

In[48]:=
```
RandomizeTps2[psiAngles_, opTolerances_, Num_] := Module[{psisNew},

   psisNew = ConstantArray[0, Num];

   psisNew = Table[RandomReal[{(psiAngles[[i]] - opTolerances[[i]]),
       (psiAngles[[i]] + opTolerances[[i]])}], {i, Num}];
   (*psisNew=Map[If[#<-Pi,#+2*Pi,If[#>Pi,#-2*Pi,#]]&,psisNew];*)

   psisNew];
```

Calculate the Common Normal Lengths, Angular displacement and displacement along rotation axis find two d, one a, and one α.

Note that when the dot product is 0 the value of α is assigned to $\pi/2$.
ptc = p+tprime*S1
ptr = q+sprime*S2

In[49]:=
```
CommonNorm[p_, S1_, q_, S2_] :=
  Module[{Cnorm, tprime, sprime, ptc, ptr, d1pt, d2pt, d1out, d2out, aout, αout},
   Cnorm = Normalize[Cross[S1, S2]];
   If[Cnorm == {0, 0, 0},
    d1out = 0;
    aout = 0;
    αout = 0;
    d2out = 0;
    d1pt = p;
    d2pt = q;
    Goto[SameLine];,
    tprime = Dot[Cross[(q - p), S2], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    sprime = Dot[Cross[(q - p), S1], Cnorm] / Dot[Cross[S1, S2], Cnorm];
    ptc = p + tprime * S1;
    ptr = q + sprime * S2;
    d2pt = ptr;
    d1pt = ptc;
    d1out = (*Sign[Dot[S1,ptc-q]]**)Norm[p - ptc];
    d2out = (*Sign[Dot[S2,q-ptr]]**)Norm[ptr - q];
    aout = (*Sign[Dot[Cnorm,ptc-ptr]]**)Norm[ptc - ptr];
    If[Dot[S1, S2] == 0,
     αout = Pi / 2,
     αout = ArcTan[Dot[Cross[S1, S2], Cnorm] / Dot[S1, S2]]];];
   Label[SameLine];
   {d1out, aout, αout, d2out, d1pt, d2pt}];
```

Synthesis Modules

The synthesis modules below is very similar to that of the planar four bar synthesis method by khaus-tub. It follows the following steps

1.) Collect the input values and constants.

2.) Substitute into the homogeneous transform matricies

3.) Calculate the Relative Transform Matricies which are relative to the location of the first point (transform matrix)

4.) Define the coordinates of the moving pivots and stationary pivot which are being solved for. (W and G respectively)

5.) Set up the Constrain Equations. The link length between the two moving pivots must remain constant. Using the relative transform matricies the two defined coordiantes can be moved through space.

6.) Set up the Design Equations. Subtract the first constraint equation from the remaining constraint equations to remove the unknown link length variable.

7.) Solve the design equations. For a spatial four bar function generator there are seven sets of inputs and outputs, so there will be six design equations which coorrelate with the six unknown coordinates that need to be solved for.

8.) Keep only the real numbered solutions.
9.) Calculate the distance between the points. (I call this leg lengths, but this is not actually the leg lengths used in the Analysis. Only the legth between the points remains the same. The input crank and output are actually the distance the respective points are from the axis of rotation. I use the leg lengths as the "title" for each set of points.)
10.) The solutions of of the coordinates and "leg lengths" are stored.
11.) The solutions are plugged back into the constrain equations to ensure that all values are returned as zero.

In[51]:=
```
FirstPRPSynthesis6[joint1_, joint2_, Const_] :=
 Module[{PRPsubs, Ainput2useB, PRPrelA, W, (*x,y,z,*)G, (*u,v,
   w,*)a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, test2, legsubs, subs, resub},
  PRPsubs = Thread[Flatten[{PRPin[[1 ;; 2]], ConstSym}] →
     Flatten[{joint1, joint2, Const}]];
  Ainput2useB = AmatPRP /. PRPsubs;
  PRPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  w = 0;
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelA[[i]].W, G - PRPrelA[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0}, {u, v, x, y, z}
    (*,WorkingPrecision→ 40*)];
  resub = Table[PRPDesignEqn /. numsols[[i]], {i, Length[numsols]}];
  (*Print[resub];*)
  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];
  Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u1,v1,w1,x1,y1,z1}]];*)
  (*Print[numsols];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {BC}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[{b} → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  test2 = Chop[Table[((PRPDesignEqn) /. numsols2use[[i]]),
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test, test2}]
```

In[53]:=

```
SecondPRPSynthesis6[joint1_, joint2_, joint3_, Const_] :=
 Module[{PRPsubs, Dinput2useB, PRPrelD, W, x, y, z, G, u, v, w,
   a, c, b, R, PRPConstraintEqn, PRPDesignEqn, numsols, realnumsols,
   numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  PRPsubs = Thread[Flatten[{PRPin, ConstSym}] →
     Flatten[{joint1, joint2, joint3, Const}]];
  Dinput2useB = DmatPRP /. PRPsubs;
  PRPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  w = 0;
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelD[[i]].W, G - PRPrelD[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];
  numsols = NSolve[PRPDesignEqn == {0, 0, 0, 0, 0}, {u, v, x, y, z}];

  (*Print[Chop[PRPrelA]];
  Print[PRPConstraintEqn];
  Print[PRPDesignEqn];
  Print[numsols];*)
  (*Print[PRPDesignEqn/.Thread[{u,v,w,x,y,z}→ {u2,v2,w2,x2,y2,z2}]];*)
  (*The removes the imaginary solutions*)
  realnumsols =
   Flatten[Position[(u + v + w + x + y + z) /. numsols, val_ /; Head[val] == Real]];
  numsols2use = Table[numsols[[realnumsols[[i]]]], {i, Length[realnumsols]}];
  (*This is the length of {EF}*)
  leglengths =
   Table[Norm[({u, v, w} /. numsols2use[[i]]) - ({x, y, z} /. numsols2use[[i]])],
    {i, Length[numsols2use]}];
  (*This sets up the outputs*)
  uvwxyz2use = {u, v, w, x, y, z} /. numsols2use;
  legsubs = Table[Thread[b → leglengths[[i]]], {i, Length[leglengths]}];
  test = Chop[Table[((PRPConstraintEqn /. R → b) /. numsols2use[[i]]) /. legsubs[[i]],
     {i, Length[numsols2use]}]];
  {leglengths, uvwxyz2use, test}]
```

In[54]:=
```
DesignEquations[joint1_, joint2_, joint3_, Const_] :=
 Module[{PRPsubs, Dinput2useB, Ainput2useB, PRPrelD, PRPrelA, W, G, W2, G2, a, c, b,
   R, PRPConstraintEqn, PRPConstraintEqn2, PRPDesignEqn, PRPDesignEqn2, numsols,
   realnumsols, numsols2use, leglengths, uvwxyz2use, test, legsubs, subs},
  PRPsubs = Thread[Flatten[{PRPin, ConstSym}] →
     Flatten[{joint1, joint2, joint3, Const}]];

  Ainput2useB = AmatPRP /. PRPsubs;
  PRPrelA = Table[Ainput2useB[[i]].Inverse[Ainput2useB[[1]]], {i, 6}];
  W = {x, y, z, 1};
  G = {u, v, w, 1};
  PRPConstraintEqn =
   Table[Chop[Dot[G - PRPrelA[[i]].W, G - PRPrelA[[i]].W] - R^2], {i, 6}];
  PRPDesignEqn = Table[Chop[Expand[PRPConstraintEqn[[i + 1]] - PRPConstraintEqn[[1]]]],
    {i, 5}];

  Dinput2useB = DmatPRP /. PRPsubs;
  PRPrelD = Table[Dinput2useB[[i]].Inverse[Dinput2useB[[1]]], {i, 6}];
  W2 = {m, n, o, 1};
  G2 = {p, q, r, 1};
  PRPConstraintEqn2 =
   Table[Chop[Dot[G2 - PRPrelD[[i]].W2, G2 - PRPrelD[[i]].W2] - R^2], {i, 6}];
  PRPDesignEqn2 = Table[Chop[Expand[
      PRPConstraintEqn2[[i + 1]] - PRPConstraintEqn2[[1]]]], {i, 5}];
  Print[{PRPDesignEqn, PRPDesignEqn2}];
  {NumberForm[PRPDesignEqn, 2], NumberForm[PRPDesignEqn2, 2]}
 ]
```

Analysis Modules

The analysis is performed in a similar way as the planar four bar with some additional rotations and translations implemented. The general procedures are essentially the same for the two analysis procedures.

1.) Initial check to skip the analysis if there are no solutions from the synthesis.

2.) Import the linkage information from the synthesis

3.) Calculate the displacement and link length values given the imported coordinates.

4.) Check each branch by subtracting the input phi values from the calculated phi values based off the input theta.

5.) The solutions that do not branch are saved.

6.) solutions that nearly do not branch saved. these are saved for potential future optimization, but are not used currently.

7.) The pertinent information is packaged together for the output.

In[55]:=
```
FirstPRPAnalysis6[acbin_, uvwxyzin_, joint1_, joint2_, Const_] := Module[
  {len, InputLength, PRPsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
```

```
 Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
finalsols = {};
Altfinalsols = {};
Lastsols = {finalsols, Altfinalsols};
If[Length[acbin] == 0, Print["No Solutions found!"];
 Goto[noSolutions]];
InputLength = Length[joint1];
len = Length[uvwxyzin];
Constsub = Thread[ConstSym → Const];
uvwxyzsub = Table[Thread[{u1, v1, w1, x1, y1, z1} → uvwxyzin[[i]]], {i, len}];
NewABsub =
 Chop[Table[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub[[i]]),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub[[i]], VariableSym[[2]] →
            (joint2[[1]]), VariableSym[[1]] → (joint1[[1]])}])}]], {i, len}]];
legsubs = Table[b → acbin[[i]], {i, len}];
If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
      Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
    {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
 Goto[noSolutions]];
(*Print[NewABsub];*)
If[ToString[VariableSym[[2]]] == "γ",
 Branch1 =
  Quiet[N[Table[Mod[Round[Table[(Anal1eqn1 /. Flatten[{legsubs[[j]], NewABsub[[
              j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
         (joint2[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
 Branch2 = Quiet[N[Table[Mod[Round[Table[(Anal1eqn2 /. Flatten[{legsubs[[
              j]], NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
         (joint2[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];,
 Branch1 = Quiet[N[Table[Round[Table[(Anal1eqn1 /. Flatten[{legsubs[[j]],
            NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
        (joint2[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
 Branch2 = Quiet[N[Table[Round[Table[(Anal1eqn2 /. Flatten[{legsubs[[j]],
            NewABsub[[j]], Constsub, VariableSym[[1]] → (joint1[[i]])}]) -
        (joint2[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ joint2[[1]]}]*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
```

```
      Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
  AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
     {i, Length[acbin]}], _Integer];
  Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
     1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
     Length[ValidSols]}];
  Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
  coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

  finalsols = Table[Thread[{b, u1, v1, w1, x1, y1, z1, braval1} → Flatten[
       {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

  Altfinalsols = If[Length[AltSols] > 0, {joint1, joint2}, {}];
  Lastsols = {finalsols, Altfinalsols};
  Label[noSolutions];
  Lastsols]
```

In[57]:=

```
SecondPRPAnalysis[acbin_, uvwxyzin_, joint1_, joint2_, joint3_, Const_] := Module[
  {len, InputLength, PRPsubs, Constsub, legsubs, uvwxyzsub, Branch1, Branch2, Branch1adj,
   Branch2adj, Branch1vals, Branch2vals, ValidSols, AltSols, Sols2use, coord2use,
   Branch2use, finalsols, Test2, Altfinalsols, Lastsols, Alen, Blen, Clen, NewABsub},
  finalsols = {};
  Altfinalsols = {};
  Lastsols = {finalsols, Altfinalsols};
  If[Length[acbin] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  InputLength = Length[joint1];
  (*Print[InputLength];*)
  len = Length[uvwxyzin];
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Table[Thread[{u2, v2, w2, x2, y2, z2} → uvwxyzin[[i]]], {i, len}];
  (*Print[uvwxyzsub];*)
  NewABsub =
   Chop[Table[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub[[i]]),
         (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub[[i]], VariableSym[[3]] →
              joint3[[1]], VariableSym[[2]] → (joint2[[1]]),
            VariableSym[[1]] → (joint1[[1]])}])}]], {i, len}]];
  legsubs = Table[b → acbin[[i]], {i, len}];
  If[Total[Table[If[Or[Round[a /. legsubs[[i]], 10^-6] == 0,
       Round[c /. legsubs[[i]], 10^-6] == 0, Round[b /. legsubs[[i]], 10^-6] == 0], 0],
     {i, len}]] == 0, Print["Only Degenerate Solutions Found!"];
   Goto[noSolutions]];
  (*Print[ToString[VariableSym[[3]]]=="ϕ"];*)
  (*Print[NewABsub];*)
  If[ToString[VariableSym[[3]]] == "ϕ",
   Branch1 = Quiet[N[Table[
       Mod[Round[Table[(Anal2eqn1 /. Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
              VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
            (joint3[[i]]), {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];
   Branch2 = Quiet[N[Table[Mod[Round[Table[(Anal2eqn2 /.
```

```
            Flatten[{legsubs[[j]], NewABsub[[j]], Constsub, VariableSym[[1]] →
                (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) - (joint3[[i]]),
        {i, InputLength}], π * 10^-4], 2 * Pi], {j, len}]]];,
 Branch1 = Quiet[ N[Table[Round[Table[ (Anal2eqn1 /.
          Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
            VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
        (joint3[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
 Branch2 = Quiet[N[Table[Round[Table[ (Anal2eqn2 /.
          Flatten[{legsubs[[j]], NewABsub[[j]], Constsub,
            VariableSym[[1]] → (joint1[[i]]), VariableSym[[2]] → joint2[[i]]}]) -
        (joint3[[i]]), {i, InputLength}], π * 10^-4], {j, len}]]];
];
(*Print[NewABsub];*)
(*Print[legsubs];*)
(*Print[legsubs,uvwxyzsub];*)
(*Print[{((projptG1/.uvwxyzsub[[1]])/.Constsub),
  ((projptS/.uvwxyzsub[[1]])/.Constsub)/.γ→ joint2[[1]]}]*)
(*Print[Branch1];*)
Branch1adj = N[Branch1];
Branch2adj = N[Branch2];
Branch1vals =
 Table[Table[If[Or[Branch1[[i, j]] === Indeterminate, Abs[Branch1[[i, j]]] > 0.0001],
    0, 1], {j, Length[Branch1[[1]]]}], {i, Length[Branch1]}];
Branch2vals = Table[Table[If[Or[Branch2[[i, j]] === Indeterminate,
     Abs[Branch2[[i, j]]] > 0.0001], 0, 1],
   {j, Length[Branch2[[1]]]}], {i, Length[Branch2]}];
(*Print[Branch1,Branch2];
Print[Branch1vals,Branch2vals];*)
ValidSols = Cases[Table[If[Or[Total[Branch1vals[[i]]] == InputLength,
     Total[Branch2vals[[i]]] == InputLength], i], {i, Length[acbin]}], _Integer];
AltSols = Cases[Table[If[Or[Max[Branch1[[i]]] < .1, Max[Branch2[[i]]] < .1], i],
   {i, Length[acbin]}], _Integer];
Branch2use = Table[If[Total[Branch1vals[[ValidSols[[i]]]]] == InputLength,
   1, If[Total[Branch2vals[[ValidSols[[i]]]]] == InputLength, 2, 0]], {i,
   Length[ValidSols]}];

Sols2use = Table[acbin[[ValidSols[[i]]]], {i, Length[ValidSols]}];
coord2use = Table[uvwxyzin[[ValidSols[[i]]]], {i, Length[ValidSols]}];

finalsols = Table[Thread[{b, u2, v2, w2, x2, y2, z2, braval2} → Flatten[
     {Sols2use[[i]], coord2use[[i]], Branch2use[[i]]}]], {i, Length[ValidSols]}];

Altfinalsols = If[Length[AltSols] > 0, {joint1, joint2 , joint3}, {}];
Lastsols = {finalsols, Altfinalsols};
Label[noSolutions];
Lastsols]
```

Continuity Test

These check that the lengths of the links do not change through out its movements. The interval can be selected by setting the limit number. *** NOTE: The values all go from 0 to a ThetaMax... a Theta Min Value has not been implemented. ******

1.) Import the linkage information that passed through the analysis

2.) calculate the coordinates of the spherical joints in the correct reference. The moving joints should be in the frame of the revolute joint in which they rotate about.

*** In the second test tables of the γ and ϕ angles were calculated *****

3.) Calculate the initial link lengths

4.) Calculate the link lengths at the specified intervals

5.) Subtract the original link lengths from the subsequent link lengths

In[58]:=
```
ContTest16[inputs_, Const_, joint1_, joint2_(*,ThetaMin_,ThetaMax_*)] :=
 Module[{len, Constsub, legsubs, uvwxyzsub, PRPsubs, Joint1Table,
   NewABsub, limit, OAlinklen, ABlinklen, OAlen, ABlen, BClen, TestedSols,
   TableOfO, TableOfD, TableOfE, TableOfS, ThetaMin, ThetaMax},
  If[Length[inputs] == 0, Print["No Solutions found!"];
   Goto[noSolutions]];
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = inputs[[2 ;; 7]];
  legsubs = {b → (b /. inputs)};
  PRPsubs = Constsub;
  limit = 200;
  ThetaMin = Min[joint1];
  ThetaMax = Max[joint1];
  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  NewABsub = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
  Print[NewABsub];
  OAlinklen = OAlink /. Flatten[{Constsub, NewABsub,
     (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
  ABlinklen = ABlink /. Flatten[{Constsub, NewABsub,
     (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}] ;

  OAlen = N[Table[Norm[(((Ocoord - Acoord) /. Flatten[{legsubs, NewABsub, PRPsubs}]) /.
       {VariableSym[[1]] → Joint1Table[[n]]})], {n, limit}]];
  ABlen = N[Table[Norm[((((Bcoord - Acoord) /. {VariableSym[[2]] → ((If[
                  (braval1 /. inputs) == 1, Anal1eqn1, Anal1eqn2]) /. Flatten[{legsubs,
                 PRPsubs, NewABsub, VariableSym[[1]] → Joint1Table[[n]]}])}) /.
         Flatten[{legsubs, PRPsubs, NewABsub}]) /. VariableSym[[1]] →
        Joint1Table[[n]])], {n, limit}]];
  BClen = N[Table[Norm[(((((Ccoord - Bcoord) /. {VariableSym[[2]] → ((If[(braval1 /.
                      inputs) == 1, Anal1eqn1, Anal1eqn2]) /. Flatten[{legsubs,
                  PRPsubs, NewABsub, VariableSym[[1]] → Joint1Table[[n]]}])}) /.
          Flatten[{legsubs, PRPsubs, NewABsub}])) /. VariableSym[[
         1]] → Joint1Table[[n]])], {n, limit}]];
  (*Print[({OAlinklen,ABlinklen,b}/.legsubs)];
  Print[{OAlen,ABlen,BClen}];*)
  TestedSols = {Chop[{OAlen, ABlen, BClen} - ({OAlinklen, ABlinklen, b} /. legsubs)]};
  Label[noSolutions];
  TestedSols]
```

In[59]:=
```
ContTest26[inputs1_, inputs2_, Const_,
   joint1_, joint2_, joint3_(*,ThetaMin_,ThetaMax_*)] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    Joint1Table, Joint2Table, Joint3Table, FClinklen, DElinklen, limit, FClen, DElen,
    EFlen, TestedSols, TableOfO, TableOfA, TableOfB, TableOfC, ThetaMin, ThetaMax},
```

```
Constsub = Thread[ConstSym → Const];
uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
legsubs1 = {b → (b /. inputs1)};
legsubs2 = {b → (b /. inputs2)};
limit = 200;
ThetaMin = Min[joint1];
ThetaMax = Max[joint1];

NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
      (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
          (VariableSym[[1]] → joint1[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
          (VariableSym[[1]] → joint1[[1]])}])}]]];

Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];
Joint2Table =
 Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
      Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];

FClinklen = FClink /. Flatten[{Constsub, NewABsub2, NewABsub1,
    (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
    (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
    (VariableSym[[3]] → joint3[[1]])}];

FClen = N[Table[Norm[
    ((Fcoord - Ccoord) /. Flatten[{Constsub, NewABsub2, NewABsub1, (VariableSym[[
           2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]])}]) /.
     {VariableSym[[1]] → Joint1Table[[n]]}], {n, limit}]];
(*Print[Alen];*)

DElen =
 N[Table[Norm[(Ecoord - Dcoord) /. Flatten[{Constsub, NewABsub2, (VariableSym[[
          2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]]),
       (VariableSym[[1]] → Joint1Table[[n]])}]], {n, limit}]];

EFlen =
 N[Table[Norm[(Fcoord - Ecoord) /. Flatten[{Constsub, NewABsub2, (VariableSym[[
          2]] → Joint2Table[[n]]), (VariableSym[[3]] → Joint3Table[[n]]),
       (VariableSym[[1]] → Joint1Table[[n]])}]], {n, limit}]];
```

```
    (*Print[{Alen,Blen,Clen}];*)
    TestedSols = {(*inputs2[[1;;3]],*)
      Chop[{FClen, DElen, EFlen} - ({FClinklen, DElinklen, b /. legsubs2})]};
    (*Print[TestedSols];*)
    TestedSols];
```

Plotting Solutions

In[60]:=

```
PlottingSolutions6[inputs1_, inputs2_, Const_, joint1_,
  joint2_, joint3_, Seqn_, phieqn_(*,ThetaMin_,ThetaMax_*)] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, NewABsub1, NewABsub2, Joint1Table,
   Joint2Table, Joint3Table, limit, plotJoint2eqn1calc, plotJoint2eqn2calc,
   plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1, plotJoint2eqn2,
   plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3, plotJoint2,
   Joint2Points2Plot, Joint3Points2Plot, Joint2plot, Joint3plot, ThetaMin, ThetaMax},
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  limit = 200;
  ThetaMin = Min[joint1];
  ThetaMax = Max[joint1];
  NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
        (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
            (VariableSym[[1]] → joint1[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
      {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
            (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
            (VariableSym[[1]] → joint1[[1]])}])}]]];

  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

  Joint2Table =
   Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
       Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
  plotJoint2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
  plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];

  plotJoint1 = Joint1Table;

  plotJoint3eqn1calc =
   Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
       Joint1Table[[n]], VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];
  plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[{legsubs2, Constsub,
       NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
       VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];
```

```
plotJoint3 = N[(Seqn /. x → #) & /@ plotJoint1];
plotJoint2 = N[(phieqn /. x → #) & /@ plotJoint1];

Joint3Points2Plot = Table[{joint1[[i]], joint3[[i]]}, {i, Length[joint1]}];
Joint2Points2Plot = Table[
  {joint1[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, joint2][[i]]},
  {i, Length[joint1]}];

plotJoint3eqn1 = plotJoint3eqn1calc;
plotJoint3eqn2 = plotJoint3eqn2calc;
plotJoint2eqn1 =
 Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotJoint2eqn1calc];
plotJoint2eqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &,
  plotJoint2eqn2calc];

(*Print[plotJoint3eqn2calc];*)

Joint2plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn2}], Joint2Points2Plot},
  PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
  Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "γ Plot Comparison"];
Joint3plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint3}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn2}], Joint3Points2Plot},
  PlotLegends → {"Input Function", "Branch 1", "Branch 2", "Precision Points"},
  Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "d3 Plot Comparison"];
{{inputs1[[1]], inputs2[[1]]} // MatrixForm, Joint2plot, Joint3plot}
]
```

In[62]:=
```
FirstPosition3DPlot[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
  Module[{Constsub, uvwxyzsub, InitialPos, ptO,
    ptA, ptB, ptC, ptD, ptE, ptF, TubeDia, SphereDia},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub =  Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   InitialPos = Thread[VariableSym → {joint1[[1]], joint2[[1]], joint3[[1]]}];
   ptO = Ocoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptA = Acoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptB = Bcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptC = Ccoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptD = Dcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptE = Ecoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];
   ptF = Fcoord.rem /. Flatten[{Constsub, uvwxyzsub, InitialPos}];

   Print[uvwxyzsub];

   TubeDia = 0.125;
   SphereDia = .25;
   Graphics3D[{Gray, Tube[{ptO, ptC, ptF}, TubeDia], Green,
     Tube[{ptO, ptA}, TubeDia], Blue, Tube[{ptB, ptA, ptD}, TubeDia],
     Yellow, Tube[{ptB, ptC}, TubeDia], Sphere[{ptB, ptC}, SphereDia],
     Red, Tube[{ptD, ptE}, TubeDia], Purple, Tube[{ptE, ptF}, TubeDia],
     Sphere[{ptE, ptF}, SphereDia], Orange, Tube[{{0, 0, 0}, ptO}, TubeDia / 2]}]
  ];
```

In[247]:=
```
Plot4Paper[inputs1_, inputs2_, Const_, joint1_, joint2_,
  joint3_, Joint2eqn_, Joint3eqn_(*,ThetaMin_,ThetaMax_*)] :=
 Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, NewABsub1, NewABsub2, Joint1Table,
   Joint2Table, Joint3Table, limit, plotJoint2eqn1calc, plotJoint2eqn2calc,
   plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1, plotJoint2eqn2,
   plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3, plotJoint2,
   Joint2Points2Plot, Joint3Points2Plot, Joint2plot, Joint3plot, ThetaMin,
   ThetaMax, Joint24export, Joint34export, OutputPlot, ComparisonPlot},
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  legsubs1 = {b → (b /. inputs1)};
  legsubs2 = {b → (b /. inputs2)};
  limit = 200;
  ThetaMin = Min[joint1];
  ThetaMax = Max[joint1];
  NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
        (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
            (VariableSym[[1]] → joint1[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
      {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
            (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
            (VariableSym[[1]] → joint1[[1]])}])}]]];

  Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];
```

```
Joint2Table =
 Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
       Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
plotJoint2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];
plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
       NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}])], {n, limit}];

plotJoint1 = Joint1Table;

plotJoint3eqn1calc =
 Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
        Joint1Table[[n]], VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];
plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[{legsubs2, Constsub,
       NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
       VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];

Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
    Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      VariableSym[[2]] → Joint2Table[[n]]}])], {n, limit}];


plotJoint3 = N[(Joint3eqn /. x → #) & /@ plotJoint1];
plotJoint2 = N[(Joint2eqn /. x → #) & /@ plotJoint1];

Joint3Points2Plot = Table[{joint1[[i]], joint3[[i]]}, {i, Length[joint1]}];
Joint2Points2Plot = Table[
  {joint1[[i]], Map[If[# < 0 - .2, # + 2 * Pi, If[# > 2 * Pi, # - 2 * Pi, #]] &, joint2][[i]]
   (*Map[If[#<-Pi,#+2*Pi,If[#>Pi,#-2*Pi,#]]&,joint2][[i]]*)}, {i,
   Length[joint1]}];

plotJoint3eqn1 = plotJoint3eqn1calc;
plotJoint3eqn2 = plotJoint3eqn2calc;
plotJoint2eqn1 = plotJoint2eqn1calc
(*Map[If[#<-Pi,#+2*Pi,If[#>Pi,#-2*Pi,#]]&,plotJoint2eqn1calc];
plotJoint2eqn2 = plotJoint2eqn2calc(*+2*Pi*)
(*Map[If[#<-Pi,#+2*Pi,If[#>Pi,#-2*Pi,#]]&,plotJoint2eqn2calc];

Joint2plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint2eqn2}], Joint2Points2Plot},
  PlotLegends → {"Desired Rotation Angle", "Branch 1", "Branch 2",
    "Precision Points"}, Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "S Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
Joint3plot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint3}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn1}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3eqn2}], Joint3Points2Plot},
```

```
  PlotLegends → {"Desired Slide", "Branch 1", "Branch 2", "Precision Points"},
  Joined → {True, True, True, False},
  PlotStyle → {Thick, Dashing[Medium], Dashing[Tiny], PointSize[Large]},
  PlotLabel → "ϕ Plot Comparison",
  LabelStyle → Directive[FontFamily → "Times New Roman", 15]];
(*{{inputs1[[1]],inputs2[[1]]}//MatrixForm,γplot,Joint3plot}*)

Joint24export = Show[Joint2plot, ImageSize → Large, PlotLabel → "",
  FrameLabel → {Style["Input Slide(in)", 18], Style["Crank Angle (Radians)", 18]},
  Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];
Joint34export = Show[Joint3plot, ImageSize → Large, PlotLabel → "",
  FrameLabel → {Style["Input Slide(in)", 18], Style["Output Slide(in)", 18]},
  Frame → {True, True, False, False},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ,
  TicksStyle → Directive[FontSize → 20]];

OutputPlot =
 ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}], MapThread[{#1, #2} &,
    {plotJoint1, plotJoint3}], Joint2Points2Plot, Joint3Points2Plot}, PlotLegends →
   {"Desired Rotation Angle", "Desired Slide Distance", "Rotation Precision Points",
    "Slide Precision Points"}, Joined → {True, True, False, False},
  PlotStyle → {Thick, Dashing[Medium], PointSize[.015], PointSize[.02]},
  ImageSize → Large, FrameLabel → {{Style["Rotation Angle (Radians)", 18],
     Style["Slide Distance (in)", 18]}, {Style["Input Slide (in)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

ComparisonPlot = ListPlot[{MapThread[{#1, #2} &, {plotJoint1, plotJoint2}],
   MapThread[{#1, #2} &, {plotJoint1, plotJoint3}], Joint2Points2Plot,
   Joint3Points2Plot, MapThread[{#1, #2} &, {plotJoint1, Joint2Table}],
   MapThread[{#1, #2} &, {plotJoint1, Joint3Table}]}, PlotLegends →
   {"Desired Rotation Angle", "Desired Slide Distance", "Rotation Precision Points",
    "Slide Precision Points", "Rotation Angle", "Slide Distance"},
  Joined → {True, True, False, False, True, True}, PlotStyle → {{Thick}, {Medium},
    {PointSize[.015]}, {PointSize[.02]}, {Dashing[Medium]}, {Dashing[Tiny]}},
  ImageSize → Large, FrameLabel → {{Style["Rotation Angle (Radians)", 18],
     Style["Slide Distance (in)", 18]}, {Style["Input Slide (in)", 18], None}},
  FrameTicks → {{All, All}, {All, None}}, Frame → {True, True, False, True},
  LabelStyle → Directive[FontFamily → "Times New Roman", 15] ];

(*DesiredRotateRetractPlot=ListPlot[{MapThread[{#1,#2}&,{plotJoint1,plotJoint2}],
   MapThread[{#1,#2}&,{plotJoint1,plotJoint3}]},PlotLegends→
   {"Desired Slide Distance", "Desired Rotation Angle"}, Joined→ {True,True},
  PlotStyle→ {Thick,Dashing[Medium]},ImageSize→ Large,FrameLabel→
   {{Style["Slide Distance (in)",18],Style["Rotation Angle (Radians)",18]},
    {Style["Crank Angle (Radians)",18],None}},
  FrameTicks→ {{All,All},{All,None}},Frame→ {True,True,False,True},
  LabelStyle→ Directive[FontFamily→ "Times New Roman",15] ]*)
```

```
  {{inputs1[[1]], inputs2[[1]]} // MatrixForm,
   Joint24export, Joint34export, OutputPlot, ComparisonPlot}
 ]
```

Ranking Solutions

The LinkRatio module calculates all of the link lengths and then calculates the ratio between the longest and shortest link.

In[65]:=
```
LinkRatio[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
  Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
    EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen,
    OClinklen, OFlinklen, BDlinklen, FClinklen, LinkLengths},
   Constsub = Thread[ConstSym → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
         (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
             (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
        {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
             (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
             (VariableSym[[1]] → joint1[[1]])}])}]]];

   BClinklen = b /. inputs1;
   EFlinklen = b /. inputs2;

   OAlinklen = OAlink /. Flatten[{Constsub,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
   ADlinklen = ADlink /. Flatten[{Constsub, (VariableSym[[2]] → joint2[[1]]),
       (VariableSym[[1]] → joint1[[1]]), (VariableSym[[3]] → joint3[[1]])}];
   ABlinklen = ABlink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}];
   DElinklen = DElink /. Flatten[{Constsub, NewABsub2,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
       (VariableSym[[3]] → joint3[[1]])}];
   OClinklen = OClink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[1]] → joint1[[1]])}];
   OFlinklen = OFlink /. Flatten[{Constsub, NewABsub2,
       (VariableSym[[1]] → joint1[[1]])}];
   BDlinklen = BDlink /. Flatten[{Constsub, NewABsub1,
       (VariableSym[[2]] → joint2[[1]]), (VariableSym[[1]] → joint1[[1]]),
       (VariableSym[[3]] → joint3[[1]])}];
   FClinklen = FClink /. Flatten[{NewABsub1, NewABsub2}];

   LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
     ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
   (*Print[LinkLengths];*)
   Max[LinkLengths] / Min[LinkLengths]
  ];
```

The CurveComparison module find the root mean square difference between the γ and ϕ equations given in the input and the

In[66]:=
```
CurveComparison[inputs1_, inputs2_, Const_,
   joint1_, joint2_, joint3_, seqn_, phieqn_, ThetaMin_, ThetaMax_] :=
  Module[{Constsub, legsubs1, legsubs2, uvwxyzsub, PRPsubs, Joint1Table,
    NewABsub1, NewABsub2, Joint2Table, Joint3Table, limit, plotJoint2eqn1calc,
```

```
 plotJoint2eqn2calc, plotJoint3eqn1calc, plotJoint3eqn2calc, plotJoint2eqn1,
 plotJoint2eqn2, plotJoint3eqn1, plotJoint3eqn2, plotJoint1, plotJoint3, plotJoint2,
 Joint2Points2Plot, Joint3Points2Plot, splot, Joint3plot, SError, PhiError},
Constsub = Thread[ConstSym → Const];
uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
legsubs1 = {b → (b /. inputs1)};
legsubs2 = {b → (b /. inputs2)};
PRPsubs = Constsub;
limit = 200;

Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
      (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
          (VariableSym[[1]] → joint1[[1]])}])}]]];
NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
          (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
          (VariableSym[[1]] → joint1[[1]])}])}]]];

Joint1Table = Table[ThetaMin + (ThetaMax - ThetaMin) * (n / limit), {n, 0, limit - 1}];

Joint2Table =
 Table[(If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1,
      Constsub, NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
plotJoint2eqn1calc = Table[(Anal1eqn1 /. Flatten[{legsubs1, Constsub,
      NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];
plotJoint2eqn2calc = Table[(Anal1eqn2 /. Flatten[{legsubs1, Constsub,
      NewABsub1, VariableSym[[1]] → Joint1Table[[n]]}]), {n, limit}];

plotJoint1 = Joint1Table;

plotJoint3eqn1calc =
 Table[(Anal2eqn1 /. Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
       Joint1Table[[n]], VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];
plotJoint3eqn2calc = Table[(Anal2eqn2 /. Flatten[{legsubs2, Constsub,
      NewABsub2, VariableSym[[1]] → Joint1Table[[n]],
      VariableSym[[2]] → Joint2Table[[n]]}]), {n, limit}];

plotJoint3 = N[(seqn /. x → #) & /@ plotJoint1];
plotJoint2 = N[(phieqn /. x → #) & /@ plotJoint1];

Joint2Points2Plot = Table[{joint1[[i]], joint2[[i]]}, {i, Length[joint1]}];
Joint3Points2Plot = Table[
  {joint1[[i]], Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, joint3][[i]]},
  {i, Length[joint1]}];

plotJoint3eqn1 = plotJoint3eqn1calc;
plotJoint3eqn2 = plotJoint3eqn2calc;
```

```
  plotJoint2eqn1 =
   Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, plotJoint2eqn1calc];
  plotJoint2eqn2 = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &,
    plotJoint2eqn2calc];

  SError =
   Sqrt[Total[MapThread[(#1 - #2)^2 &, {If[(braval1 /. inputs1) == 1, plotJoint2eqn1,
         plotJoint2eqn2], plotJoint2}]] / limit];
  PhiError = Sqrt[Total[MapThread[(#1 - #2)^2 &, {If[(braval2 /. inputs2) == 1,
         plotJoint3eqn1, plotJoint3eqn2], plotJoint3}]] / limit];
  (*Print[{GammaError,PhiError}];*)
  SError + PhiError
 ];
```

In[67]:=

```
AllLinkLengths[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_] :=
 Module[{Constsub, uvwxyzsub, NewABsub1, NewABsub2, BClinklen,
   EFlinklen, OAlinklen, ADlinklen, ABlinklen, DElinklen, OClinklen,
   OFlinklen, BDlinklen, FClinklen, LinkLengths, Sublist},
  Constsub = Thread[ConstSym → Const];
  uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
  NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} → Flatten[{(NewUVW1 /. uvwxyzsub),
       (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[2]] → joint2[[1]]),
           (VariableSym[[1]] → joint1[[1]])}])}]]];
  NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[
     {(NewUVW2 /. uvwxyzsub), (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub,
           (VariableSym[[3]] → joint3[[1]]), (VariableSym[[2]] → (joint2[[1]])),
           (VariableSym[[1]] → joint1[[1]])}])}]]];

  BClinklen = b /. inputs1;
  EFlinklen = b /. inputs2;

  Sublist =
   Flatten[{Constsub, NewABsub1, NewABsub2, (VariableSym[[2]] → joint2[[1]]),
     (VariableSym[[1]] → joint1[[1]]), (VariableSym[[3]] → joint3[[1]])}];
  OAlinklen = OAlink /. Sublist;
  ADlinklen = ADlink /. Sublist;
  ABlinklen = ABlink /. Sublist;
  DElinklen = DElink /. Sublist;
  OClinklen = OClink /. Sublist;
  OFlinklen = OFlink /. Sublist;
  BDlinklen = BDlink /. Sublist;
  FClinklen = FClink /. Sublist;

  LinkLengths = {BClinklen, EFlinklen, OAlinklen, ADlinklen,
    ABlinklen, DElinklen, OClinklen, OFlinklen, BDlinklen, FClinklen};
  (*Print[LinkLengths];*)
  LinkLengths
 ];
```

Finding Solutions

In[68]:=
```
SolutionPath[inputs1_, inputs2_, Const_, joint1_, joint2_, joint3_, ThetaMax_] :=
  Module[{len, Constsub, legsubs1, , legsubs2, uvwxyzsub, NewABsub1, NewABsub2,
    Joint1Table, Joint2Table, Joint3Table, OutputTable, limit, GammaOut, PhiOut},
   Constsub = Thread[{d1, d2, d3, a1, a2, α1, α2} → Const];
   uvwxyzsub = Flatten[{inputs1[[2 ;; 7]], inputs2[[2 ;; 7]]}];
   legsubs1 = {b → (b /. inputs1)};
   legsubs2 = {b → (b /. inputs2)};
   limit = 200;

   NewABsub1 = Chop[Thread[{u1, v1, w1, x1, y1, z1} →
       Flatten[{(NewUVW1 /. uvwxyzsub), (NewXYZ1 /. Flatten[{Constsub, uvwxyzsub,
             (γ → joint2[[1]]), (VariableSym[[1]] → joint1[[1]])}])}]]];
   NewABsub2 = Chop[Thread[{u2, v2, w2, x2, y2, z2} → Flatten[{(NewUVW2 /. uvwxyzsub),
        (NewXYZ2 /. Flatten[{Constsub, uvwxyzsub, (VariableSym[[3]] → joint3[[1]]),
            (γ → (joint2[[1]])), (VariableSym[[1]] → joint1[[1]])}])}]]];

   Joint1Table = Table[ThetaMax * (n / limit), {n, limit}];
   Joint2Table = Table[
     (If[(braval1 /. inputs1) == 1, Anal1eqn1, Anal1eqn2] /. Flatten[{legsubs1, Constsub,
        NewABsub1, VariableSym[[1]] → (ThetaMax * (n / limit))}]), {n, limit}];
   Joint3Table = Table[(If[(braval2 /. inputs2) == 1, Anal2eqn1, Anal2eqn2] /.
       Flatten[{legsubs2, Constsub, NewABsub2, VariableSym[[1]] →
          (ThetaMax * (n / limit)), γ → Joint2Table[[n]]}]), {n, limit}];


   GammaOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, Joint2Table];
   PhiOut = Map[If[# < -Pi, # + 2 * Pi, If[# > Pi, # - 2 * Pi, #]] &, Joint3Table];

   OutputTable = {Joint1Table, GammaOut, PhiOut};
   OutputTable];
```

Inputs

Input Functions/Values

Polynomial Fitting

```
d1pts = {3.93305294, 3.5, 3, 2.5, 2, 1.5}
```

```
{3.93305, 3.5, 3, 2.5, 2, 1.5}
```

```
γpts = N[{162.92215157, 185, 190, 210, 230, 250} * Degree]
```

```
{2.84353, 3.22886, 3.31613, 3.66519, 4.01426, 4.36332}
```

```
d3pts = {3.76447160, 3.8, 3.82, 4.2, 4.3, 6}
```

```
{3.76447, 3.8, 3.82, 4.2, 4.3, 6}
```

```
d1γpts = MapThread[{#1, #2} &, {d1pts, γpts}]
```

```
{{3.93305, 2.84353}, {3.5, 3.22886},
 {3, 3.31613}, {2.5, 3.66519}, {2, 4.01426}, {1.5, 4.36332}}
```

```
d1d3pts = MapThread[{#1, #2} &, {d1pts, d3pts}]
```

```
{{3.93305, 3.76447}, {3.5, 3.8}, {3, 3.82}, {2.5, 4.2}, {2, 4.3}, {1.5, 6}}
```

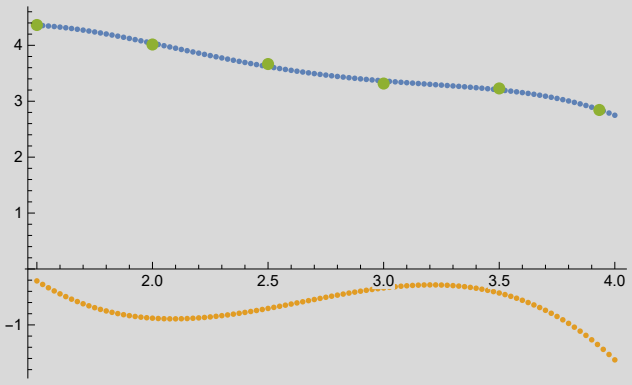
```
d1γFunction = Fit[d1γpts, {1, x, x^2, x^3, x^4}, x]
```

$$-2.69864 + 13.3161\,x - 8.68921\,x^2 + 2.2899\,x^3 - 0.216182\,x^4$$

```
Sfitplot = Table[({x, d1γFunction} /. (x → 1.5 + 2.5 * n / 100)), {n, 0, 100}];
```

```
dSfitplot = Table[({x, D[d1γFunction, x]} /. (x → 1.5 + 2.5 * n / 100)), {n, 0, 100}];
```

```
ListPlot[{Sfitplot, dSfitplot, d1γpts}, PlotStyle → {Thick, Thick, PointSize[Large]}]
```

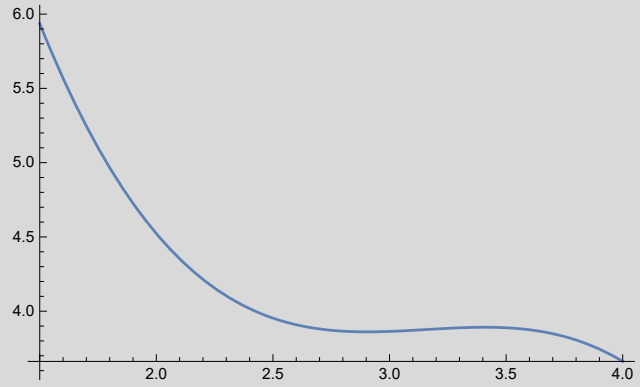


Piece Wise function for Phi Angle

```
d1d3Function = Fit[d1d3pts, {1, x, x^2, x^3}, x]
 (*Piecewise[{{-15/20*x+.65,x<.87},{0,x≥.87}}];*)
```

$$18.8954 - 14.4602\,x + 4.6107\,x^2 - 0.486936\,x^3$$

```
Plot[d1d3Function, {x, 1.5, 4}]
```



Precision Points

These are the functions that are needed to drive the slide and rotation

```
precisionptsd1 = d1pts
```

```
{3.93305, 3.5, 3, 2.5, 2, 1.5}
```

```
precisionptsd3 = Map[(d1d3Function /. x → #) &, precisionptsd1]
```

```
{3.72016, 3.88857, 3.86397, 3.95352, 4.52241, 5.93584}
```

```
precisionptsγ = Map[(d1γFunction /. x → #) &, precisionptsd1]
```

```
{2.84959, 3.20345, 3.36329, 3.61912, 4.03702, 4.3588}
```

Tolerances

These are the tolerances that are used for the randomization process

```
d1Tols = {.1, .1, .1, .1, .1, .1, .1}
```

```
{0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
```

```
d3Tols = {.5, .5, .5, .5, .5, .5, .5}
```

```
{0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5}
```

```
GammaTols = N[{15, 15, 15, 15, 15, 15, 15} * Degree]
```

```
{0.261799, 0.261799, 0.261799, 0.261799, 0.261799, 0.261799, 0.261799}
```

D-H Table

```
ConstSym
```

```
{θ, d2, ϕ, a1, a2, α1, α2}
```

```
θval = -44 * Degree;
d2val = 2.02525869;
ϕvals = 0;
a1vals = 0.3473733;
a2vals = 1.10513158;
α1val = -100.54529059 * Degree - 60 * Degree;
α2val = 104.83676791 * Degree;
```

```
Const = N[{θval, d2val, ϕvals, a1vals, a2vals, α1val, α2val}]
```

```
{-0.767945, 2.02526, 0., 0.347373, 1.10513, -2.80204, 1.82975}
```

```
DHTable = {{θ, d1, α1, a1}, {γ, d2, α2, a2}, {ϕ, d3, 0, 0}} // MatrixForm
```

$$\begin{pmatrix} \theta & d1 & \alpha1 & a1 \\ \gamma & d2 & \alpha2 & a2 \\ \phi & d3 & 0 & 0 \end{pmatrix}$$

```
DHTable /. Thread[Flatten[{ConstSym, θ, γ, ϕ}] → Flatten[{Const, θ1, θ2, θ3}]]
```

$$\begin{pmatrix} -0.767945 & d1 & -2.80204 & 0.347373 \\ \theta_2 & 2.02526 & 1.82975 & 1.10513 \\ 0. & d3 & 0 & 0 \end{pmatrix}$$

```
TeXForm[DHTable /. Thread[Flatten[{ConstSym, θ, γ, ϕ}] → Flatten[{Const, θ1, θ2, θ3}]]]
```

```
\left(
\begin{array}{cccc}
 -0.767945 & \text{d1} & -2.80204 & 0.347373 \\
 \theta _2 & 2.02526 & 1.82975 & 1.10513 \\
 0. & \text{d3} & 0 & 0 \\
\end{array}
\right)
```

Position of Points

```
ConstSubs = Thread[Flatten[{ConstSym, r1, γ1, s1}] →
   Flatten[{Const, precisionptsd1[[1]], precisionptsγ[[1]], precisionptsd3[[1]]}]]
```

```
{θ → -0.767945, d2 → 2.02526, ϕ → 0., a1 → 0.347373, a2 → 1.10513,
 α1 → -2.80204, α2 → 1.82975, r1 → 3.93305, γ1 → 2.84959, s1 → 3.72016}
```

O Position and Vector

```
Opos = (OmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{0., 0., 3.93305}
```

```
Ovec = (OmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{0., 0., 4.93305}
```

```
Oxpos = (OmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{0.71934, -0.694658, 3.93305}
```

A Position and Vector

```
Apos = (AmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{0.718451, 0.243915, 2.02343}
```

```
Avec = (AmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{0.949816, 0.483499, 1.08052}
```

```
Axpos = (AmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{-0.158993, 0.713912, 1.92755}
```

D Postition and Vector

```
Dpos = (DmatPRP[[1]] /. ConstSubs).{0, 0, 0, 1}.rem
```

```
{-1.9827, -2.51992, 1.66865}
```

```
Dvec = (DmatPRP[[1]] /. ConstSubs).{0, 0, 1, 1}.rem
```

```
{-2.44813, -3.40248, 1.60177}
```

```
Dxpos = (DmatPRP[[1]] /. ConstSubs).{1, 0, 0, 1}.rem
```

```
{-2.86015, -2.04992, 1.57278}
```

Solvers

Main Solver

This is the solver. The solver first randomizes the θ and γ values. Once a set of θ and γ values are found that pass analysis they are used repeatedly for sets of randomized ϕ values.

This solver utilizes multithreading. It will still work if multithreading is not initialized, but it will run single threaded.

```
(*{precisionptsd1,precisionptsγ,precisionptsd3}={testRandValsGood1[[1,1;;6]],
   testRandValsGood1[[2,1;;6]],testRandValsGood1[[3,1;;6]]};*)
Iterations = 10000;
SetSharedVariable[IterationCounter];
IterationCounter = 0;
k1 = 1;
SecondIterations = 100;
NumInputs = 6;
SolutionList = Array[0 &, Iterations];
Randvals = {precisionptsd1, precisionptsγ, precisionptsd3};
SolutionList = ParallelTable[
   (*Print[i];*)
   If[i > 1, Randvals = {RandomizeTps2[precisionptsd1, d1Tols, NumInputs],
       RandomizeTps1[precisionptsγ, GammaTols, NumInputs],
       RandomizeTps2[precisionptsd3, d3Tols, NumInputs]};,
    Randvals = {precisionptsd1, precisionptsγ, precisionptsd3};];
   (*First Four Bar Code *)
   test1 = Chop[FirstPRPSynthesis6[Randvals[[1]], Randvals[[2]], Const]];
   testanal1 =
    FirstPRPAnalysis6[test1[[1]], test1[[2]], Randvals[[1]], Randvals[[2]], Const];
   test1totals = If[Length[testanal1[[1]]] == 0, "No Solution", Table[Total[
       Flatten[ContTest16[testanal1[[1, i]], Const, Randvals[[1]], Randvals[[2]],
         Randvals[[1, 1]], Randvals[[1, 6]]]]], {i, Length[testanal1[[1]]]}]];
   (*Print[test1totals];*)
   test1sols = Table[If[test1totals[[i]] == 0, {testanal1[[1, i]],
       Randvals[[1]], Randvals[[2]]}, 0], {i, Length[test1totals]}];
   test1sols = DeleteCases[test1sols, 0];
   (*Print[test1sols];*)
   (*Second Four Bar Code *)
   SecondSolutionList = Array[0 &, SecondIterations];
   (*Creates array of zeros to store solutions from second four bar*)
   If[Length[test1sols] == 0, 0,
```

```
    IterationCounter++;
    Theta1sols = test1sols[[1, 2]]; (*Saves the randomized theta values*)
    S1sols = test1sols[[1, 3]]; (* Saves the randomized gamma values*)
    k2 = 1;
    Randd3 = precisionptsd3;
    For[n2 = 0, n2 < SecondIterations, n2++,
     (*Print["Inside ", n2];*)
     test2 = Chop[SecondPRPSynthesis6[Theta1sols, S1sols, Randd3, Const]];
     testanal2 =
      SecondPRPAnalysis[test2[[1]], test2[[2]], Theta1sols, S1sols, Randd3, Const];
     test2totals = If[Length[testanal2[[1]]] == 0, "No Solution",
       Table[Table[Total[Flatten[ContTest26[test1sols[[k, 1]], testanal2[[1, i]],
            Const, Theta1sols, S1sols, Randd3, Randvals[[1, 1]], Randvals[[1, 6]]]]],
         {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}]];
     test2sols = Table[Table[If[test2totals[[k, i]] == 0,
         {test1sols[[k, 1]], testanal2[[1, i]], Theta1sols, S1sols, Randd3}, 0],
        {i, Length[testanal2[[1]]]}], {k, Length[test1sols]}];
     test2sols = DeleteCases[Flatten[test2sols, 1], 0];
     If[test2sols == {}, , SecondSolutionList[[k2]] = test2sols; k2++;];
     Randd3 = RandomizeTps2[precisionptsd3, d3Tols, NumInputs];
    ](*End of second four bar FOR loop*)
   ]; (* End of Second four bar IF statement*)
   SecondSolutionList = DeleteCases[SecondSolutionList, 0];
   SecondSolutionList
   , {i, Iterations}];(*End of First Four bar FOR loop*)
Print["Stop 1"];
SolutionList = DeleteCases[SolutionList, (0 | {})];
Print["Stop 2"];
SolutionList = Flatten[SolutionList, 2];
Print["Stop 3"];
Print["Iterations: ",
  (Iterations - IterationCounter + IterationCounter * SecondIterations)];
```

```
SolutionList
```

```
Length[SolutionList]
```

Ranking Solutions

Calculates the link length ratios of each solution.

```
Length[SolutionList]
```

```
SolsRankLinkLen =
  Sort[ParallelTable[{LinkRatio[SolutionList[[i, 1]], SolutionList[[i, 2]],
      Const, SolutionList[[i, 3]], SolutionList[[i, 4]], SolutionList[[i, 5]]],
     SolutionList[[i]]}, {i, Length[SolutionList]}]];
```

```
MaxRatio = 3;
```

```
SolsRankLinkLen2use =
  DeleteCases[Table[If[SolsRankLinkLen[[i, 1]] < MaxRatio, SolsRankLinkLen[[i, 2]], 0],
    {i, Length[SolsRankLinkLen]}], 0];
```

```
(*SolsRankLinkLen*)
```

```
SolsRankLinkLen[[1]]
```

```
SolsRankLinkLen[[2]]
```

```
Append[SolsRankLinkLen2use[[1]], SolsRankLinkLen[[1, 1]]]
```

```
Length[SolsRankLinkLen2use]
```

Compares the RMS errors.

```
SortedSols = Sort[ParallelTable[
    {CurveComparison[SolsRankLinkLen2use[[i, 1]], SolsRankLinkLen2use[[i, 2]],
      Const, SolsRankLinkLen2use[[i, 3]], SolsRankLinkLen2use[[i, 4]],
      SolsRankLinkLen2use[[i, 5]], d1γFunction, d1d3Function,
      Min[SolsRankLinkLen2use[[i, 3]]], Max[SolsRankLinkLen2use[[i, 3]]]],
     SolsRankLinkLen2use[[i]]}, {i, Length[SolsRankLinkLen2use]}]];
```

```
SortedSols[[1]]
```

```
Length[SortedSols]
```

```
ListPlot[MapThread[{#1, #2} &, {SortedSols[[1, 2, 3]], SortedSols[[1, 2, 4]]}]]
```

```
ParallelTable[PlottingSolutions6[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
  Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]],
  d1d3Function, d1γFunction(*,1.5,4*)], {i, Min[Length[SortedSols], 200]}]
```

```
ParallelTable[FirstPosition3DPlot[SortedSols[[i, 2, 1]],
  SortedSols[[i, 2, 2]], Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]],
  SortedSols[[i, 2, 5]]], {i, Min[Length[SortedSols], 200]}]
```

```
ParallelTable[{AllLinkLengths[SortedSols[[i, 2, 1]], SortedSols[[i, 2, 2]],
   Const, SortedSols[[i, 2, 3]], SortedSols[[i, 2, 4]], SortedSols[[i, 2, 5]]],
  Append[SortedSols[[i, 2]], Const]}, {i, Length[SortedSols]}]
```