Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

8-21-2015 12:00 AM

A 3-DOF Stewart Platform for Trenchless Pipeline Rehabilitation

Derek K. Brecht
The University of Western Ontario

Supervisor
Dr. Ken McIsaac
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science
© Derek K. Brecht 2015

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Electrical and Computer Engineering Commons

Recommended Citation
Brecht, Derek K., "A 3-DOF Stewart Platform for Trenchless Pipeline Rehabilitation" (2015). *Electronic Thesis and Dissertation Repository*. 3156.
https://ir.lib.uwo.ca/etd/3156

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

A 3-DOF Stewart Platform for Trenchless Pipeline Rehabilitation

(Thesis Format: Monograph)

by:

Derek Brecht

Graduate Program in Engineering Science
Department of Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Derek Brecht 2015

Abstract

A major component of the infrastructure of any modern city is a network of underground pipes that transport drinking water, storm water and sewage. Most of the pipes currently being used are made out of concrete or various plastics. As with any material, they have an expected lifespan after which deterioration begins to occur. This can result in cracks, and in some cases, even large holes in the pipe which can cause a complete loss of function of the pipe. These defects invariably lead to water losses that necessitate the repair of the pipeline, which is an expensive undertaking.

The purpose of this thesis is to give a detailed report of the development and testing of a robot with a spray head that is autonomously controlled. This spray head will deposit a liquid material onto the pipe that will then cure to form the new interior wall of the pipe. The design of the robot most suited to this task is a Stewart platform: a parallel manipulator that uses prismatic actuators to control a single end-effector. In contrast to the traditional Stewart platform design, which has six independently controlled legs that are used to control the position of the top platform, a novel design is used which has only three independently controlled legs. The advantages of this design are less weight, less complicated kinematics and a smaller design envelope. A circular trajectory was implemented in the microcontroller code and the accuracy of the Stewart platform was evaluated using videos and image processing techniques.

An optimization algorithm is proposed which combines the controlled random search algorithm and the particle swarm optimization algorithm. The effectiveness of this algorithm is demonstrated by selecting the design parameters of a 3-DOF Stewart platform so that the radius of the circular spray path is maximized.

Keywords: 3-DOF, Stewart Platform, Parameter Optimization, Workspace maximization, Pipeline spraying, Controlled Random Search, Particle Swarm Optimization

Acknowledgments

There are many people that I wish to offer my gratitude, for their help and support during the course of this project. First, I would like to thank my supervisor, Dr. Ken McIsaac. I would also like to thank Clayton Cook and Dan Sweiger from University Machine Services for their help with the parts selection, design and construction of the Stewart platform.

My thanks are extended to Eugen Porter from the University electronics shop for sharing his knowledge of DC motor controllers, microcontrollers and actuator control. I would also like to acknowledge the help I received from Ron Struke, Trent Steensma and Ken Strong with electronic part selection and procurement, as well as for allowing me to use their soldering equipment.

Finally, I am grateful for the support of my family and friends, who were always ready to assist me in any way that they could.

Table of Contents

Contents Page

Abstract………………………………………………………………………. ii

Acknowledgements……………………………………………………………... iii

Table of Contents……………………………………………………………….. iv

List of Tables……………………………………………………………………. vii

List of Figures………………………………………………………………….... viii

List of Acronyms………………………………………………………….......... xi

Chapter 1 – Introduction……………………………………………………….. 1

1.1 Motivation of Study……………………………………………… 1

1.2 Design Considerations…………………………………………… 2

1.3 Research Objectives……………………………………………… 6

1.4 Contributions.. 10

Chapter 2 – Background and Kinematic Analysis…………………………... 11

2.1 Parallel Manipulators..…………………………………………… 11

2.2 Inverse Kinematics of the Stewart Platform……………………………….. 13

2.3 Workspace Calculation…………………………………………... 18

2.3.1 Actuator Stroke Length…........…………………………... 18

2.3.2 Range of the Passive Joints……………………………….. 18

2.4 Algorithm for Workspace Calculation…………………………… 25

Chapter 3 – Parameter Optimization………………………………………… 35

3.1 Delta Robot Optimization Introduction………………………….. 35

3.2 Design Problem Formulation…………………………………….. 38

3.2.1 The Objective Function………………………………… 38

3.2.2 The Workspace Constraints……………………………….. 39

3.2.3 The Manipulability Constraints………………………… 40

3.3 Optimization Algorithms………………………………………………………. 41

3.3.1 Controlled Random Search…………………………………….. 41

3.3.2 Particle Swarm Optimization…………………………………… 43

3.4 Design Problem………………………………………………………….... 44

3.4.1 Optimal Design of the Delta Robot…………………………….. 44

3.4.2 Computation……………………………………………………. 45

3.5 Delta Robot Optimization Results and Discussion………………………. 49

3.6 3-DOF Stewart Platform Optimization Introduction……………………... 52

3.7 Design Problem Formulation…………………………………………….. 52

3.7.1 The Objective Function………………………………………… 52

3.7.2 The Workspace Constraints……………………………………. 54

3.8 Design Problem………………………………………………………….... 55

3.8.1 Optimal Design of the Stewart Platform………………………... 55

3.8.2 Computation…………………….. 55

3.9 Stewart Platform Optimization Results and Discussion…………………. 56

Chapter 4 – Robot Design………………………………………………………. 59

4.1 Actuator Types…………………………………………………………… 59

4.2 Actuator Selection……………………………………………………….... 65

4.3 Electrical System Design………………………………………………… 69

4.4 Microprocessor Selection………………………………………………… 78

Chapter 5 – Circular Trajectory Implementation…………………………………….. 82

5.1 Description of Implementation Code…………………………….. 82

5.2 Experimental Set-up……………………………………………… 84

5.3 Experimental Results…………………………………………….... 86

Chapter 6 – Conclusions and Future Work…………………………………………... 88

6.1 Summary of Project and Conclusions…………………………………….. 88

6.1.1 Summary of Parameter Optimization………………………….... 88

6.1.2 Summary of Robot Design……………………………………… 89

6.1.3 Summary of Trajectory Implementation……………………….... 90

6.2 Recommendations for Future Work……………………………………… 91

References………………………………………………………………………. 94

Appendix A…………………………………………………………………….... 98

Appendix B…………………………………………………………………….... 106

Appendix C…………………………………………………………………….... 148

Curriculum Vitae……………………………………………………………….... 173

List of Tables

Table 1.2.1 – Serial Robot Characteristics……………………………………………...... 4

Table 2.4.1 – Summary of Constraint Violations……………………………………… 26

Table 3.5.1 – PSO only Algorithm Results…………………………………………….. 49

Table 3.5.2 – Combined PSO and CRS Algorithm Results……………………………. 50

Table 3.9.1 – PSO only Algorithm Results…………………………………………….. 56

Table 3.9.2 – Combined PSO and CRS Algorithm Results……………………………. 57

Table 4.2.1 – Advantages and Disadvantages for each Actuator Type………………... 65

Table 4.2.2 – Design Criteria and Ratings for each Actuator Type…………….......... 67

Table 4.4.1 – Specifications Summary the F28377D Delfino Experimenter Kit……… 79

Table A.1 – Specifications Summary for C2000 Piccolo Launchpad…………….…... 100

Table A.2 – Specifications Summary for C2000 Piccolo Launchpad vs. the F28377D Delfino Experimenter Kit………………... 103

List of Figures

Figure 1.2.1 – Kinematic Chain of a Serial Manipulator……………………………..... 3

Figure 1.2.2 – "Scara" type robot…………………………………………………… 5

Figure 1.2.3 – Kinematic Model for the 6-6 Type of Stewart Platform……………….. 6

Figure 1.3.1 – Fixed Orientation Spray Method…………………………………….. 8

Figure 1.3.2 – Stewart Platform Spray Capabilities…………………………………… 9

Figure 2.1.1 – Three Types of Stewart platforms with Index Numbers Denoting the Number of Connection Vertices on the Base and the Platform………………………………………………………... 12

Figure 2.1.2 – Kinematic Model for a 3-DOF Stewart Platform………………………. 13

Figure 2.2.1 – Platform Joint Locations………………………………………………... 15

Figure 2.2.2 – Platform Joint Locations………………………………………………... 15

Figure 2.3.1 – Stewart Platform in its Initial Configuration Where $\alpha_i = 0$ and $\beta_i = 0$, Stewart Platform in a Raised Position Where $\alpha_i > 0$, Stewart Platform in a Raised Position Where $\beta_i > 0$ …………………………………….. 19

Figure 2.3.2 – Assumed Circular Spray Trajectory……………………………………. 20

Figure 2.3.3 – Constraint Plane Defined for l_1………………………………………… 22

Figure 2.3.4 – Diagram Illustrating the Maximum Allowable Distance D from the Constraint Plane……………………………………………………... 23

Figure 2.4.1 – Flowchart for the Algorithm used to Generate the 3-D Vector Workspace…………………………………………………….…..... 27

Figure 2.4.2 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (Top View)……………………………………………………….... 28

Figure 2.4.3 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (Side View)…………………………………………….................. 28

Figure 2.4.4 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (3-D View)……………………………………………………….... 29

Figure 2.4.5 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (3-D View)…………………………….... 29

Figure 2.4.6 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (Side View)………………………………….. 30

Figure 2.4.7 – Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (Top View)………………………………….. 30

Figure 2.4.8 – 3-D Workspace of a 3-DOF Stewart Platform Visualized Using Vectors Normal to each Platform Orientation at 0.1 Radians/Step……. 31

Figure 2.4.9 – 3-D Workspace of a 3-DOF Stewart Platform Top View at 0.1 Radians/Step (Down Z-Axis)…………………………………………... 32

Figure 2.4.10 – 3-D Workspace of a 3-DOF Stewart Platform Side View at 0.1 Radians/Step (Down Y-Axis)………………………………………… 32

Figure 2.4.11 – 3-D Workspace of a 3-DOF Stewart Platform Side View at 0.1 Radians/Step (Down X-Axis)………………………………………… 33

Figure 2.4.12 – 3-D Workspace of a 3-DOF Stewart Platform Visualized Using Vectors Normal to each Platform Orientation at 0.01 Radians/Step…. 33

Figure 2.4.13 – 3-D Workspace of a 3-DOF Stewart Platform Top View at 0.01 Radians/Step (Down Z-Axis)………………………………………… 34

Figure 3.1.1 – Architecture of the Delta Robot……………………………………... 37

Figure 3.1.2 – *i*th Subchain of the Delta Robot……………………………………... 37

Figure 3.3.1 – A Typical Number of Function Evaluations for the CRS Algorithm to Convergence When Determining the Optimal Design of a Delta Robot. 42

Figure 4.1.1 – Relative Lifespan of Bearings vs. Water Concentration in the Hydraulic Fluid…………………………………………………… 61

Figure 4.1.2 – Hydraulic Actuator Diagram………………………………………… 62

Figure 4.1.3 – Pneumatic Actuator Diagram………………………………………... 63

Figure 4.1.4 – Electric Actuator Diagram…………………………………………… 64

Figure 4.2.1 – Figure 4.2.1 Overview of Electrical System Components for a 3-DOF Stewart Platform…………………………………………………….. 68

Figure 4.3.1 – Thomson Electrak 1 Schematic……………………………………… 70

Figure 4.3.2 – Thomson Electrak 1 3-D Solidworks Model………………………… 70

Figure 4.3.3 – MW126 Power Supply (Front View)………………………………... 72

Figure 4.3.4 – MW126 Power Supply (Top/Side View)……………………………. 72

Figure 4.3.5 – AndyMark Talon SR Speed Controller……………………………… 73

Figure 4.3.6 – Talon SR motor controller CIM motor RPM vs PWM test results, courtesy of Whirlpool motor lab…………………………………….. 74

Figure 4.3.7 – Talon SR motor controller CIM motor RPM vs load torque test results, courtesy of Whirlpool motor lab…………………………………………. 75

Figure 4.4.1 – F28377D Delfino Experimenter Kit……………………………………. 80

Figure 4.4.2 – Electrical System Components for a 3-DOF Stewart Platform with all Parts Listed……………………………………………………………….. 81

Figure 5.2.1 – Experimental Set-up……………………………………………………. 85

Figure 5.2.2 – Experimental Set-up with Side View of Webcam and Arm.................... 85

Figure 5.2.3 – View of Entire Experimental Set-up after Orientation Paper and Arm Added... 86

Figure 5.2.4 – Close-up View of Orientation Paper.. 86

Figure A.1 – 68HC12 Microcontroller mounted on Platform………………………... 99

Figure A.2 – C2000 Piccolo Launchpad……………………………………………… 101

List of Acronyms

CCS – Code composer studio

CRS – Controlled random search

DE – Differential evolution

DOF – Degrees of freedom

GA – Genetic algorithm

PKM – Parallel kinematic machine

PSO – Particle swarm optimization

SQP – Sequential quadratic programming

Chapter 1
Introduction

1.1 Motivation of Study

A major component of the infrastructure of any modern city is a network of underground pipes that transport drinking water, sewage and storm water. Most of the pipes currently being used are made out of concrete or various plastics. As with any material, they have an expected lifespan after which deterioration begins to occur. This can result in cracks, and in some cases, even large holes in the pipe which can cause a complete loss of function of the pipe. These defects invariably lead to water losses that necessitate the repair of the pipeline, which is an expensive undertaking.

Traditionally, the pipes are repaired by a process that involves an inefficient excavation that often requires roads, lawns and sidewalks to be torn up. In recent years, trenchless pipe rehabilitation has been revolutionizing the pipeline repair industry. Trenchless pipeline rehabilitation uses tele-operated tools inserted into the system through access ports that allow the operator to inspect, clean and repair the interior of the pipe. Currently, the state-of-the-art process uses an inflatable tube, which has an external liner attached that is soaked with the repairing solution.

Although the current method is effective, it has some drawbacks. Firstly, each liner needs to be customized to the pipe that is being repaired. This means that each pipe must be inspected and measured, using the tele-operated tools mentioned above, before the liner can be fabricated. Another drawback is that the entire process of soaking the liner with the solution, fitting it around the inflatable tube, inserting the tube into the pipe, allowing it time to cure and finally removing the inflatable tube, is very time consuming and requires a lot of manpower. In addition to that, there are cases when the inflatable tube is very difficult to remove and, in the worst cases, a trench must be dug in order to remove it.

Keeping in mind the drawbacks mentioned above, a new approach was imagined that would eliminate the need for the inflatable tube and liner. Instead, they would be replaced with a tele-operated robot. This robot would be outfitted with a spray head which would be used to deposit a repairing solution onto the damaged areas of the pipe. This would give the user far more control over exactly where the repairing coating would be deposited and how much would be deposited, when compared to the current technology. If the robot were also equipped with a depth sensor and a camera, this would give the user the information needed to decide the amount of repairing solution that is required and where it should be deposited in order to repair the damage correctly. With these additional tools, this could possibly eliminate the need to perform the initial inspection that currently needs to be done before the liner is fabricated. Lastly, the robot would require only one person to operate it, compared to the six or seven people that are required to carry out the current process, thus cutting down on the cost of the repair.

1.2 Design Considerations

When considering the design of robot, there are two possible choices: serial manipulator or parallel manipulator. For most people, the more familiar robot design is a serial manipulator. This type of manipulator typically works similar to a human arm, where there are a series of links that are connected by a motor-actuated joints. These links and joints comprise several kinematic chains, each of which contributes to the overall motion of the end effector. Typically, serial manipulators have six joints, thus giving the end effector six degrees of freedom [1]. This means that this type of robot has a fairly large workspace, although this is limited by singularities and the maximum length of the robot itself. This type of robot also has disadvantages, such as a relatively low load that can be effectively manipulated, error accumulation and amplification from link to link, a low stiffness, low dexterity, high motion coupling [2]. All of these drawbacks are related to the openness of the kinematic structure, where the cumulative weights of each actuator and joint adversely affect the performance and load bearing capacity of the robot. In addition to that, the additional weight of each link and actuator, which will have a different effect depending on the configuration and orientation of the robot, complicates the kinematic calculations. Finally, when performing these kinematic calculations, the effect of moving through the kinematic chains leads to error accumulation and amplification, which can greatly impact the final result. This can mean that the position of the end effector differs from what was calculated, due to error accumulated at each link, but how much it differs is dependent on many parameters, particularly the stiffness of the joint.

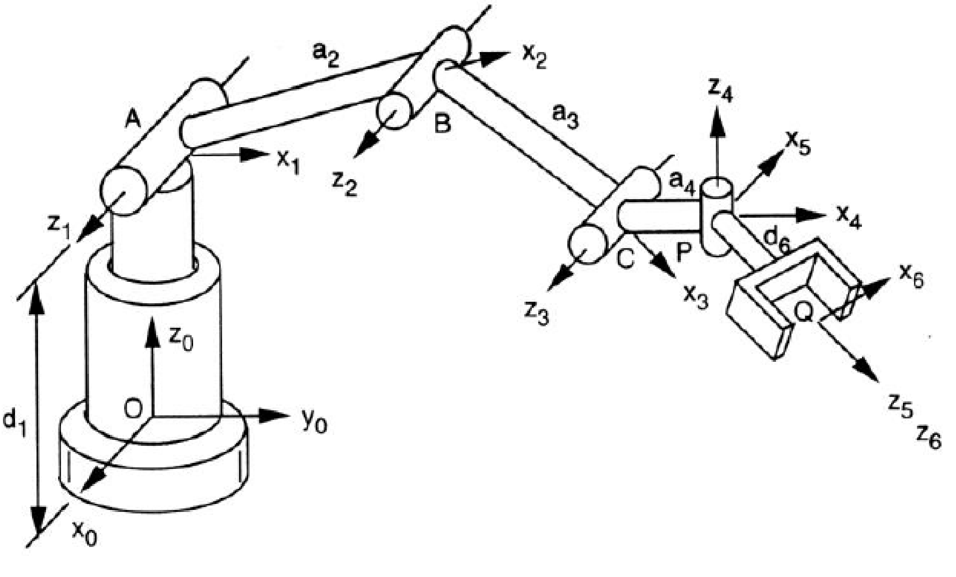

Figure 1.2.1 Kinematic Chain of a Serial Manipulator [3]

Serial manipulators are commonly used in industry due to large workspace and easily calculated forward kinematics. Examples of this type of robot, as well as their important physical characteristics were reported by Merlet and summarized in Table 1.2.1 [4].

Table 1.2.1 Serial Robot Characteristics

Robot	DOF	Mass (kg)	Payload (kg)	Repeatability (mm)	Payload/Mass
Adept 3XL	4	265	25	± 0.038	0.0943
Epson E2S45x	4	20	5	± 0.015	0.25
Seiko EH850	4	43	10	± 0.025	0.2325
Toshiba SR-504HSP	4	38	2	± 0.02	0.0526
ABB IRB 140T	6	98	5	± 0.03	0.051
Fanuc M420iA	6	620	40	± 0.5	0064516
Kuka KR 60-3	6	665	60	± 0.2	0.09022

Some interesting points arise from this data. It is very clear that the payload/mass ratio is quite low, and exceptionally low for the serial robots which have heavier payloads. The robots that are listed in the table as having 4-DOF (degrees of freedom) are Scara type, which means they can move in the 3 translational directions and rotate around a vertical axis, and the robots listed as having 6-DOF are spherical type and capable of moving in the 3 translational directions as well as rotate around the 3 axes. Due to their design, the Scara type robots generally have more support around their joints and therefore have a much higher payload/mass ratio. Nevertheless, the ratio is still quite low and in applications where a high payload/mass ratio is required there is another type of robot that is available that exhibits such a characteristic.

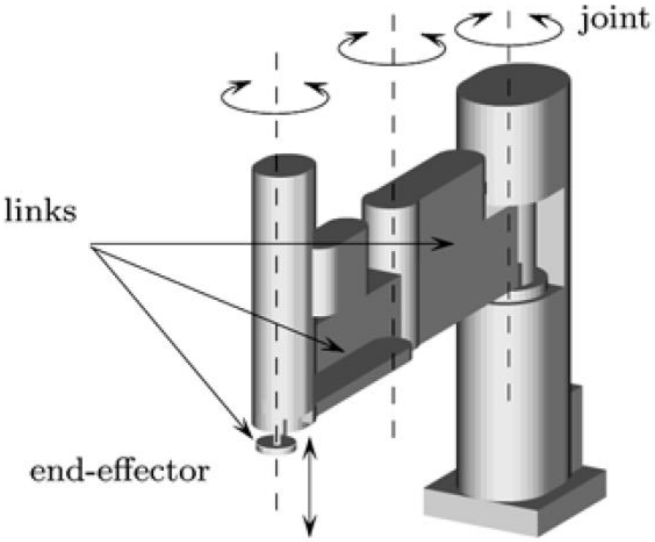

Figure 1.2.2 "Scara" type robot [5]

The other possible choice for the robot design is a parallel manipulator, with the best known design being the Stewart-Gough platform. This design differs from the serial manipulator in the fact that the end effector is connected to the base by multiple separate and independent links. Each of these links work in parallel to move and orient the end effector. Typically, these parallel manipulators will have six legs in order to have the ability to operate with six degrees of freedom.

The main benefit of this design is the rigidity and stiffness of the robot, due to the end effector being supported by multiple arms. This means that the end effector can move with high precision and speed. It also means that parallel manipulators are capable of supporting a much higher load when compared to serial manipulators. The drawback of this design is the limited workspace and their nonlinear behaviour. The nonlinearity of the parallel manipulator means that a command to the actuators that produces a linear movement and one point in the workspace likely will not produce the same movement at another point [6].

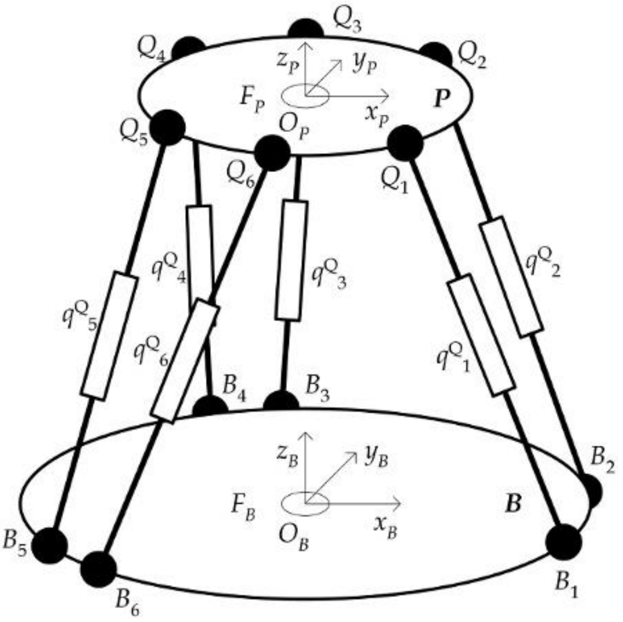

Figure 1.2.3 Kinematic Model for the 6-6 Type of Stewart Platform [7]

1.3 Research Objectives

As mentioned in section 1.1, the overall objective of this project was to create a tele-operated robot that was capable of spraying a structural coating in order to repair cracks or holes in the existing pipe infrastructure, with an additional focus on repairing the structure in the manholes leading into the pipe. This project was divided into two major parts, the first being the development of the robot, and the second being the development of a spray-head and the spraying liquid that will be deposited on the pipe. This thesis deals solely with the research and development of the robot.

The overall aim of this project was to develop a system that could deposit the coating onto the wall of any diameter of pipe or manhole and would have the flexibility in its design so that it could be scaled up or scaled down to accommodate larger or smaller diameter pipes, respectively. Additionally, there was a requirement for more accuracy and control when compared to the existing technology used by other companies in the pipeline repair industry

[8]. The current technology requires the user to pull a spray head through the pipe while the spray head deposits the coating on the wall. This method lacks a precision when depositing the coating because the spray head is only able to spray in a direction that is perpendicular to the movement of the spray head carriage. This does not give the user any control over the angle at which the spray is deposited, which is an important design feature when trying to repair a pipe with unknown and unpredictable damage.

To reinforce the importance of this point, one needs only to consider the difference in repairing a large gaping hole in the pipe with that of a small thin crack in the pipe. In both cases, the user could simply use a large amount of coating to repair each defect, or the user could use the coating more efficiently (i.e. using enough to repair the pipe damage), thus saving money by reducing the amount of spray used. Additionally, the current technology coats the entire pipe with the repairing solution, rather than targeting the individual sections of the pipe that are damaged, which is also a very inefficient use of the coating.

The flexibility in controlling the orientation of the spray head is crucial to giving the user the ability to repair the pipe in one pass, rather than first needing to use an inspection tool to collect data about the pipeline in order to customize the repair method of each pipe. Using the proposed robot design, the user would be able to adjust the spray angle, spray distance and the depth of the deposit “on the fly” as the user is given information about the pipe repair through the use of a camera attached to the robot, for example.

An additional consideration when spraying pipes is that often there is variability in the diameter of the pipe at different sections in the pipeline. This variability in the diameter may be due to corrosion, deposits or because a cut out and replacement repair was performed that caused a mismatch in the diameter of the pipe. If a spray head with a fixed orientation was used, it would be impossible to reach certain areas of the pipe with the spray cone. This scenario is illustrated in figures 1.3.1 and 1.3.2, where the importance of a spray head that is capable of changing its orientation, is demonstrated. In figure 1.3.1, we can see that even in the best possible position, a fixed orientation spray head would be unable to reach the corner areas of the pipeline where the diameter of the pipe has changed. Even in the areas which are reachable, it would be difficult to build up the layer of coating because only the very edge of the spray cone is able to touch these sections of the pipe, and at edge of the spray cone the density of the spray would be very small. In figure 1.3.2, the capabilities of the Stewart platform are demonstrated. By having a spray head that is capable of changing its orientation, it can reach the corner areas of the pipe that the fixed orientation spray head could not reach. Therefore, the necessity of a spray head that can change its orientation is shown, and this necessity is especially relevant when dealing with the unpredictability of the diameters in the different sections of the pipeline.

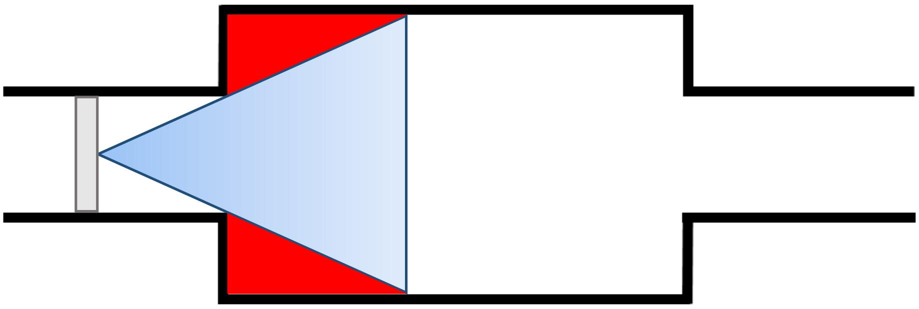

Figure 1.3.1 Fixed Orientation Spray Method

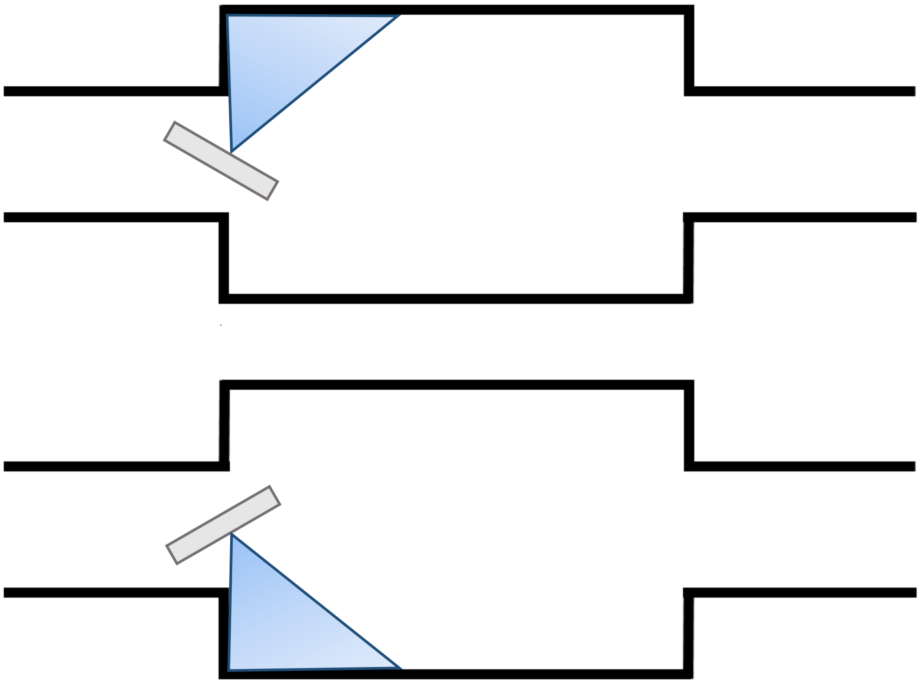

Figure 1.3.2 Stewart Platform Spray Capabilities

The original idea for this research project was the result of a collaboration between Liquiforce and the University of Western Ontario (specifically Dr. Ken McIsaac and Derek Brecht). Liquiforce specializes in trenchless pipeline rehabilitation for clean water, waste water and other pipeline systems, and is a well-established leader in southern Ontario in this field. Originally, Liquiforce wanted to expand their pipeline repair process capabilities through the use of a tele-operated robot, which could be used to repair the pipe more efficiently, and cost effectively, in certain situations when compared to the inflatable tube/liner repair approach that was described in section 1.1.

During the course of this collaboration, the scope of the project evolved and expanded to include the design of a larger robot that would be used for manhole rehabilitation. Due to the similarity of these tasks, it was decided that a prototype for the manhole rehabilitation would be designed and constructed first. The reasoning behind this decision was that it would be less difficult to construct and test a larger robot, and the lessons that were learned during its construction and testing could be applied to the construction of the smaller robot. This design approach stems from idea that processes of pipeline and manhole rehabilitation are similar, and because the processes are similar, the design of the larger robot could be scaled down to a size that would allow it to be used for pipeline rehabilitation, which involves pipes with a significantly smaller diameter than the manhole.

1.4 Contributions

The major research contributions that can be found in this thesis are as follows:

- Designed and developed a 3-DOF Stewart platform through optimization techniques and by analyzing its kinematics properties
- Improved the speed and efficiency of existing optimization algorithms through a novel combination of the controlled random search and particle swarm optimization algorithms
- Constructed a 3-DOF Stewart platform by integrating components that were chosen by a thorough selection process
- A component selection process was recorded in detail to be used as a tutorial for related projects in the future
- Implemented a trajectory that mimics a probable spraying trajectory on a 3-DOF Stewart platform prototype to verify proof of concept

Chapter 2

Background and Kinematic Analysis

2.1 Parallel Manipulators

Parallel manipulators have been used in many different industrial applications. One of the first proposed applications was a tire testing machine [9]. For this application, the proposed machine needed a full range of motion so that a load could be applied to the tire in any possible direction, in order to determine how the tire responded to loads applied in the lateral or a tangential direction, as opposed to the traditionally tested radial directions. Another early application was to use the parallel manipulator design as a flight simulator [10]. Eventually, the potential of parallel manipulators was fully recognized and they began seeing use in many other areas including being used as a camera-orienting device [11], underground excavator [12] and in satellite positioning [13].

The Stewart platform is a specific type of parallel manipulator which traditionally has six legs and six DOF, although variations do exist. This allows for many possible positions and orientations of the top platform, which are only limited by the length of its legs and the range of motion of its joints [14]. Figure 2.1.1 shows the most commonly seen configurations, with the 6-6 type being the most common. The first number denotes the number of platform points and the second number denotes the number of base points. Ideally, both the platform and the base joints would be spherical joints so that the robot’s movement would not be impeded by the range of the joint, although the necessity of this is highly dependent on the application. In most cases, the platform and base joints are all universal type joints with two DOF.

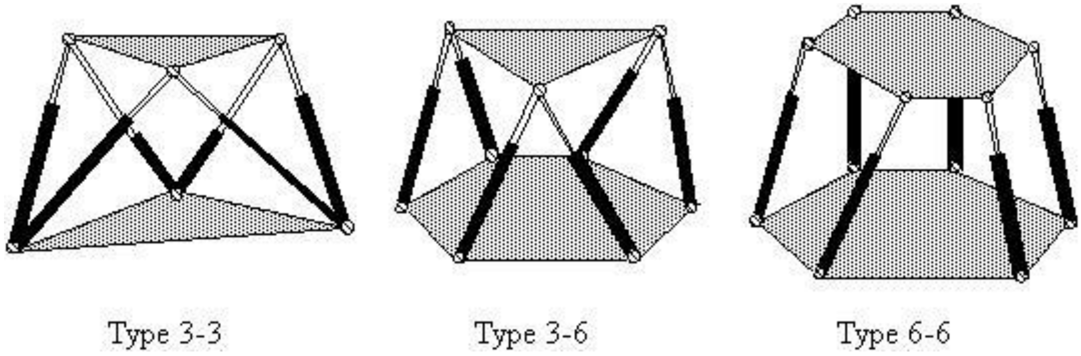

Figure 2.1.1 Three Types of Stewart Platforms with Index Numbers Denoting the Number of Connection Vertices on the Base and the Platform [15]

In recent years, research has been conducted into the feasibility of having a type 3-3 robot, with only three legs [16] [17]. This design has the drawback of reducing the number of degrees of freedom to three, but has the advantages of less complicated kinematics and lower overall weight. Grübler’s formula can be applied to examine the number of DOF for a closed loop parallel kinematic machine (PKM) [18].

$$F_e = \lambda(l - j - 1) + \sum_{i=1}^{j} f_i - I_d \tag{1}$$

Where: F_e is the effective DOF of the assembly or mechanism,
λ is the DOF of the space in which the mechanism operates,
l is the number of links,
j is the number of joints,
f_i is the DOF of the *i*th joint, and
I_d is the idle or passive DOFs.

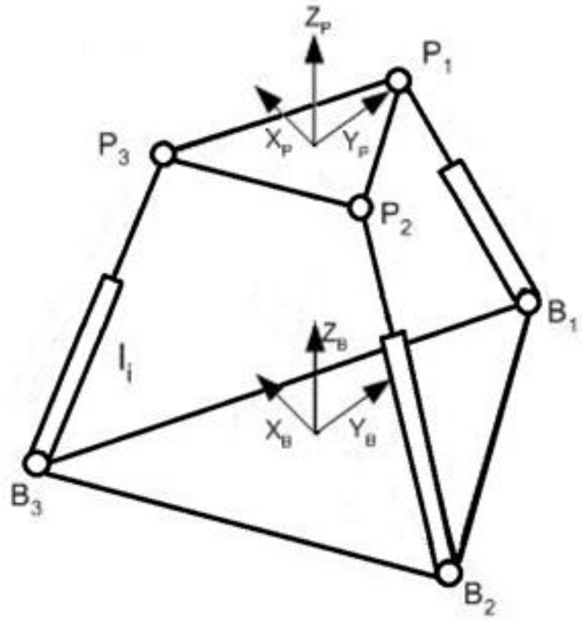

Figure 2.1.2 Kinematic Model for a 3-DOF Stewart Platform

This will be demonstrated with an example. Based on the model in figure 2.1.2, each leg is composed of two links and both the base and the platform count as one link. Therefore, there are eight links in total in the robot. There are six universal joints and three prismatic joints, bringing the total number of joints to nine. The number of DOF of the space that the robot operates in are six. Finally, each universal joint has two DOF and each prismatic joint has one DOF, bringing the total number of DOF for all of the joints to 15. Using these numbers in the formula we have $F_e = 6(8 - 9 - 1) + 15 = 3$. Therefore, this type of platform has three DOF.

2.2 Inverse Kinematics of the Stewart Platform

The kinematics of a Stewart platform is the relationship between the joint coordinates and the end effector. If the position of the end effector is known, then one would use the inverse kinematics to find the joint coordinates. If the joint coordinates are known, then one would use the forward kinematics to find the position of the end effector. Generally speaking, it is more intuitive to use the inverse kinematics because it is likely preferable to be able to direct the end effector where to go, rather than knowing where the joints of the robot are and having to calculate where the end effector is.

In addition to the benefit of the inverse kinematics being more intuitive for a human user, for a PKM, a nonlinear closed form solution to the inverse kinematics generally exists and is relatively easy to find. In contrast, it is much more difficult to find a closed form solution for the forward kinematics for a PKM, if one exists at all. For a serial manipulator, however, the opposite is true. A closed form solution typically exists, and can be easily constructed, for the forward kinematics. The solution of the inverse kinematics, however, is more computationally expensive and difficult to solve [19].

Since the legs of a PKM are controlled by actuators, which usually have some sort of sensor that gives the user information about their length, the most logical use of the inverse kinematics would be to determine the required leg lengths in order to pose the top platform in the desired orientation. The method consists of using the joint coordinates and a rotation matrix to calculate leg lengths, given the rotation about the x-axis, ψ (roll), the rotation about the y-axis, θ (pitch), and the rotation about the z-axis, $\emptyset$ (yaw) and the translation vector ${}^{B}\vec{q} = [q_x \quad q_y \quad q_z]^T$, where q_x, q_y and q_z are the x, y and z components of the translation vector that describes the translation from reference frame {B} to {P}, with the initial q_z being set to 31.75 cm. This q_z value of 31.75 cm represents the minimum height of the platform, with each actuator at its minimum stroke length. Each of the aforementioned rotations is relative to the reference frame for the platform, {P}. This method is easily modified so that only three leg lengths are calculated in the case of a 3-DOF platform. The leg lengths are calculated as follows [20].

Referring to figures 2.2.1 and 2.2.2, the base joint centers are defined as ${}^{B}\overrightarrow{b_i} = \left[b_{ix}\ b_{iy}\ b_{iz}\right]^T$ and the platform joint centers are defined as ${}^{P}\overrightarrow{p_i} = \left[p_{ix}\ p_{iy}\ p_{iz}\right]^T$. Since each of the base and joint platform centers are equally spaced at 120° from one another, their location can be described using geometry and special triangles. Where r is the radius of the base and R is the radius of the platform, which are 19.05 cm and 9.525 cm, respectively. These radii were chosen

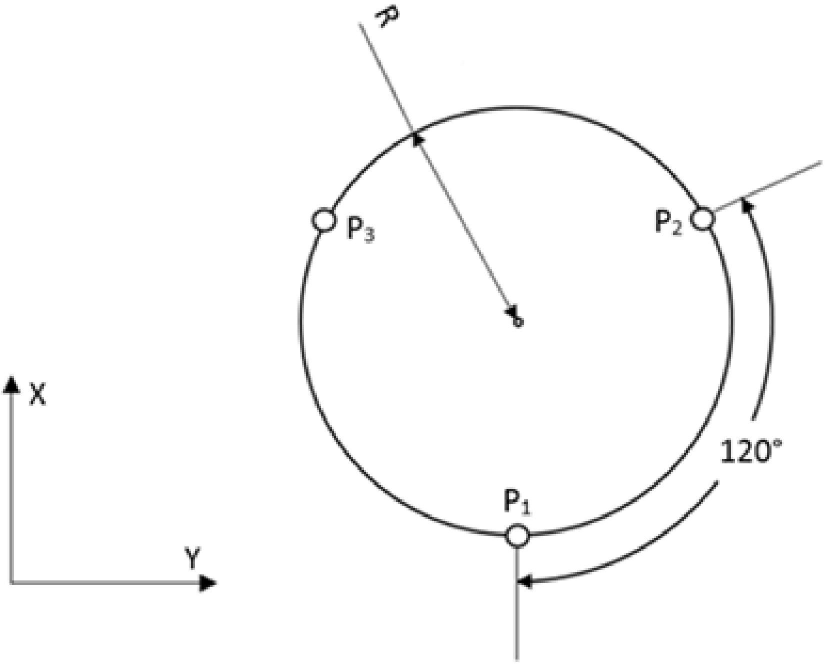

Figure 2.2.1 Platform Joint Locations

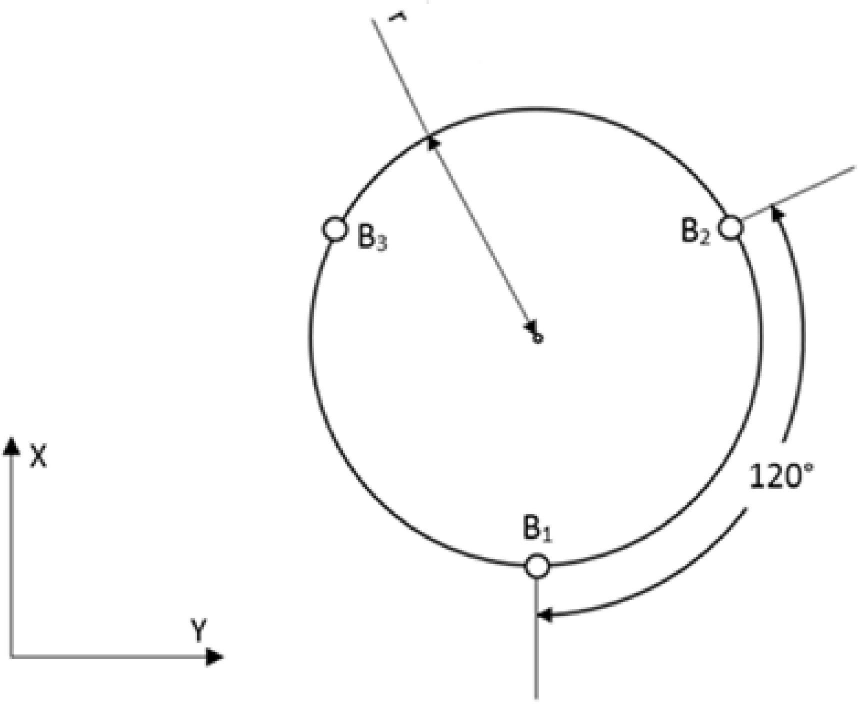

Figure 2.2.2 Base Joint Locations

so that the Stewart platform would have the requisite dimensions for the manholes and pipelines that it would be spraying inside of.

$${}^{P}\overrightarrow{p_1} = [r, \qquad 0\,, \qquad 0]^T$$

$${}^{P}\overrightarrow{p_2} = \left[-\frac{1}{2}r, \qquad \sqrt{\frac{3}{2}}r\,, \qquad 0\right]^T$$

$${}^{P}\overrightarrow{p_3} = \left[-\frac{1}{2}r, \qquad -\sqrt{\frac{3}{2}}r\,, \qquad 0\right]^T$$

$${}^{B}\overrightarrow{b_1} = [R\,, \qquad 0\,, \qquad 0]^T$$

$${}^{B}\overrightarrow{b_2} = \left[-\frac{1}{2}r, \qquad \sqrt{\frac{3}{2}}r\,, \qquad 0\right]^T$$

$${}^{B}\overrightarrow{b_3} = \left[-\frac{1}{2}r, \quad \sqrt{\frac{3}{2}}r, \quad 0\right]^T$$

The homogeneous transformation ${}^{B}\boldsymbol{T}_P$ relates the transformation from the platform reference frame, {P}, to the base reference frame, {B}. The rotation matrix ${}^{B}\boldsymbol{R}_P$ within the homogenous transformation ${}^{B}\boldsymbol{T}_P$ is defined using the pitch, yaw and roll convention that was described in the paragraph above. It is populated as follows [21]:

$${}^{B}\boldsymbol{R}_P = \begin{bmatrix} \cos\theta\cos\emptyset & \cos\theta\sin\emptyset & -\sin\theta \\ \sin\psi\sin\theta\cos\emptyset - \cos\psi\sin\emptyset & \sin\psi\sin\theta\sin\emptyset + \cos\psi\cos\emptyset & \cos\theta\sin\psi \\ \cos\psi\sin\theta\cos\emptyset + \sin\psi\sin\emptyset & \cos\psi\sin\theta\sin\emptyset - \sin\psi\cos\emptyset & \cos\theta\cos\psi \end{bmatrix} \quad (2)$$

The homogeneous transformation ${}^{B}\boldsymbol{T}_P$ matrix is populated as follows:

$$^{B}\boldsymbol{T}_{P} = \begin{bmatrix} & ^{B}\boldsymbol{R}_{P} & & q_x \\ & & & q_y \\ & & & q_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

In order to find the leg lengths we need first to transform the coordinates $^{P}\overrightarrow{P_{i}}$ into the base frame (i.e.$^{B}\overrightarrow{P_{i}}$). To accomplish this we use the following relationship:

$$\begin{bmatrix} P_{ix} \\ P_{iy} \\ P_{iz} \\ 1 \end{bmatrix} = {}^{B}\boldsymbol{R}_{P} \begin{bmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \\ 1 \end{bmatrix}$$

Next, the vectors that describe each of the legs are found. This is done by subtracting $^{B}\overrightarrow{b_{i}}$ from $^{B}\overrightarrow{P_{i}}$, which is possible since they are now in the same reference frame. Then the magnitude of each of those vectors is found in order to determine the length of each leg. The following equations are used to calculate the length of each leg:

$$l_1 = \sqrt{(P_{1x} - b_{1x})^2 + (P_{1y} - b_{1y})^2 + (P_{1z} - b_{1z})^2} \tag{4}$$

$$l_2 = \sqrt{(P_{2x} - b_{2x})^2 + (P_{2y} - b_{2y})^2 + (P_{2z} - b_{2z})^2} \tag{5}$$

$$l_3 = \sqrt{(P_{3x} - b_{3x})^2 + (P_{3y} - b_{3y})^2 + (P_{3z} - b_{3z})^2} \tag{6}$$

We now have a method with which we can determine the length of each leg for any orientation of the platform, as long as we are given ψ, θ, ∅ (yaw) and $^{B}\vec{q}$, all of which are easily determined and each can be controlled by the robot operator.

2.3 Workspace Calculation

The workspace of any robot is the set of points and orientations that the robot is able to reach given the physical and mechanical limitations of the design. When designing any type of robot it is essential that the workspace is calculated before the robot is manufactured. This needs to be done because the designer must know that the robot is capable of performing all of the tasks that it will be required to perform. The only way to be sure of this is to perform a detailed workspace analysis.

For the purposes of planning a spraying trajectory and because the platform has only 3-DOF, a number of simplifications can be made to the method proposed by Bonev and Ryu [22]. Firstly, since the robot has only three legs, all of which are located 120° from each other, there is no need to check for leg interference. This lack of leg interference was observed by Chung even in a 6-DOF Stewart platform [23]. Instead, the most limiting criteria will be checked, which is whether the calculated leg length exceeds the maximum or minimum length of the actuator.

2.3.1 Actuator Stroke Length

The limited stroke of each actuator imposes a constraint on each actuator leg l_i. This can be expressed as $l_{min} \leq l_i \leq l_{max}$, for $i = 1, 2, 3$ and where l_{max} is the maximum length of the actuator and l_{min} is the minimum length of the actuator. Since all three actuators are identical and their limits have been measured and documented, $l_{min} = 31.75$ cm and $l_{max} = 41.91$ cm. These lengths include the housing of the actuator.

2.3.2 Range of the Passive Joints

Each of the six joints has a range limitation on its angular motion. The joints which connect the bottom of each actuator to the base of the robot are revolute joints. Since they are revolute joints, they have only have one degree of freedom that is unrestricted. However, there are physical limitations on the robot, around that degree of freedom, at which the actuator housing will come into contact with the base and prevent any further movement in that direction. Therefore, a constraint of 45° of maximum misalignment is imposed on this joint, which is illustrated in figure 2.3.1 as β_i. The joints which connect the top of the actuator to the platform are spherical joints. These also have a range limitation because part of the platform will come into contact with the actuator leg. A constraint of 25° of maximum misalignment is imposed on this joint, which is illustrated in figure 2.3.1 as α_i.

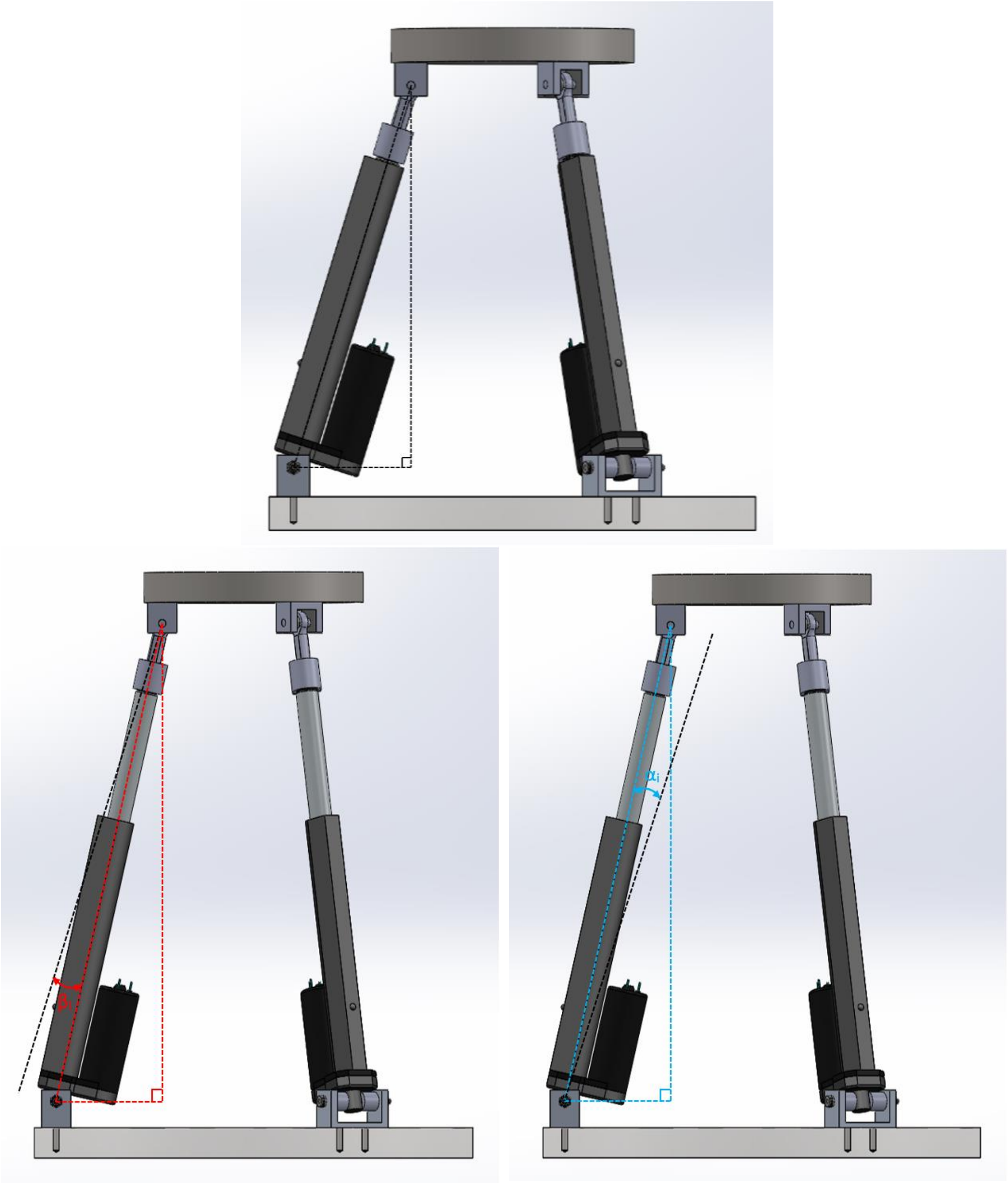

Figure 2.3.1 (Clockwise from top picture) Stewart Platform in its Initial Configuration Where $\alpha_i = 0$ *and* $\beta_i = 0$*, Stewart Platform in a Raised Position Where* $\alpha_i > 0$*, Stewart Platform in a Raised Position Where* $\beta_i > 0$

Assuming that we are spraying in a circular motion where the radius of the circle is 19.05 cm (the typical radius of a manhole) and the height above the robot that we are spraying is 50 cm, we can show that the pitch and roll angles will not exceed 25° at any point during the spray trajectory. In order to illustrate what this trajectory, we refer to figure 2.3.2, where r = 19.05 cm and d = 50 cm. The hypotenuse of the triangle that is formed (h) is the vector normal to the platform (i.e. the center of the spray cone), and should not be confused with the limits of the spray cone, which are not considered, or relevant, to this example. We are solely concerned with spraying in a circular trajectory, regardless of the shape of the spray cone itself. Using basic trigonometry we can demonstrate that the maximum pitch and roll angles of the platform will not exceed 25° when the robot is moving during the spraying motion:

$$\tan\theta = \frac{r}{d} = \frac{19.05\ cm}{50\ cm}$$

$$\theta = \tan^{-1}\left(\frac{19.05\ cm}{50\ cm}\right)$$

$$\theta = 20.8°$$

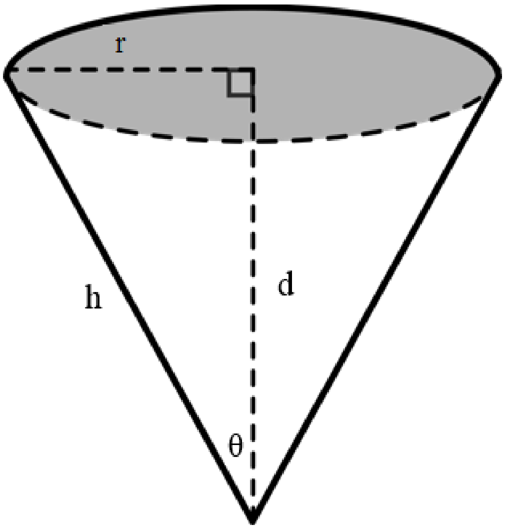

Figure 2.3.2 Assumed Circular Spray Trajectory

Therefore, in practice, it will not be possible for the robot to exceed any of the aforementioned misalignment constraints on the passive joints.

However, when analyzing the kinematics, it can be shown that there is a possibility for significant misalignment of the joint in the directions that were mentioned in the above paragraphs. In order to ensure that the range of motion constraints on the joints are not exceeded (due to interference between the actuator housing and the base, and between the actuator leg and the platform), we must check the maximal misalignment constraints that are imposed on the spherical and revolute joints.

This can be done using the following relationship, which relates the height difference between the base and the *i*th platform point (P_{iz}) and the leg lengths (l_i) to the revolute joint angular constraint (α_i) and the spherical joint angular constraint (β_i):

$$\sin^{-1}\left(\frac{P_{iz}}{l_i}\right) \leq \alpha_i, for\ i = 1,2,3 \tag{7}$$

$$\cos^{-1}\left(\frac{P_{iz}}{l_i}\right) \leq \beta_i, for\ i = 1,2,3 \tag{8}$$

In addition to the above misalignment constraints, an additional constraint must be included in the algorithm which excludes any points that require the revolute joint to be excessively strained and rotate in the direction pictured in 2.3.4. This constraint can be implemented by having a limitation on the distance that a potentially viable ${}^{B}\overrightarrow{P_i}$, is from the *i*th plane, which is defined as passing through both the center of the *i*th actuator leg, and the z-axis in reference frame {B}. Figure 2.3.3 depicts this plane for one of the legs, and two other planes would be defined using the same criteria for the remaining legs.

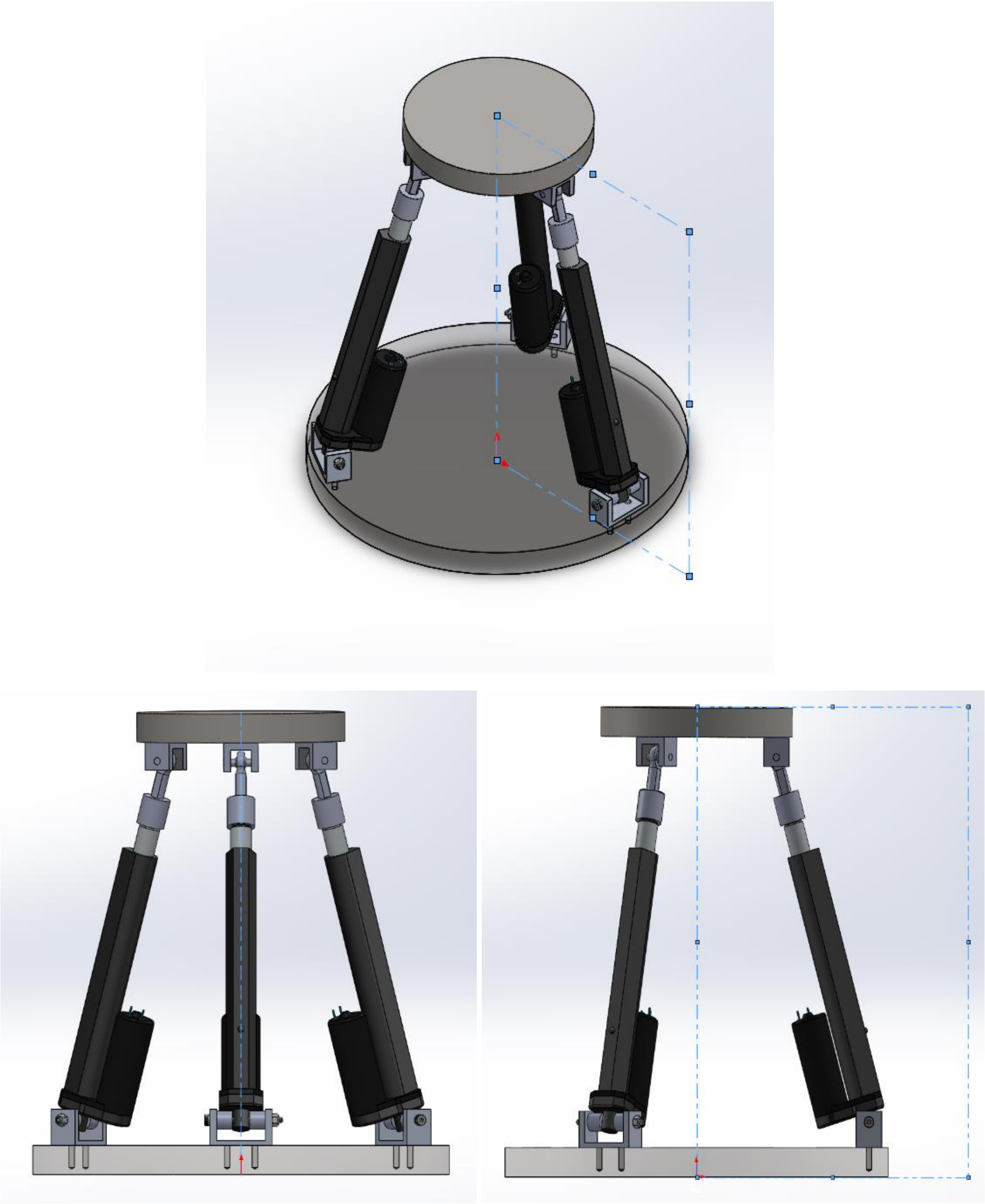

Figure 2.3.3 Constraint Plane Defined for l_1

Now that the constraint plane has been defined, we need to define the maximum distance that an acceptable ${}^{B}\vec{P_i}$ can be removed from the plane, which will be designated as *D*, as shown in figure 2.3.4. Again, referring to figure 2.3.4, the maximum angle of the revolute joint (Δ) has been observed to be 0.45°. Using Δ, and the l_{min} (the length at which the largest Δ occurs), we can calculate *D* using basic trigonometry:

$$\tan \Delta = \frac{D}{l_{min}}$$

$$D = l_{min}(\tan \Delta)$$

$$D = (31.75\ cm)(\tan(0.45°))$$

$$D = 0.25\ cm$$

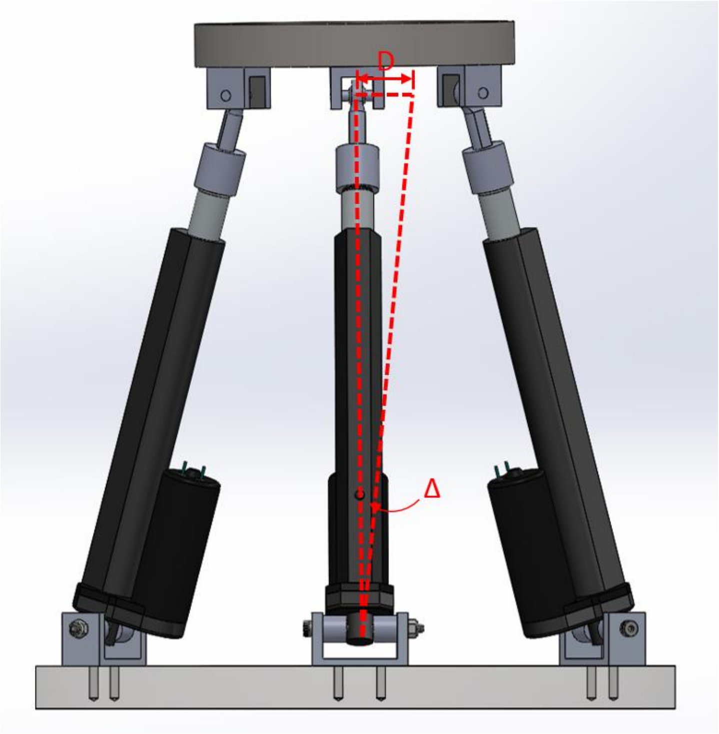

Figure 2.3.4 Diagram Illustrating the Maximum Allowable Distance D from the Constraint Plane

From the above calculation, we can see that the largest allowable D value, without causing excessive strain and damage to the actuator housing, would be 0.25 cm. In terms of the kinematic simulation, 0.25 cm is the largest distance that the ${}^{B}\vec{P_i}$ points can be removed from the plane. Since the constraint plane and the maximum allowable distance is defined, we can use a projective geometry method to determine if each potentially viable point ${}^{B}\vec{P_i}$ meets the distance constraint. This method is demonstrated as follows [24]:

Given a plane defined using the general equation of a plane:

$$ax + by + cz + d = 0 \tag{9}$$

The normal vector is given by:

$${}^{B}\overrightarrow{n_\imath} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{10}$$

We have a potentially viable point defined as:

$${}^{B}\overrightarrow{P_\imath} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{11}$$

The vector the plane to the point is given by:

$${}^{B}\overrightarrow{v_\imath} = -\begin{bmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{bmatrix} \tag{12}$$

If we project ${}^{B}\overrightarrow{v_\imath}$ onto ${}^{B}\overrightarrow{n_\imath}$, we can get the distance D from the point to the plane:

$$D = \left| proj_{{}^{B}\overrightarrow{n_\imath}} \quad {}^{B}\overrightarrow{v_\imath} \right| \tag{13}$$

$$= \frac{\left| {}^{B}\overrightarrow{n_\imath} \cdot {}^{B}\overrightarrow{v_\imath} \right|}{\left| {}^{B}\overrightarrow{n_\imath} \right|} \tag{14}$$

$$= \frac{\left| a(x - x_0) + b(y - y_0) + c(z - z_0) \right|}{\sqrt{a^2 + b^2 + c^2}} \tag{15}$$

By simplifying:

$$D = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{16}$$

From this equation we can determine whether each potentially viable point falls within the maximum allowable distance of $D = 0.25$ cm.

2.4 Algorithm for Workspace Calculation

The workspace can be calculated by using an algorithm that steps through all possible platform orientations, and all viable translational values along the z-axis, at a fixed step size. As each step is taken, and the resulting platform points are calculated for that platform orientation, these points are tested to determine if they satisfy all of the mechanical constraints that were discussed in section 2.3, specifically, $\beta_i < 45°$, $\alpha_i < 25°$, $D < 0.25$ cm, $l_i < 41.91$ cm and $l_i > 31.75$ cm, where $i = 1, 2, 3$. The algorithm was implemented using MATLAB®. The rotation matrix uses a fixed pitch-yaw-roll reference that was defined in section 2.2.

The algorithm will first initialize ${}^{P}\overrightarrow{p_i}$, ${}^{B}\overrightarrow{b_i}$, ${}^{B}\vec{q}$ (specifically q_z), ψ (roll), θ (pitch) and $\emptyset$ (yaw). ${}^{P}\overrightarrow{p_i}$, ${}^{B}\overrightarrow{b_i}$ are initialized according to the respective radius of the platform and base, ${}^{B}\vec{q}$ is initialized according to the physical limits of the actuators (a reasonable range for the actuators that are used is 33 cm to 40.5 cm), ψ and θ are initialized to begin at $-\pi/4$ radians, $\emptyset$ is initialized to zero because no rotation is possible about the z-axis.

The algorithm then calculates l_1, l_2, l_3 and ${}^{B}\overrightarrow{P_i}$. Based on these values, the algorithm determines whether the mechanical constraints are satisfied. The most frequent violation was observed to be the violation of D, followed by the maximum actuator length violations. This means that it is a most frequently the physical restrictions on the revolute joint are violated by forcing it to move in a direction that it is unable to move in. The violation counts for each constraint are summarized in table 2.4.1. If they are satisfied, then the ${}^{B}\overrightarrow{P_i}$ points are added to the matrix that stores the viable points. From these points, a vector that is normal to the platform center is calculated by subtracting one set of points from another and then using the cross product of these vectors. This vector is then stored in a matrix so that it can be plotted when the algorithm terminates. The algorithm will then continue to increment the ψ, θ parameters at a step size of 0.1 and q_z at a step size of 0.2 until the algorithm finally terminates at the maximum height value. A flowchart of the process that the algorithm uses to generate the 3-D vector plot is presented in Table 2.4.1.

Table 2.4.1 Summary of Constraint Violations

Constraint	Violation Count	Percentage of Total Violations
$\alpha_i < 25°$	15983	0.16
$\beta_i < 45°$	397653	3.94
$D < 0.25$ cm	4929966	48.72
$l_i < 41.91$ cm or $l_i > 31.75$ cm	4775551	47.18
Total	10119153	100

Figures 2.4.2, 2.4.3 and 2.4.4 show the workspace of a 3-DOF Stewart platform that does not have the misalignment constraints imposed on it. These figures are generated by running the same algorithm that was discussed above, but instead of plotting the 3-D vector that is normal to the center of the platform, the location of each of the ${}^{B}\overrightarrow{p_i}$ points are plotted, along with the ${}^{B}\overrightarrow{b_i}$ points for reference. It can be clearly seen that there is misalignment about the aforementioned axis. Therefore, a constraint on the joint in this direction must be imposed when running the algorithm.

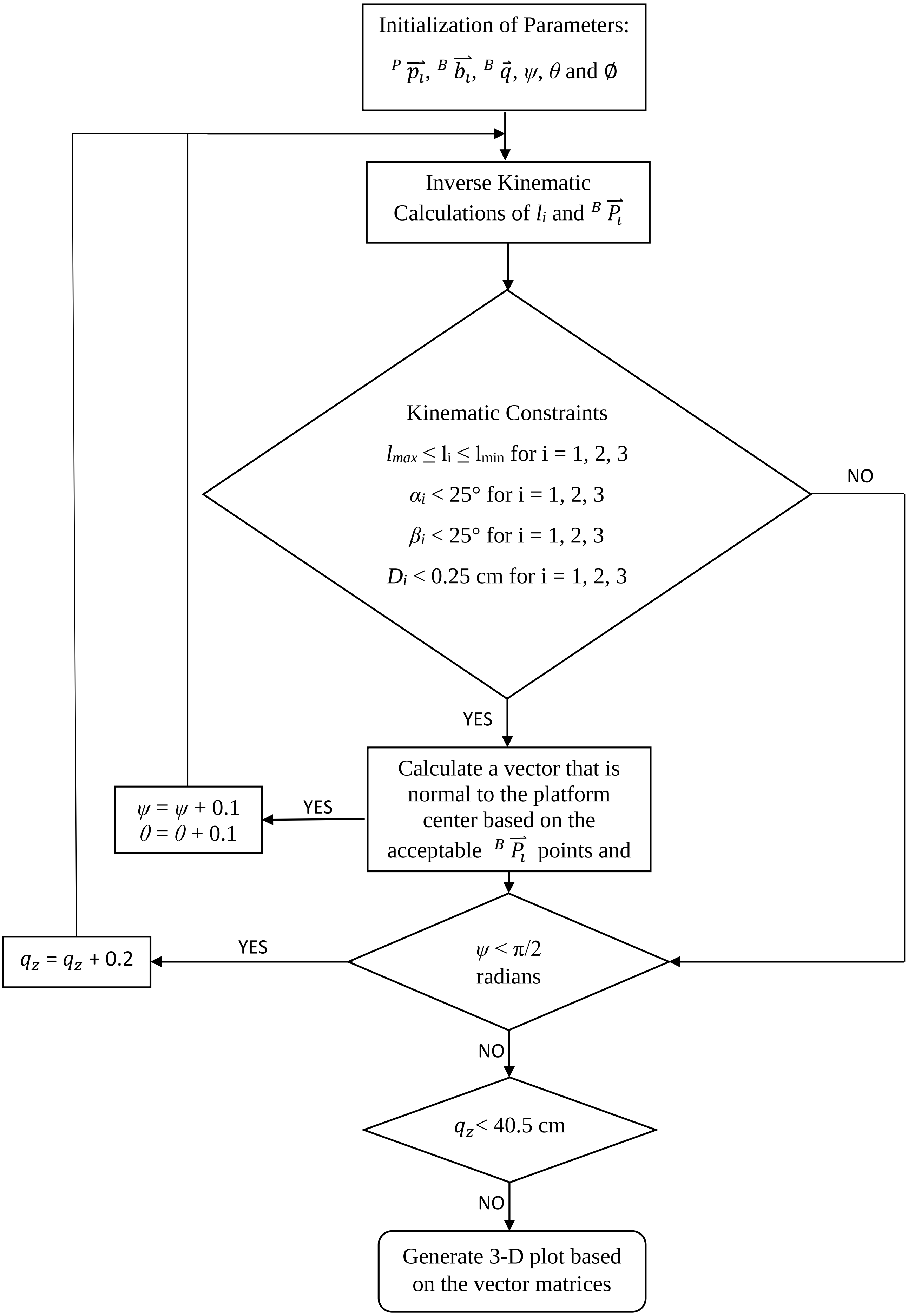
Initialization of Parameters:
$^{P}\overrightarrow{p_\iota}$, $^{B}\overrightarrow{b_\iota}$, $^{B}\vec{q}$, ψ, θ and $\emptyset$
Inverse Kinematic Calculations of l_i and $^{B}\overrightarrow{P_\iota}$
Kinematic Constraints
$l_{max} \leq l_i \leq l_{min}$ for i = 1, 2, 3
$\alpha_i < 25°$ for i = 1, 2, 3
$\beta_i < 25°$ for i = 1, 2, 3
$D_i < 0.25$ cm for i = 1, 2, 3
NO
YES
Calculate a vector that is normal to the platform center based on the acceptable $^{B}\overrightarrow{P_\iota}$ points and
YES
$\psi = \psi + 0.1$
$\theta = \theta + 0.1$
$\psi < \pi/2$ radians
YES
$q_z = q_z + 0.2$
NO
$q_z < 40.5$ cm
NO
Generate 3-D plot based on the vector matrices

Figure 2.4.1 Flowchart for the Algorithm used to generate the 3-D Vector Workspace

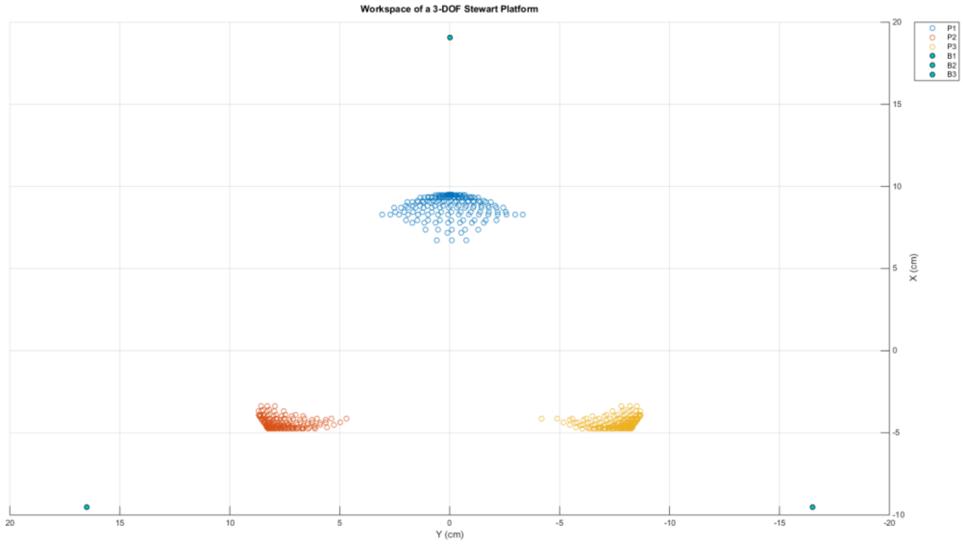

Figure 2.4.2 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (Top View)

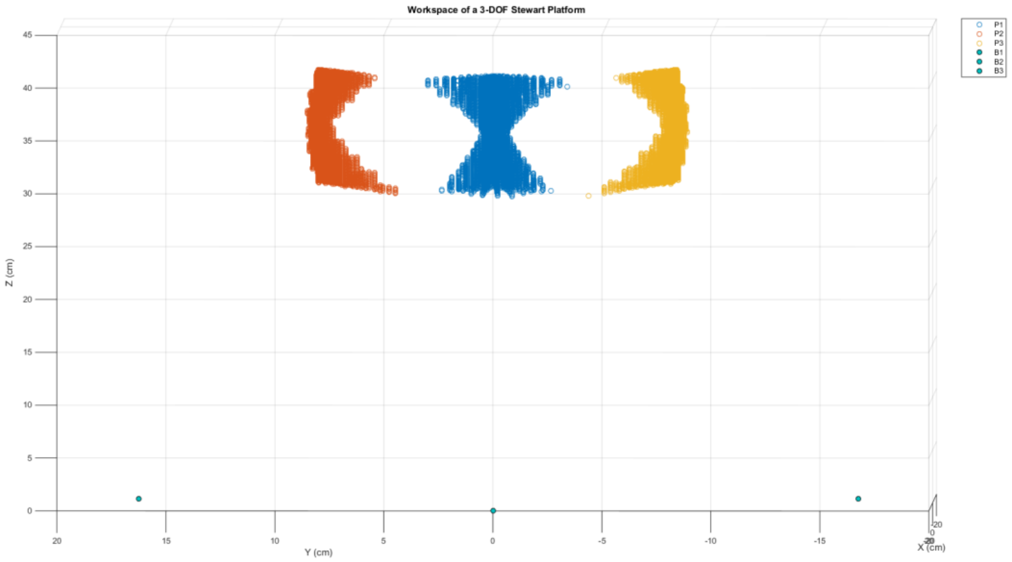

Figure 2.4.3 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (Side View)

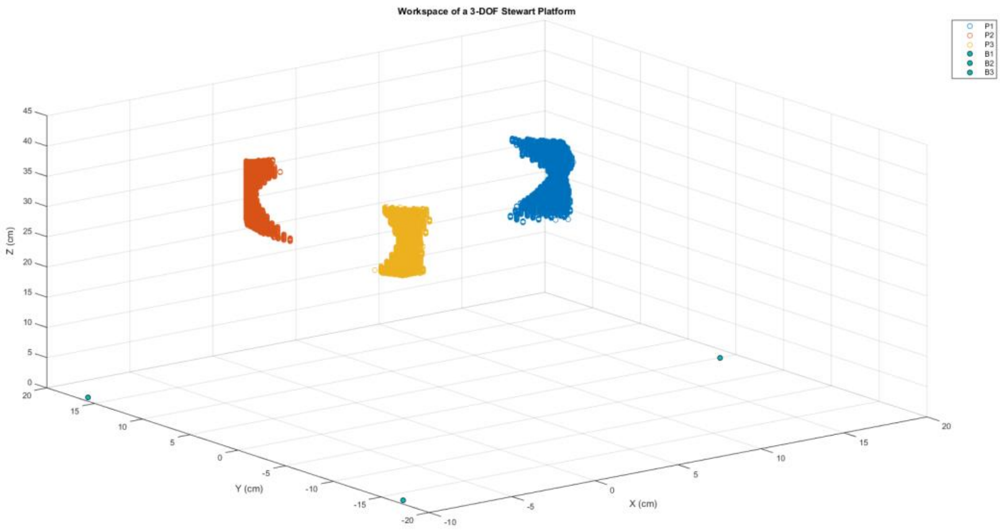

Figure 2.4.4 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points before Misalignment Constraints are Imposed (3-D View)

After the misalignment constraints are imposed, figures 2.4.5, 2.4.6 and 2.4.7 are generated.

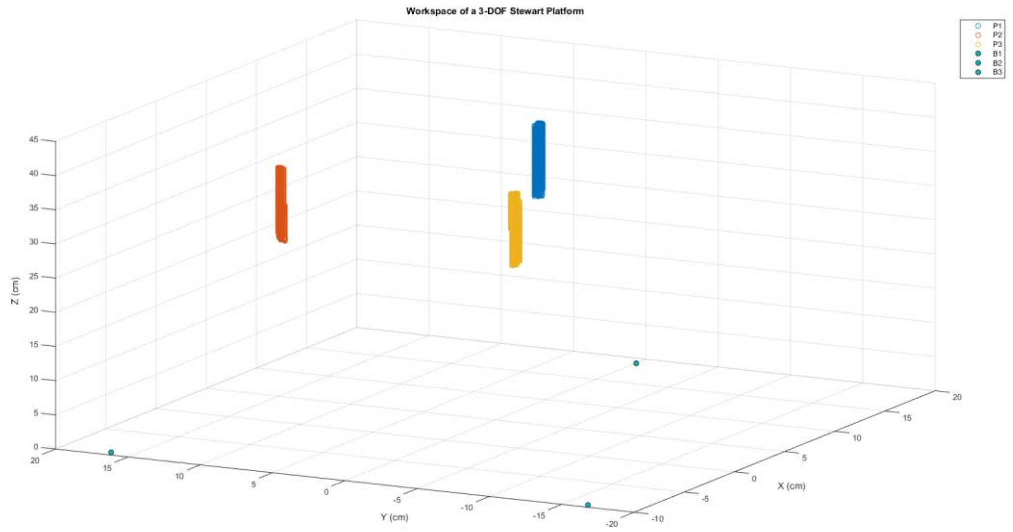

Figure 2.4.5 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (3-D View)

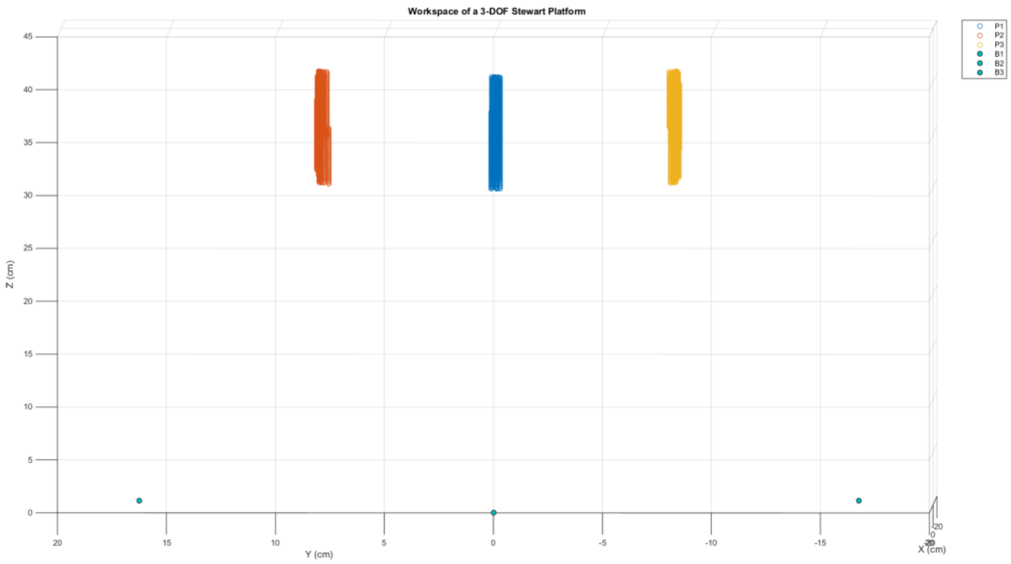

Figure 2.4.6 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (Side View)

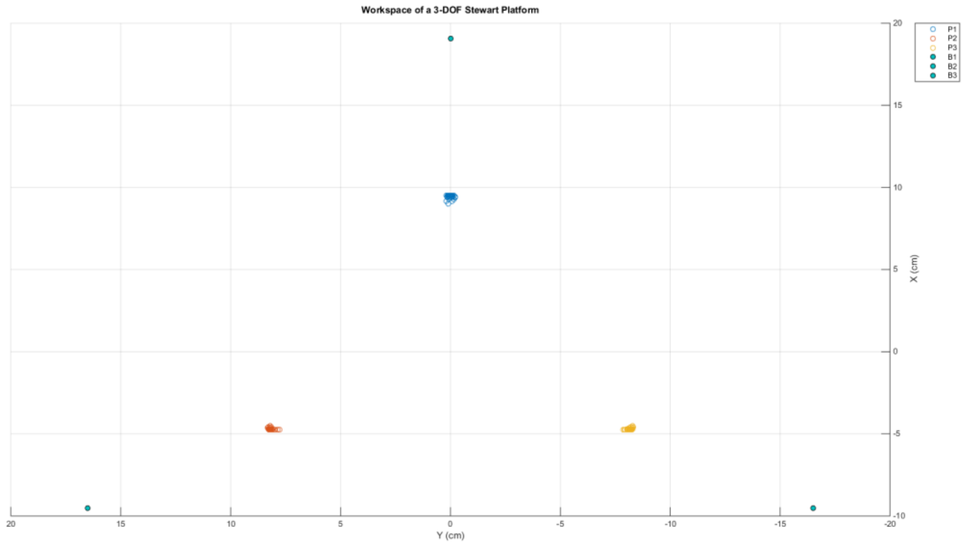

Figure 2.4.7 Workspace of a 3-DOF Stewart Platform Illustrated by all of the Possible Platform Points (Top View)

From figures 2.4.5, 2.4.6 and 2.4.7 we can see that the platform points are now fully constrained so that they do not exceed any of the physical limitations imposed on the robot by the range of motion of the joints and the maximum and minimum lengths of the actuator legs. The implementation of the maximum and minimum actuator leg length constraint can be verified by observing that none of the points are located below 31.75 cm or above 41.91 cm on the z-axis. The constraint on the range of motion of the joints, while more difficult to verify from the graphs alone, can be seen to have been implemented by observing that all of the viable platform points are in an almost straight line that is parallel with the z-axis. This is in stark contrast with the graphs in figures 2.4.2, 2.4.3 and 2.4.4, where it can be seen that some points are located in a position that the robot platform would be incapable of reaching due to the aforementioned limitations on the joint ranges. Figure 2.4.8 to 2.4.13 show the constrained 3-D vector plots generated by the algorithm described above.

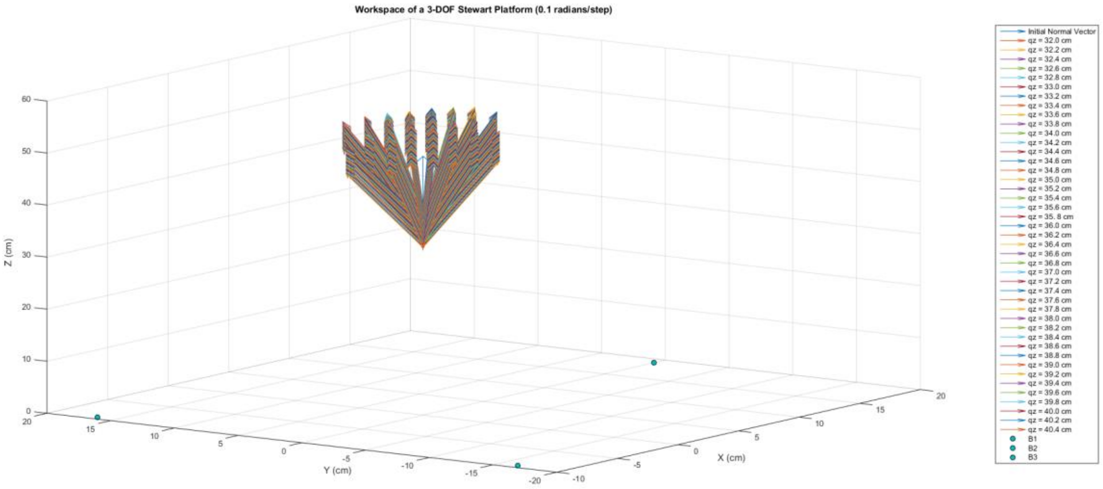

Figure 2.4.8 3-D Workspace of a 3-DOF Stewart Platform Visualized Using Vectors Normal to each Platform Orientation at 0.1 Radians/Step

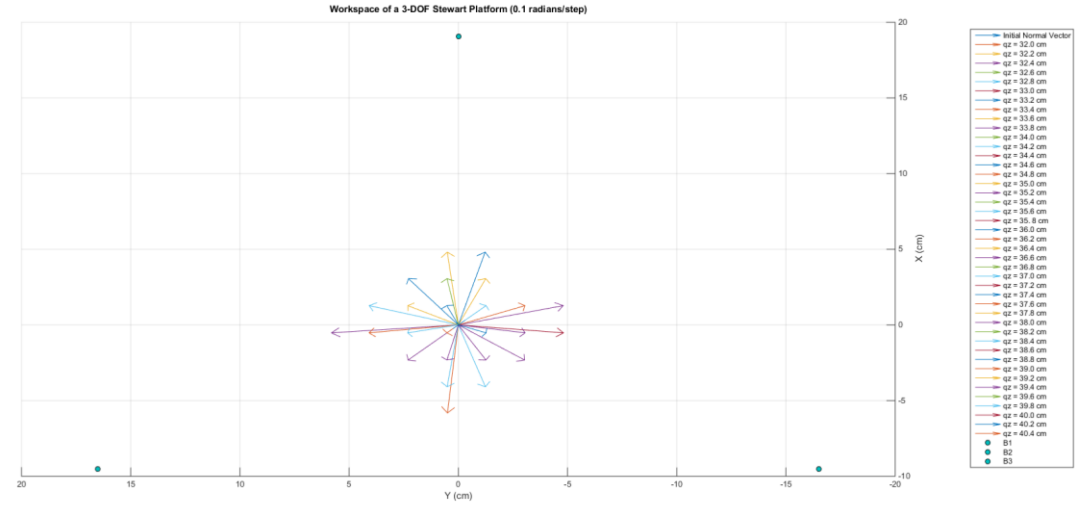

Figure 2.4.9 3-D Workspace of a 3-DOF Stewart Platform Top View at 0.1 Radians/Step (Down Z-Axis)

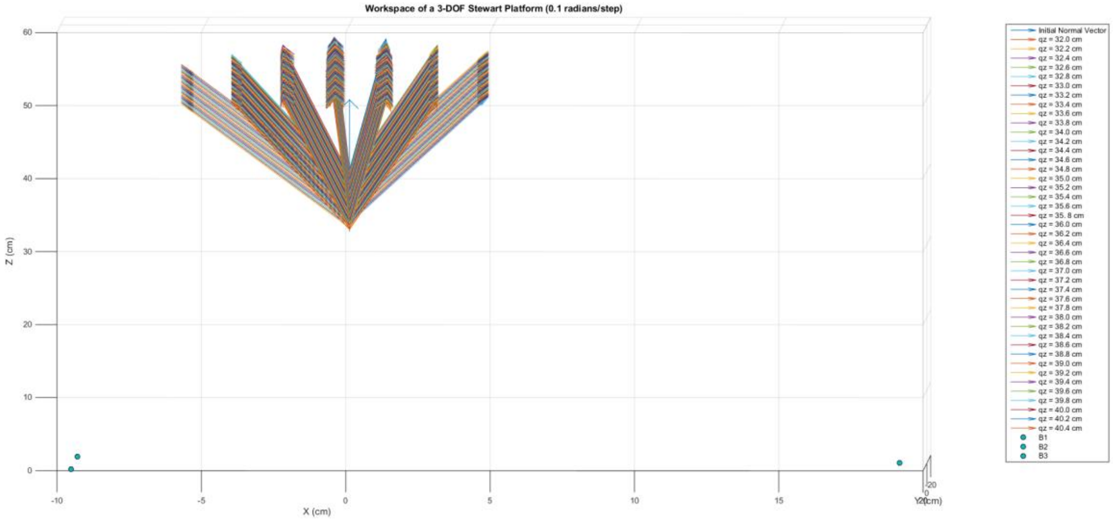

Figure 2.4.10 3-D Workspace of a 3-DOF Stewart Platform Side View at 0.1 Radians/Step (Down Y-Axis)

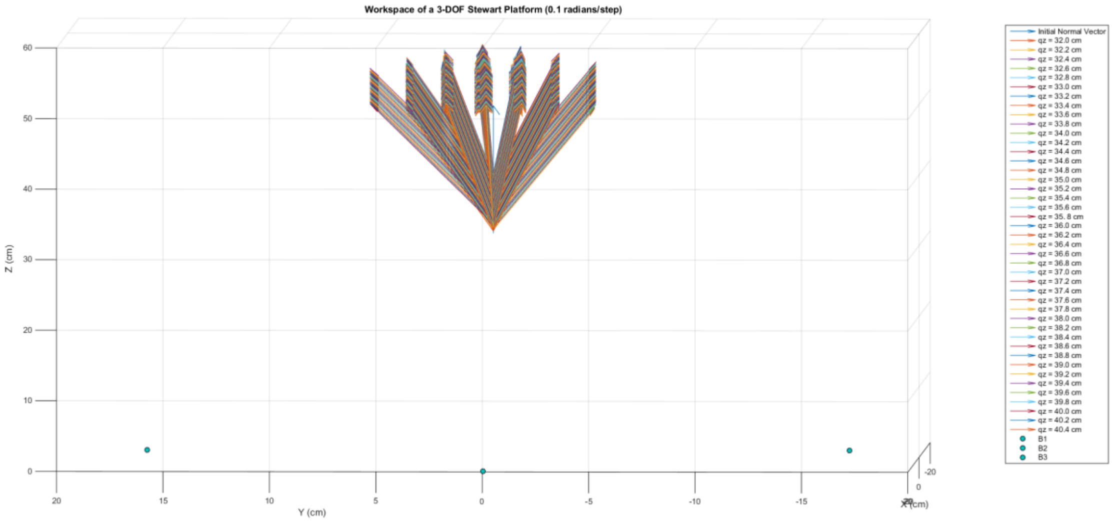

Figure 2.4.11 3-D Workspace of a 3-DOF Stewart Platform Side View at 0.1 Radians/Step (Down X-Axis)

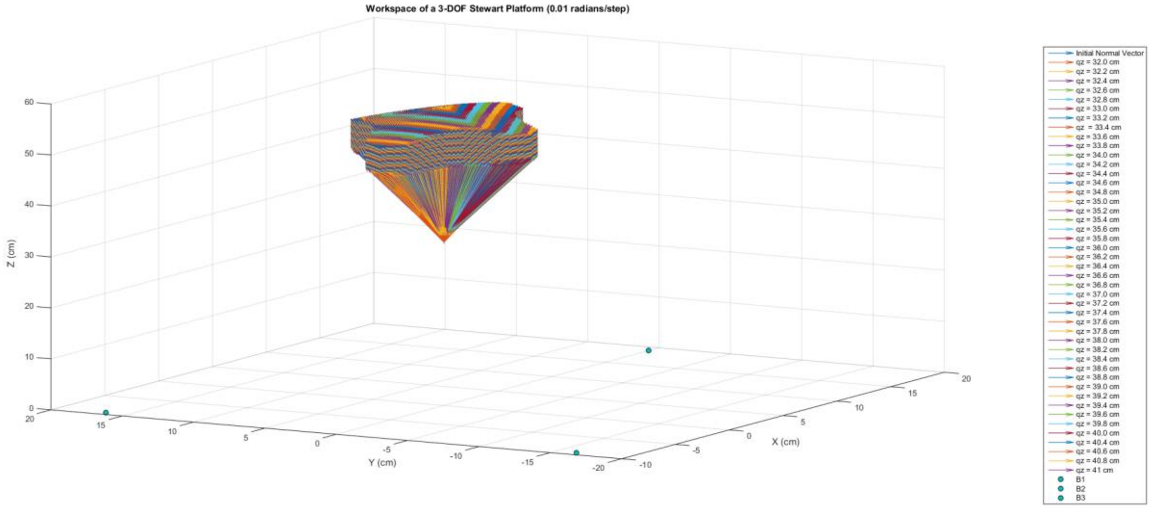

Figure 2.4.12 3-D Workspace of a 3-DOF Stewart Platform Visualized Using Vectors Normal to each Platform Orientation at 0.01 Radians/Step

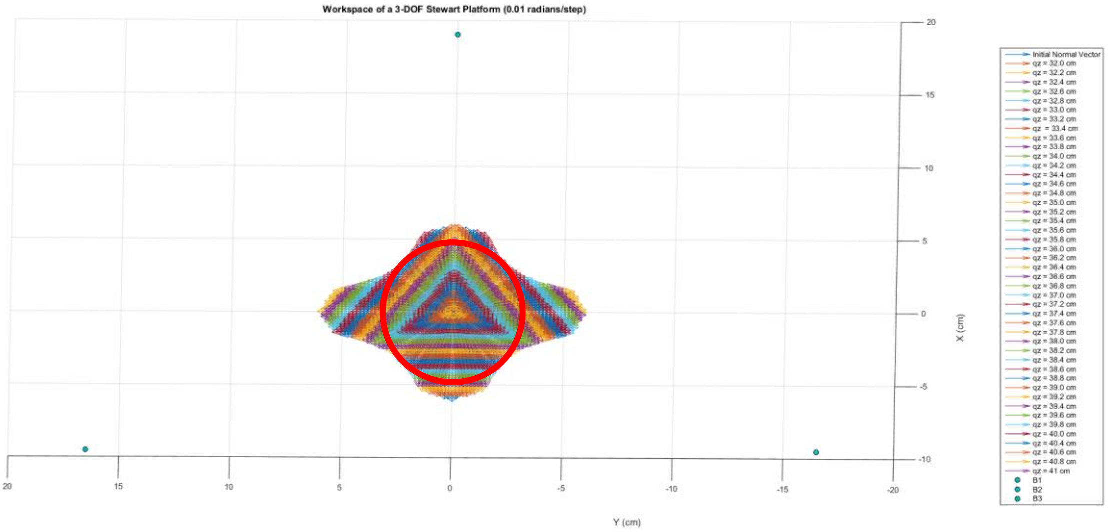

Figure 2.4.13 3-D Workspace of a 3-DOF Stewart Platform Top View at 0.01 Radians/Step (Down Z-Axis)

Figure 2.4.13 is of particular interest because the maximum circular trajectory that the platform's normal vector could follow (designated by the red circle) can be clearly seen and demonstrated on the graph. This concept of a maximum circular trajectory is important because it can be used as an objective function when choosing the dimensions of a robot so that the workspace is optimized. This is especially important for 3-DOF robots because a traditional objective function would seek to maximize the workspace volume. However, since a better measure of the workspace in a 3-DOF robot is based on the how far that platform is able to rotate about the x and y-axes, it can be difficult to find an effective objective function. This concept of using the maximum circular trajectory as an objective function, around which the design parameters of a 3-DOF robot can be optimized, rather than the traditional maximum volume approach, will be further explored in the next chapter.

Chapter 3

Parameter Optimization

3.1 Delta Robot Optimization Introduction

When designing a parallel manipulator, the selection of the parameters of the robot such as the leg lengths, the minimum length of the actuators, the maximum length of the actuators, the diameter of the base and the diameter of the platform can considerably effect the optimality of the workspace as demonstrated by Lou et al. [25]. This problem can be formulated in many different ways. One method proposed by Gosselin and Guillot prescribes the optimal workspace and the algorithm attempts to design a manipulator that has a workspace that is as close to the prescribed workspace as possible [26]. Another method uses an algorithm that optimizes the geometry of the parallel manipulator so that the workspace of the manipulator is maximized [27]. The problem that arises when using these methods is that a manipulator that is designed to have a maximum workspace may be lacking in other desirable characteristics, such as high dexterity or manipulability.

In order to avoid this possible issue, other performance indices must be taken into account. A further refinement of the optimization criteria was done by Stock and Miller, where they used a linear combination of performance metrics that quantified the manipulability and the workspace of the manipulator [28]. This linear combination of the performance metrics was used in the objective function, where weighting coefficients were assigned to the two indices. Thus, the single variable optimization problem becomes a multivariable optimization problem, which will provide a solution that leads to a better designed and more realistic manipulator. However, because it is now a multivariable problem, the objective function could be difficult to solve using standard techniques.

Due to the complexity of the multivariable design problem, Lou et al. suggest using different well known and researched search algorithms to solve the multivariable optimization problem. Depending on the complexity of the objective function, different search algorithms should be used because some will converge quicker on less complicated objective functions, but may not be able to converge when more variables are used to define the objective function. The opposite is true for other algorithms used in the paper. The algorithms presented in the paper were sequential quadratic programming (SQP), controlled random search (CRS), genetic algorithm (GA), differential evolution (DE) and particle swarm optimization (PSO) [29].

It was noted that while it generally took CRS longer to converge to a solution to the objective function for the design of a Stewart platform, when compared to the other algorithms, it was able to converge to the neighbourhood of the global optimum much faster. However, the CRS algorithm was generally slower at reaching the optimum itself, and there were also trial runs where the algorithm was not able to converge at all. In contrast, the other algorithms always converged and were generally faster at converging to the optimum, with the exception of SQP. Out of all of the algorithms tested, PSO was observed to be the most reliable at converging to the global optimum.

As an extension of these results, an algorithm is proposed that takes advantage of the quick initial convergence of the CRS algorithm and the robustness and quick final convergence of the PSO algorithm. In order to initially test the proposed algorithm, we first used a simpler robot design for the optimization test. The robot used for the test is called a Delta robot. This robot design also has 3-DOF, but they are only in the translational directions. It is not possible for a Delta robot to change the orientation of its end effector. Therefore, the number of constraint equations that govern the objective function are reduced. The architecture of the delta robot is pictured in figures 3.1.1 and 3.1.2.

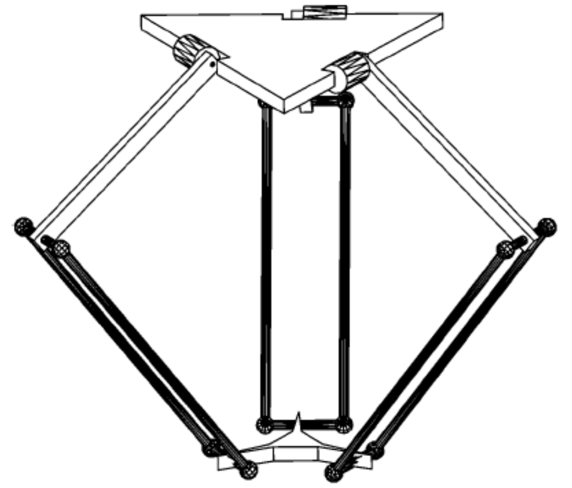

Figure 3.1.1 Architecture of the Delta Robot [30]

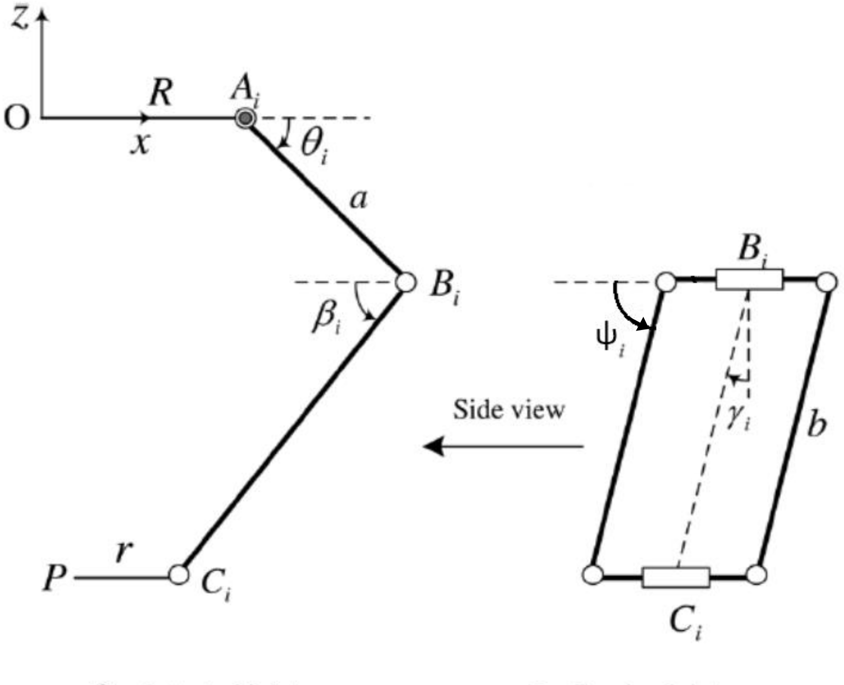

Figure 3.1.2 ith Subchain of the Delta Robot [31]

3.2 Design Problem Formulation

3.2.1 The Objective Function

In order to evaluate the effectiveness of the proposed algorithm, we need to first define the objective function. Since the Delta robot is a purely translational robot, the maximum workspace of the robot can be defined as certain volume. To simplify calculations, the workspace will be assumed to be a regular shape, which in this case will be assumed to be a cube. This assumption is particularly valid in the case of the Delta robot because it is symmetric along the Z-axis. This symmetry further simplifies calculation because the maximum reachable point in Cartesian coordinates in the positive X-Y direction will have the same limit in the negative X-Y direction, thus effectively cutting the number of points, whose viability needs to be determined, in half. This assumption will both increase the speed and decrease the running time of the algorithm.

All real numbers within Cartesian space will be considered as a possible, valid workspace point, therefore $W \in \mathbb{R}^3$, where W is the workspace and $\mathbb{R}$ is all real numbers. Since the workspace is assumed to be cubic, the side length of the cube will be denoted as $2s$. Using the side length of the cube we can calculate the workspace, which is defined in terms of volume. Therefore, the objective function can be defined as:

$$\Phi = (2s)^3 \tag{17}$$

In order to use this algorithm to find a set of design parameters that are not specific to one robot and can be easily scaled to match their intended use, the parameters will be normalized according to the following equation:

$$\sum_{i=1}^{q} \alpha_i = \tau \tag{18}$$

In the case of this design problem, $\tau = 1$ and $q = 4$. It should be noted that i is the individual parameters of α, and not the set of parameters for each arm of the robot. There will be four parameters for each α_i: a, b, d and z_c. Referring to figure 3.1.2, the geometric definitions for a and b are clear, $d = R - r$, and z_c is the center of the cube that defines the maximum workspace of the robot. It was observed by Lou et al. that the maximum workspace volume occurs when d approaches 0 and the parameters a and b are in the neighbourhood of 0.5 [32]. Therefore, the random numbers that are initially generated for these parameters are between 0 and 0.5 for a and b and between 0 and 0.01 for d. The z_c value is randomly generated between 0 and -1 until a valid z_c is found.

3.2.2 The Workspace Constraints

The constraint equations describe the physical limitations of the robot and determine whether a given point is a viable workspace point. There are two types of joints in the Delta robot: passive and actuated, with each type having their own set of constraints. In order to evaluate whether, given end effector position, the robot's structure obeys these constraints, the inverse kinematics of the Delta robot will be used. Again, figure 3.1.2 will be used to clarify what each constraint means in geometric terms. There will be constraints placed on β_i, θ_i and γ_i, where $i = 1, 2, 3$, so that there will be no interference between the arms and the parallelogram rods (the parallelogram is depicted on the right hand side of figure 3.1.2), and to ensure that the range of motion of these joints is not violated. The numerical values that were selected are as follows:

$$-30° \leq \theta_i \leq 100° \tag{19}$$

$$45^\circ \leq \theta_i + \beta_i \leq 180^\circ \tag{20}$$

$$-40^\circ \leq \gamma_i \leq 40^\circ \tag{21}$$

3.2.3 The Manipulability Constraints

Constraints on the manipulability index are introduced so that the manipulator is guaranteed to be able to conduct tasks effectively within the workspace. Typically, the measure for manipulability is the inverse condition of the kinematic Jacobian matrix, which is defined as:

$$\kappa(J) = \frac{\sigma_{min}(J)}{\sigma_{max}(J)}$$

where $\kappa(.)$ is the inverse condition number function for matrices ($\kappa \in [0, 1]$.), and $\sigma_{min}(.)$ and $\sigma_{max}(.)$ are minimal and maximal singular value functions, respectively. Since the Delta robot is only a translational robot, we do not need to take into account the orientation manipulability. Therefore, the position manipulability can be defined as:

$$\dot{Z} = J_Z(\dot{\theta})$$

where $\dot{Z}$ is the linear velocity. Therefore, we can see that $\kappa(J_Z)$ gives a measure for the position manipulability and the following constraint can be applied:

$$\kappa(J_Z) \geq \delta \tag{22}$$

where δ is the lower bound for the positon manipulability, which is a constant that is defined by the user that reflect the design requirements. For our design problem, $\delta = 0.4$.

3.3 Optimization Algorithms

3.3.1 Controlled Random Search

The originally proposed CRS algorithm [33] is a global optimization algorithm that is known for its robustness and flexibility. The algorithm has previously been used by Lou et al. to optimize parallel manipulators, with good results [25, Table 1]. It is, despite improvements on the original algorithm [34], known for its poor final convergence time and its inability to reach an optimum, when dealing with complicated problems. Initially, however, it has a very fast convergence to a value that is within a fairly close neighbourhood of the optimum, even when it cannot reach an optimum value, as shown by the graph in figure 3.3.1. To take advantage of this attribute, the CRS algorithm will terminate after a specified number of generations, or when the value of the objective function is within an acceptable range of the optimum (if this is known). The parameters at this value will be used as a starting point for the PSO algorithm, in order to decrease the time until final convergence.

The CRS algorithm method is to select new points randomly from a normal probability distribution that is centered on the previous best value found by the algorithm, mathematically:

$$\alpha^{(j)} = \alpha^{(j-1)*} + \sigma\zeta, \mathrm{j} \ = \ 1,2, \dots \tag{23}$$

The above equation shows how the *j*th generation point is selected by using the previous generation's best point ($\alpha^{(j-1)*}$) and adding a value to it ($+\ \sigma\zeta$), where:

$$\zeta_i \sim N(0,1), i = 1, \dots, p$$

$$\sigma = diag(\sigma_i, \dots., \sigma_p)$$

In terms of optimization techniques, we can regard ζ as a search direction and σ as a step length. σ is adaptively modified during the search depending on whether the new point was determined to be a successful or a failure. Successful means that an improvement on the objective function occurs and failure means that no improvement occurs. If a trial is successful:

$$\sigma_i = K_1 \Delta\alpha_i, i = 1, \dots, p$$

where $\Delta\alpha_i$ is the distance between the current α_i and the nearest bound of the variable. K_1 is a compression factor that is used to reduce the search interval and keep the next value within the neighbourhood of the previous best value ($0 < K_1 < 1$). After a certain number of failures, typically chosen to be 100, σ is reduced according to [25, pp. 227]:

$$(\sigma_i)_{new} = K_2 (\sigma_i)_{old}, i = 1, \dots, p$$

where $0 < K_2 < 1$. This is done to reduce the step size at a much greater rate than if the trial is successful. This could occur when the optimum is approached for example, and it is done so that the algorithm does not fail to converge to the optimum by taking steps that are too large.

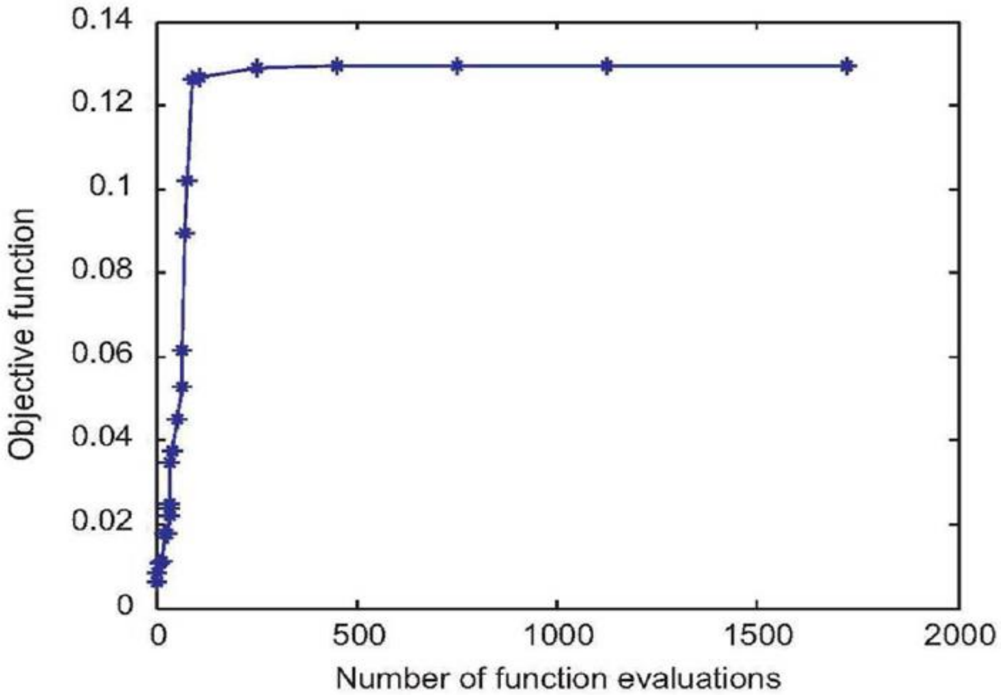

Figure 3.3.1 A Typical Number of Function Evaluations for the CRS Algorithm to Convergence When Determining the Optimal Design of a Delta Robot [25, pp. 232]

Both K_1 and K_2 are chosen by the user and in this case were chosen by Lou et al. to be 1/3 and 1/2, respectively [29, pp. 579]. The proposed algorithm also uses K_1 = 1/3. However, since the proposed algorithm that combines the CRS and PSO algorithms only needs CRS to converge to the neighbourhood of the optimum, which typically occurs within less than 20 generations as demonstrated by Lou et al. (the results of which are shown in the graph in figure 3.3.1), the second part of the algorithm where the step size is reduced after a certain number of failures, is not implemented in the proposed algorithm. This is due to the fact that there would likely be relatively little improvement in the final result compared to the time that it would take to reach that result. Therefore a value for K_2 is not required. Additionally, the global convergence of this algorithm was proven by Solis and Wets [35], thus guaranteeing that the output of the CRS algorithm is within the neighbourhood of the global optimum.

3.3.2 Particle Swarm Optimization

Particle swarm optimization was first introduced in 1995 [36]. The basic concept is that there is a group or swarm of particles in the parameter space ([0, 1], in the case of this problem), each with their own position (x_i) and velocity (v_i). Typically, initial positions and velocities are typically randomly generated within certain bounds, and the better the particles are initially spread out across the n-dimensional space, the more likely that they will find the global optimum. In the case of the proposed CRS and PSO combination algorithm, the initial positions will be generated, within certain bounds, around the point that was returned by the CRS algorithm. Using this approach, the initial few steps that the particles would make if they were randomly generated will be already completed, and all of the particles will already be within a reasonable distance of the optimum. The velocities will still be randomly generated, within certain bounds. During each iteration, x_i and v_i are updated according to the following equations:

$$v_i = v_{i-1} + c_1(rand)(p_i - x_i) + c_2(Rand)(p_g - x_i) \tag{24}$$

$$x_i = x_{i-1} + v_i \tag{25}$$

x_i and v_i have already been defined, c_1 and c_2 are learning factors (c_1 = 0.5 and c_2 = 1.25 are the values that are used by Lou et al. and will be used for this simulation [29, pp. 579]), rand and Rand are two separate random number generators that return a value between 0 and 1 (uniform distribution), p_i is the personal best of the position of the particle (as evaluated by the objective function), and p_g is the global best position of a particle (as evaluated by the objective function). By using the personal best position of each particle, as well as the global best position, PSO should converge to a global optimum without converging to a local optimum instead.

3.4 Design Problem

3.4.1 Optimal Design of the Delta Robot

Mathematically, the design problem can be summarized as follows:

$$\max_{\alpha} \Phi = (2r)^3$$

$$subject\ to\ -30° \leq \theta_i \leq 100°$$

$$45° \leq \theta_i + \beta_i \leq 180°$$

$$-40° \leq \gamma_i \leq 40°, \text{ where } i = 1, 2, 3$$

$$\kappa\big(J(X, \theta, \alpha)\big) \geq 0.4$$

$$a + b + d = 1$$

$$a, b, d \in [0,1]$$

$$z_c \in [-1,0]$$

3.4.2 Computation

As previously discussed, the CRS algorithm will be used to find a set of parameters (α) that are in the neighbourhood of the global optimum. Since the PSO algorithm works by using the best results found throughout the swarm, there must be some variability in the initial swarm particle values. Therefore, we cannot simply use the point that results from the CRS algorithm (this will be referred to as α_{CRS}) as the initial value for each of the swarm particles (these will be referred to as α_{PSO}). On the other hand, if we set too few of the initial swarm particles to α_{CRS}, it will not affect the speed of the PSO algorithm by an appreciable amount and there will have been no reason to find α_{CRS}. Therefore, a compromise must be made when deciding how many of the initial swarm particles will be equal to α_{CRS}. After experimenting with how many of the initial swarm particles are being set to α_{CRS}, a value of five (out of twenty particles) was experimentally found to yield the best results. The remaining fifteen α_{PSO}'s contain values for *a* that are randomly generated as discussed in section 3.2.1.

In order to increase the speed of the algorithm, we will determine the largest cubic workspace by first generating a valid point along the z-axis, z_c. Thus, in Cartesian coordinates, the first point will be (0, 0, z_c). Once a valid point has been generated, this point will be used as the center of the cubic workspace. As previously discussed the objective function is defined by $(2s)^3$, where $2s$ is the side length of the cube. Instead of searching the entire possible workspace, the validity of each possible cubic workspace is verified by using s to calculate the coordinates of eight vertices of each cube, and each of these points is separately verified using the constraints.

However, in order to determine if the vertex coordinates are valid, the values of each of the constraints needs to be determined for each x, y and z for each vertex of the cube. To do this we will make use of the inverse kinematics, which relates the Cartesian coordinates x, y and z to θ_i. β_i, γ_i. Referring to figure 3.1.2, the loop closure equation in the {O} reference frame is:

$$\overrightarrow{\boldsymbol{OP}} + \overrightarrow{\boldsymbol{PC_\iota}} = \overrightarrow{\boldsymbol{OA_\iota}} + \overrightarrow{\boldsymbol{A_\iota B_\iota}} + \overrightarrow{\boldsymbol{B_\iota C_\iota}} \tag{26}$$

From this equation, Pierrot derives the relationship between x, y, z and θ_i [37]:

$$X_i cos\theta_i + z\sin\theta_i = Q_i \tag{27}$$

Where:

$$X_i = x + d$$

$$Q_i = (X_i^2 + a + y^2 + z^2 - b^2)/2a$$

If $k = cos\theta$, and using the substitution $sin\theta = \sqrt{1 - cos^2\theta}$, equation (27) can be put into the form:

$$Ak^2 + Bk + C = 0 \tag{28}$$

Where:

$$A = X_i^2 + z^2$$

$$B = -2Q_i$$

$$C = Q_i - z^2$$

Quadratic formula can now be used to solve for the roots, which will give us k, from which the θ_i value for each leg can obtained using $\cos^{-1} k$.

In order to determine if the second constraint ($45° \leq \theta_i + \beta_i \leq 180°$) is satisfied we use geometry of the robot to find a formula that yields the angle β_i:

$$\beta_i = \cos^{-1}((acos\theta_i + d)/b) \tag{29}$$

Again, using the loop closure equation, Lopez demonstrates the relationship between x, y and ψ_i, as shown in figure 3.1.2. The resulting equation is as follows [38]:

$$xsin\emptyset_i + ycos\emptyset_i = bcos\psi_i \tag{30}$$

$\emptyset_i = (i-1)(2\pi/3)$ for i = 1, 2, 3. Then, the relationship between ψ_i and γ_i is simply:

$$\gamma_i = \psi_i - 90° \tag{31}$$

Finally a relationship between J and x, y, z, θ, α needs to be found so that it can be determined whether the $\left(J(x, y, z, \theta, \alpha)\right) \geq 0.4$ constraint is satisfied. Equation (32) relates the actuated joint variables θ_i for i = 1, 2, 3 to the position of the robot platform point P in the reference frame {O}, given by x, y, z.

$$(x - (d + acos\theta_i)cos\emptyset_i)^2 + (y - (d + acos\theta_i)sin\emptyset_i)^2 + (z + asin\theta_i)^2 - b^2 = 0 \tag{32}$$

For i = 1, 2, 3, and $\emptyset_i = (i-1)(2\pi/3)$ for i = 1, 2, 3. Equation (32) can be differentiated with respect to time (t) to obtain:

$$J_x = \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix} \tag{32}$$

where $v_i = [x - (d + acos\theta_i)cos\varphi_i, y - (d + acos\theta_i)sin\emptyset_i, z + asin\theta_i]^T, i = 1, .., 3.$ Using the relationship $J_x\dot{X} = J_\theta\dot{\theta}$, where:

$$J_\theta = -diag\{h_1, h_2, h_3\} \tag{33}$$

Since $J = J_X^{-1}J_\theta$, we can now use the MATLAB® command svds to determine the maximum and minimum singular value of the matrix J and, from these values, use the relationship:

$$\kappa(J) = \frac{\sigma_{min}(J)}{\sigma_{max}(J)}$$

to determine if $\kappa\big(J(X, \theta, \alpha)\big) \geq 0.4$.

As each possible cubic workspace is generated, the eight vertices of the cube are checked to see if their *x*, *y* and *z* coordinates satisfy the above constraints. In other words, the algorithm determines if the robot platform point *P* is capable of reaching each of the eight vertices. The workspace is considered to be valid if all of the constraints are satisfied at all eight vertices. To determine the size of the cubic workspace for each α that is generated, a for loop is used to decrease *r* from a sufficiently large value (in the case of the delta robot, this value was chosen to be 1) at a step size of 0.01. When all eight vertices of the cube are determined to be valid, the algorithm stops the for loop and uses the last, and therefore largest, value of r to calculate the volume of the workspace.

After the workspace of the robot is determined using the dimensions given by the elements of α, the algorithm then updates each α according to the search method being used (i.e. CRS or

PSO) being used, as outlined in section 3.3 of this chapter. The proposed algorithm will terminate when one of the particles in the swarm reaches the optimum value of 0.125.

3.5 Delta Robot Optimization Results and Discussion

The above algorithms were implemented and run in the MATLAB® (R2014b) environment. The trials were run on a personal computer with an Intel® Core™ i7-4710HQ CPU @ 2.5 GHz and 8 GB of RAM.

To determine the effectiveness of the proposed algorithm, two separate tests were run. The first test was a timed run of the optimization of the parameters using only a PSO algorithm and the second test was a timed run of the optimization of the parameters using the proposed combined CRS and PSO algorithm. Five trials were run and the results of these tests are presented in tables 3.5.1 and 3.5.2.

Table 3.5.1 PSO only Algorithm Results

PSO					
Trial	**Optimum**	**PSO Time**	**a**	**b**	**d**
1	0.125	251.639	0.3884	0.6036	0.0079
2	0.125	331.065	0.3892	0.6045	0.0063
3	0.125	331.916	0.4190	0.5747	0.0064
4	0.125	330.987	0.4176	0.5749	0.0076
5	0.125	172.752	0.3504	0.6478	0.0018
Average	0.125	283.672	0.3929	0.6011	0.0060

Table 3.5.2 Combined CRS and PSO Algorithm Results

CRS and PSO							
Trial	**Optimum**	**CRS Time**	**PSO Time**	**Total Time**	**a**	**b**	**d**
1	0.125	82.611	248.587	331.198	0.4405	0.5524	0.0072
2	0.125	87.320	177.390	264.719	0.4232	0.5671	0.0096
3	0.125	85.388	169.372	254.760	0.4354	0.5548	0.0098
4	0.125	82.598	250.556	333.154	0.3876	0.6042	0.0081
5	0.125	85.110	0	85.110	0.4506	0.5493	0.0010
Average	0.125	84.605	211.478	253.788	0.4275	0.5656	0.0071

*the "Time" is the computation time in seconds, the optimum and the parameters are in units

For the PSO only algorithm, the computation times ranged from 172.752 s to 331.916 s, with an average of 283.672 s, and a standard deviation of 63.469 s. The proposed algorithm had total times ranging from 85.110 s to 331.198 s, with an average of 253.788 s, and a standard deviation of 90.4035 s. The CRS portion of the proposed algorithm had times ranging from 82.598 s to 82.611 s, with an average of 84.605 s, and a standard deviation of 1.802 s. The PSO portion of the proposed algorithm had times ranging from 0.000 s to 250.556 s, with an average of 211.478 s (excluding trial #5) or 169.183 s (including trial #5), and a standard deviation of 38.205 (excluding trial #5) or 91.233 (including trial #5).

The results presented in tables 3.5.1 and 3.5.2 show that combining the CRS algorithm with the PSO algorithm leads to a faster overall convergence time for the proposed algorithm when compared to the PSO only algorithm. On average the total time it took the proposed algorithm to reach the optimum of 0.125 was 29.88 s faster than the PSO only algorithm. Furthermore, if we compare only the PSO running time of the proposed algorithm to the running time of PSO only algorithm, we can see that on average the PSO running time of the proposed algorithm was 114.49 s faster than the PSO only algorithm. In trial #5 of the proposed algorithm, the CRS algorithm reached the optimum within 10 generations, so the PSO algorithm converged immediately, which is an unlikely scenario, but it is clearly a possibility and will significantly reduce the overall computation time.

When analyzing the final parameter values for the PSO only algorithm, we can see that a has a range of values between 0.3504 and 0.4176, with an average of 0.3929, and a standard deviation of 0.0250, b has a range of values between 0.5749 and 0.6478, with an average of 0.6011, and a standard deviation of 0.0268, and d has a range of values between 0.0018 and 0.0079, with an average of 0.006, and a standard deviation of 0.0022. The proposed algorithm has final parameter values for a that range between 0.3876 and 0.4506, with an average of 0.4275, and a standard deviation of 0.0218, the values for b range between 0.5493 and 0.6042, with an average of 0.5655, and a standard deviation of 0.0202, the values for c range between 0.0010 and 0.0098, with an average of 0.0071, and a standard deviation of 0.0071.

While there is some discrepancy between the final parameter values of the proposed algorithm and the PSO only algorithm, the standard deviation for all the a values for all ten trials run in both algorithms is 0.02913, for b it is 0.0296, and for d it is 0.0028. These standard deviations should be considered acceptable when optimizing a problem that is highly non-linear such as this. Additionally, the optimum and parameters that the proposed and PSO only algorithms found were reasonably similar to those found by Lou et al [25, Table 1]. Any differences in the values found by the algorithms used in this project and those found by Lou et al. can be explained by slight differences in the step sizes of the workspace, in the objective function, in the convergence termination criteria, etc.

Due to the nature of the problem and the algorithms used, the convergence time of both the proposed algorithm and the PSO only algorithm is highly dependent on the suitability of the points that are initially generated at random. This means that the closer the initially generated points are to the optimum, the faster the convergence of the algorithm to the optimum. This is another reason why the proposed algorithm is more effective than the PSO only algorithm. Due to the ability of the CRS portion of the proposed algorithm to converge to the neighbourhood of the global optimum, even when the initial points are far from the optimum, the PSO portion of the proposed algorithm will rarely have a particle swarm that will not allow it to converge to the global optimum.

It should be noted that the trials shown were the successful trials, but there were a number of trials for both algorithms where the optimum could not be reached successfully. It was observed that generally the PSO only algorithm failed to reach the optimum more times than the proposed algorithm

3.6 3-DOF Stewart Platform Optimization Introduction

Now that the proposed algorithm has been found to improve the convergence time when compared to the PSO only algorithm, it will now be applied to the problem of optimizing the 3-DOF Stewart platform. Since the function and purpose of a Stewart platform is much different than that of a Delta robot, the parameters that will be used for optimization, as well as the optimization problem itself, will be significantly different. Because the purpose of our Stewart robot is focused around spraying, which is more related to the orientation rather than the translation of the platform, the robot design will be optimized to maximize the orientation workspace rather than the translation workspace.

3.7 Design Problem Formulation

3.7.1 The Objective Function

The objective function for this problem is based around the concept of a maximum circular trajectory that the robot is capable of following with the normal vector to its top platform. This concept is illustrated in the previous chapter, and specifically figure 2.4.13. In this figure, a maximum circular trajectory is highlighted by a red circle. This trajectory is of particular importance to our application because it indicates the largest diameter of pipe that the robot is capable of spraying. Therefore, for our application, we will seek to maximize the diameter of this circle by optimizing the dimensions of the robot.

In contrast to the Delta robot, the Stewart platform has fewer parameters to maximize. Since there is no “elbow” in the legs of a Stewart platform, and each leg is actuated in a linear motion, as opposed to the rotational actuation of a Delta robot, it is difficult to optimize the lengths of the Stewart platform legs because the lengths of the actuator are constantly changing as the orientation of the platform changes. Therefore, there is no real optimum length for each leg because each leg length is dictated by the orientation of the platform.

Based on the figures in section 2.4, it can be seen that the number of possible orientations, and therefore the orientation workspace, is significantly smaller when the leg lengths are near their minimum or maximum stroke. Conversely, the number of possible orientations is significantly higher when the leg lengths are near the middle of their stroke. As a result of this observation, the height of the center of the top platform will be fixed at the middle of the actuators stroke during the optimization process. By doing this, the other parameters of the Stewart platform can be maximized without having the added computational cost of finding the exact height of the robot that yields the maximum orientation workspace.

Since the leg lengths do not need to be optimized, that leaves only two other parameters: the radius of the top plate, R, and the radius of the bottom plate, r. In the same way as the method for optimizing the Delta robot, these parameters are grouped together in an array γ, such that:

$$\gamma = [R\ r]$$

These parameters will be optimized through the use of the maximum circular trajectory objective function that was described above, where r_{circle} is the radius of the largest circle that the . Therefore, the objective function will be defined as:

$$\Phi = \mathrm{r}_{circle} \tag{34}$$

3.7.2 The Workspace Constraints

The constraints that describe the physical limitations of the Stewart platform, as well as how these constraints are calculated, were presented and discussed previously in section 2.3. The numerical values that were selected are re-stated here:

$$\alpha_i \leq 25° \tag{35}$$

$$\beta_i \leq 45° \tag{36}$$

$$31.75 \leq l_i \leq 41.91\ cm \tag{37}$$

$$-0.45° \leq \Delta \leq 0.45°,\ where\ i = 1, 2, 3 \tag{38}$$

Two additional constraints are placed on *r* and *R*:

$$R \ \in [5{,}12]\ cm$$

$$r \in [18{,}25]\ cm$$

These constraints are placed on these parameters in order to reflect the size constraints that would be imposed on a robot design that would be used for spraying pipelines or manholes, this is particularly relevant for the maximum radius constraint of the bottom plate. The minimum radius constraint on the top platform is introduced to allow enough area on the top plate to mount a hose. Lastly, the maximum radius constraint on the top plate and the minimum radius constraint on the bottom plate are imposed so that no parameters are generated that would create an unrealistically dimensioned Stewart platform (if the top plate had the same. or a similar radius. as the bottom plate).

3.8 Design Problem

3.8.1 Optimal Design of the Stewart Platform

Mathematically, the design problem can be summarized as follows:

$$\max_{\gamma} \Phi = \mathrm{r}_{circle}$$

$$subject\ to\ \alpha_i \leq 25°$$

$$\beta_i \leq 45°$$

$$31.75 \leq l_i \leq 41.91\ cm$$

$$-0.45° \leq \Delta \leq 0.45°,\ where\ i = 1, 2, 3$$

$$R\ \in [5,12]\ cm$$

$$r \in [18,25]\ cm$$

3.8.2 Computation

In order to determine the maximum circular trajectory for each α, a circular trajectory will be generated using a for loop with a radius (r_{circle}) that starts at 30 cm and ends at 0 cm at a step size of -0.1. For each circle that is generated, eight points on the circle that are 45° apart are stored in a cell array. Using the inverse kinematics that were described in section 2.2 and a nested for loop, the platform points (as shown in figure 2.2.1) are calculated for every combination of theta and psi from –π/3 to π/3 at a step size of 0.025 radians. If these platform points satisfy the all of the constraints, a normal vector that is located in the center of the top plate is generated. Then, if the generated normal vectors intersect each point stored in the cell array for the current configuration of the robot given by α, the current r_{circle} trajectory is able to be followed by the robot and this r_{circle} value is the largest trajectory that the Stewart platform, dimensioned by the current α values, can follow.

3.9 Stewart Platform Optimization Results and Discussion

Again, the above algorithms were implemented and run in the MATLAB® (R2014b) environment. The trials were run on a personal computer with an Intel® Core™ i7-4710HQ CPU @ 2.5 GHz and 8 GB of RAM.

The tests that were used to compare the convergence times between the PSO only and proposed algorithm for the Delta robot optimization were again used to test the respective convergence times of these algorithms, except this time they were tested on the optimization of the 3-DOF Stewart platform. The final termination criteria for both algorithms was the optimum of 26.7 cm was reached. The CRS portion of the proposed algorithm terminated after 10 generations or after the optimum of 26.7 had been reached, whichever came first. The results of the 5 trials for each algorithm are summarized in tables 3.9.1 and 3.9.2.

Table 3.9.1 PSO only Algorithm Results

PSO				
Trial	**Optimum**	**PSO Time**	**R**	**r**
1	26.7	2146.704	5.4076	18.8201
2	26.7	900.242	5.4990	18.3038
3	26.7	823.75	5.4462	18.9655
4	26.7	765.502	5.2405	19.0685

5	26.7	1786.273	5.4051	18.9892
Average	26.7	1284.494	5.4000	18.6294

Table 3.9.2 Combined CRS and PSO Algorithm Results

CRS and PSO						
Trial	**Optimum**	**CRS Time**	**PSO Time**	**Total Time**	**R**	**r**
1	26.7	557.025	747.858	1304.883	5.0927	18.8052
2	26.7	260.674	717.547	978.221	5.0364	18.0495
3	26.7	1254.792	705.721	1960.513	5.6269	18.2005
4	26.7	438.687	794.547	1278.234	5.1869	18.6000
5	26.7	710.46	0.000	710.460	5.2052	17.9788
Average	26.7	653.328	741.418	1246.462	5.2296	18.3268

*the "Time" is the computation time in seconds, the optimum and the parameters are in cm

For the PSO only algorithm, the computation times ranged from 765.502 s to 2146.704 s, with an average of 1284.494 s, and a standard deviation of 596.996 s. The proposed algorithm had a total time ranging from 710.460 s to 1960.513 s, with an average total time of 1246.4622 s, and a standard deviation of 417.981 s. The CRS portion of the proposed algorithm had times ranging from 260.674 s to 1254.792 s, with an average time of 653.328 s, and a standard deviation of 333.842 s. The PSO portion of the proposed algorithm had times ranging from 0.000 s to 794.547 s, with an average time of 741.418 (excluding trial #5) or 593.135 (including trial #5) and a standard deviation of 34.308 (excluding trial #5) or 298.157 (including trial #5).

The results show that the total convergence time for the proposed algorithm is, on average, 38.032 s faster than the PSO only algorithm. If the average time of the PSO portion of the proposed algorithm (this average excludes trial #5 where the CRS portion was able to converge to the optimum) is compared to the average time of the PSO only algorithm, the average time of PSO portion of the proposed algorithm is 543.076 s faster than the PSO only algorithm. This is a significant improvement in the convergence time for the PSO optimization.

The final convergence parameters had some slight variations between trials and between algorithms, which can be attributed to the inherent randomness of the search algorithms. For the PSO only algorithm, the final R values ranged from 5.2504 cm to 5.4990 cm, with an average of 5.4000 cm, and a standard deviation of 0.0865 cm and the final r values ranged from 18.3038 cm to 19.0685 cm, with an average of 18.6294 cm, and a standard deviation of 0.3740 cm. For the proposed algorithm, the final R values ranged from 5.0364 cm to 5.6269 cm, with an average of 5.2296 cm, and a standard deviation of 0.2080 cm, and the final r values ranged from 17.9788 cm to 18.8052 cm, with an average of 18.3268 cm, and standard deviation of 0.3217 cm. The standard deviation for all the R values for all ten trials run in both algorithms is 0.1806, for r it is 0.3802, which should be considered acceptable in a highly nonlinear optimization problem such as this.

To take advantage of this increase in the speed of the convergence of the PSO algorithm after it receives an α that is in the neighbourhood of the optimum, the user would only need to run the CRS algorithm at most a few times to find parameters that are in the neighbourhood of the optimum. Then the user could choose the best one and then use those parameters to populate part of the initial particle swarm, instead of running the CRS algorithm and then the PSO algorithm every trial. Since this method has been shown on average to increase the speed of convergence of the PSO algorithm, it would certainly lead to a substantial increase in the speed of convergence of the PSO algorithm if the CRS algorithm were only used a few times in order to find the parameters in the neighbourhood of the optimum, rather than every single trial, which is unnecessary.

Chapter 4

Robot Design

4.1 Actuator Types

While it is possible create a linear actuator by combining a screw, gearbox and motor, it was decided that in the interest of time and guaranteed functionality, that a pre-designed and pre-constructed actuator is the better option in the case of our robot. When selecting which actuator to use as the legs for a robot, many different factors need to be considered. Firstly, the size of the actuator is among the most important. When considering a pre-constructed actuator, each specific application will likely need a different size of actuator, depending on how much linear force the actuator will need to produce, how much power is will be available to supply to the actuator, the size constraints of the space the robot will be working in and the size constraints so that the actuator will fit into the rest of the robot design.

Since, in the case our robot, the three actuators only need support the weight of the platform (include calculation for the weight of the platform and base) during the prototyping phase, the linear force that is required of each actuator is not substantial, and, therefore, was not a major design consideration. If the prototype moved to the next stage, where it would be used for a spraying application, this aspect of actuator selection would gain importance. The reason for this is that there will be extra weight of the hose and force of the spray coming out of the hose acting on the actuators, which need to be accounted for.

Since the spraying process is not well defined for this problem, it is difficult to say how much of an impact these additions to the robot structure will have. It can be said that the amount of force generated by the spraying process, and how much the hose and spray head would weigh, is largely dependent on the density of the liquid being sprayed. Intuitively, the denser the liquid, the more force required to spray it. This will require a higher pressure within the hose in order to move the liquid and this in turn will require a thicker, and likely more heavy, hose. In addition to that, the way in which the liquid is sprayed, as a cone or as a jet, for example, can also have an effect on the forces experienced by the platform [39].

Another major consideration when selecting an actuator is the type, with the three main choices being hydraulic, pneumatic or electric. When deciding which type to use, the advantages and disadvantages of each need to be evaluated so that the actuator properties suit the application. The main advantage of hydraulic actuators is that, due to the near incompressibility of the liquid used to generate the mechanical motion, they can exert a large amount of force, but their speed and acceleration is slow. It is also difficult for hydraulic actuators to halt at a certain position, due to the inability to maintain an exact amount of fluid flow from the pump to the actuator over a period of time.

Another concern about hydraulic actuators is that the hydraulic fluid could potentially leak out of the cylinder, due to damage or deterioration of the actuator. This would, in the case of a manhole or pipe spraying process, contaminate a clean water pipe, and potentially effect the curing of the spray coating. The inverse problem is also a concern, where water or other contaminates could leak into the actuator. Theoretically, all actuators could be sealed against contaminants coming into contact with the actuator by wrapping them in a sealing bag. However, for the simple reason that such a bag could easily become damaged and allow contaminants to come into contact with the actuator, the resistance of each actuator type, as well as the effect that the contaminants would have on the normal operation of the actuator, should be researched.

The hydraulic actuator is arguably the most difficult to seal against this type of penetration. Furthermore, any presence of contaminants in a hydraulic system is extremely detrimental to both performance and the life expectancy of the actuator [40]. Combined, the presence of dirt and water account for 70 to 80 percent of failures in hydraulic systems. Even before the contaminates cause a failure of the system, they can cause a myriad of performance problems with the hydraulic fluid such as increased compressibility, loss of lubricity and fluid oxidation [41]. The effects that water levels in the hydraulic fluid on the lifespan of bearings gives us an idea of the catastrophic damage that can be caused. The relative lifespan of the bearings is plotted against the water concentration in the hydraulic fluid is shown in figure 4.1.1.

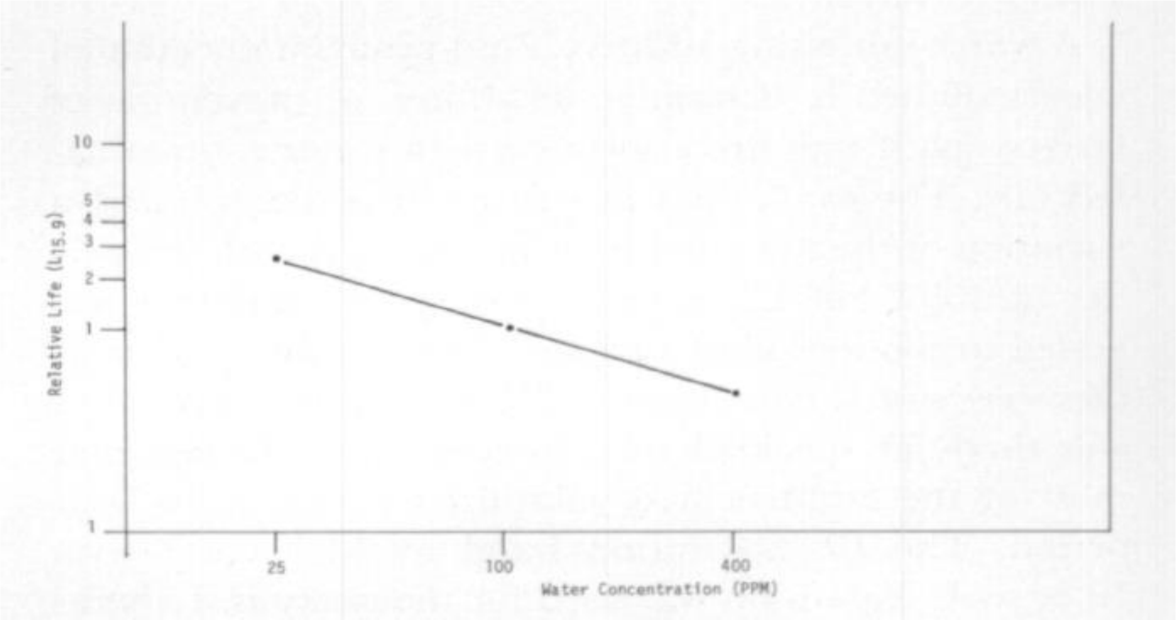

Figure 4.1.1 Relative Lifespan of Bearings vs. Water Concentration in the Hydraulic Fluid [42]

In terms of accurately controlling a hydraulic actuator, they typically lack a built-in position measuring device which would give the user feedback about the position of the leg at any given point in time. Because of this, a separate position measuring device would need to be selected and integrated into the system. Even with a position measuring device, hydraulic actuators have a lower accuracy, which can be attributed to the varying behaviour of the fluid under different operating condition (for example, temperature). Finally, a fairly large pumping system would be required in order to power three hydraulic actuators, which is impractical for a mobile application such as pipe spraying [43].

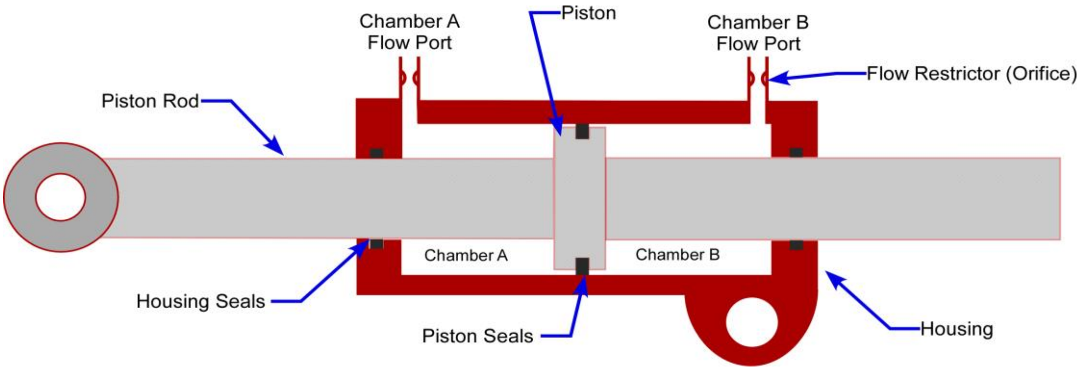

Figure 2.1.2 Hydraulic Actuator Diagram

The second possible actuator type is pneumatic. The advantages of this type of actuator are that it can quickly respond to a start or a stop command, due to the fact that the power source does not need to be stored in reserve for operation and that they have a high working speed and acceleration. A property among its disadvantages is that it is difficult to control the position of a pneumatic actuator, which is partly due to the fact that air can be compressed, especially under high loads, and partly due to the unpredictable behaviour of the air. The unpredictable behaviour of the air stems from the fact that the environmental conditions, such as temperature, pressure and humidity, can greatly affect the performance of the actuator, much the same as the fluid in a hydraulic actuator. Because of the highly variable properties of air, this type of actuator is usually used in valve applications, where the accuracy requirement is lower. Another disadvantage is the inability of the actuator to support high loads, which again is due to the compressibility of air.

The last five disadvantages for pneumatic actuators are the same as hydraulic actuators: they would typically require a separate position measuring device, which would then need to be integrated into the system, there is the possibility of the cylinder leaking from damage or deterioration (although it would not be as disastrous in terms of contaminating the pipeline spraying environment as it was in the case of hydraulic actuator leaking), there is also the possibility of environmental contaminates leaking into the cylinder (with similar consequences to the lifespan and performance of the actuator) [45], they have difficulty halting at a certain positon, and they would also require a pumping system, which would need to include an air compressor. Again, the additional pumping system requirement is not practical for a mobile system [46].

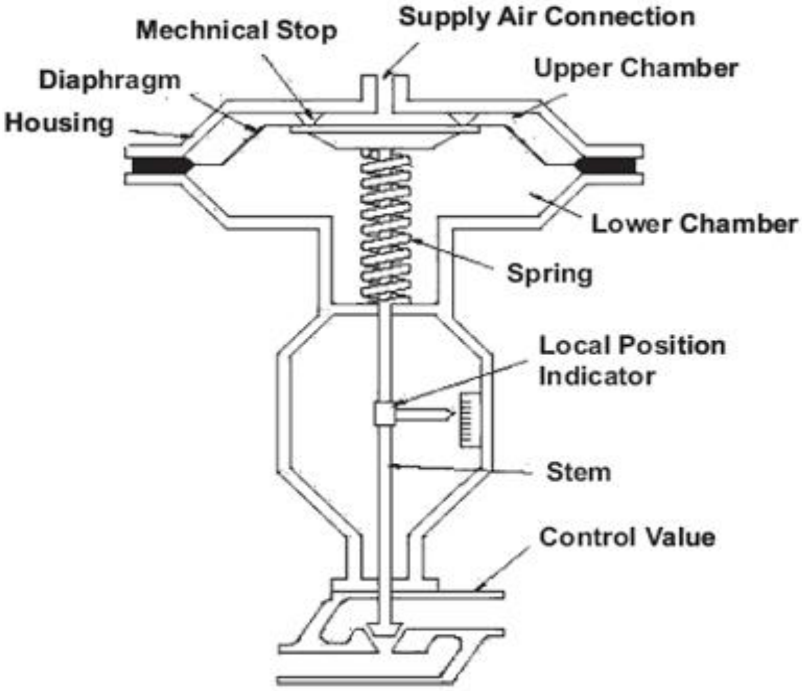

Figure 4.1.3 Pneumatic Actuator Diagram [47]

The last type of actuator that can be considered is an electric actuator. The main advantage of this type of actuator are that it can be easily and precisely controlled using PWM waveforms, a pre-motor driver and a built-in position feedback device (such as a potentiometer, a feature which was lacking in the other actuator types). They do not require a large amount of power to operate, when compared to similar sized hydraulic and pneumatic actuators, mostly due to the fact that there is no pumping system or compressor required to run the actuator. Additionally, electric actuators are able to hold a constant position, which is difficult for the other types of actuators to do, even when they lose power. They have the ability to support high loads and produce a lot of force, a property which can be further enhanced by using a gearbox with a high ratio.

The disadvantages of this type of actuator are that they generally will not be able to produce as much linear force as a comparably sized hydraulic actuator, and they will not be able to accelerate as quickly as a pneumatic actuator. They have an additional disadvantage that there will be a collection of wires that will need to run from the power supply and the operating computer (unless it is battery powered and remote controlled, which probably is not feasible given the amount of power required to run the actuators) to the robot. Another concern that is related to this disadvantage is that in a wet working environment, such as a water pipe, the electric actuator has the potential to short circuit if any wires are not properly protected and insulated against water that may be present in the pipe. This is a concern that is not shared by the pneumatic actuator or the hydraulic actuator, unless their position feedback device is connected to a power supply using wires and not battery powered, which is likely to be the case [48].

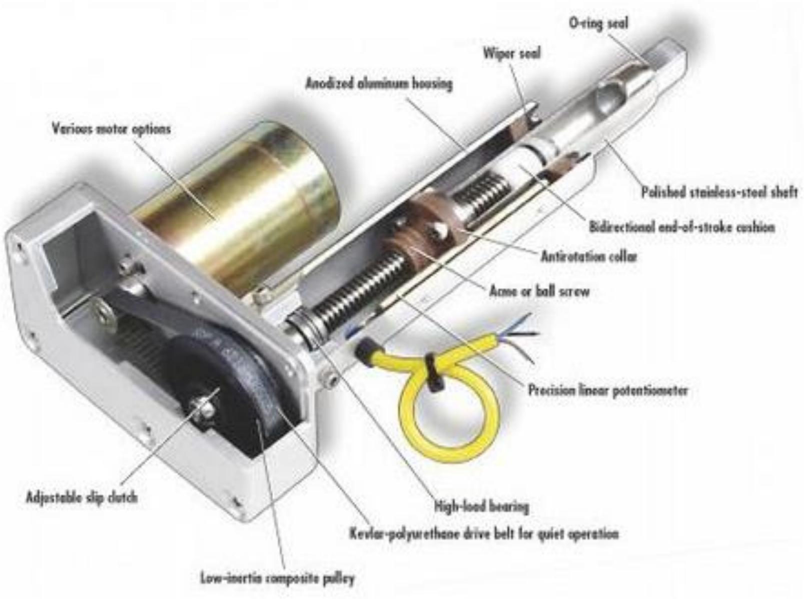
O-ring seal
Wiper seal
Anodized aluminum housing
Various motor options
Polished stainless-steel shaft
Bidirectional end-of-stroke cushion
Antirotation collar
Acme or ball screw
Precision linear potentiometer
Adjustable slip clutch
High-load bearing
Kevlar-polyurethane drive belt for quiet operation
Low-inertia composite pulley

Figure 4.1.4 Electric Actuator Diagram [49]

4.2 Actuator Selection

The next step in the selection process is to choose the type of actuator that is most suited to this particular design problem, taking into account the respective advantages and disadvantages of each type, as detailed in the preceding paragraphs. These advantages and disadvantage for each actuator type are summarized in table 4.2.1.

Table 4.2.1 Advantages and Disadvantages for each Actuator Type

Actuator Type	**Advantages**	**Disadvantages**
Hydraulic	• Able to exert a large amount of force and support high loads	• Low speed • Low acceleration • Difficult to control • Difficult to interface with the microcontroller • Position measuring device typically not built-in • Possibility of leaking hydraulic fluid • External pumping system required
Pneumatic	• Able to quickly respond to commands • High speed • High acceleration	• Difficult to control • Difficult to interface with the microcontroller • Position measuring device typically not built-in • Possibility of leaking air • Inability to support high loads and produce large forces • External pumping system required

Electric	• Accurate control • Easy to interface with the microcontroller • Built-in position measuring device • No external pumping system required • Able to support high loads and exert large forces	• Less force than a hydraulic actuator • Less speed and acceleration than pneumatic actuator • Wires running from power supply and computer to robot • Possibility of a short circuit in a wet environment

The main design considerations when choosing an actuator type for this application are summarized as follows:

- The actuator size (when integrated into the overall robot design) must allow it to fit into a design envelope that is defined as a 38.1 cm diameter cylinder, with a height of 50.8 cm (this design envelope was chosen so that the robot would fit into a typical size manhole).
- The actuator must have a high speed and load rating to weight ratio.
- The actuator must not leak fluid (hydraulic fluid or air) to contaminate the working environment.
- The actuator must have a position feedback device built-in or it must be possible to attach one to the actuator.
- The actuator must be able to be sealed/resistant against environmental contaminants (e.g. dirt, water, etc.).
- The actuator, its power supply and its pumping apparatus (if required) must be portable.
- It must be possible to accurately control the actuator with a microcontroller.

Table 4.2.2 summarizes the main design considerations that need to be addressed when choosing the actuator type, and how well each actuator type fulfills each criteria. Each design criteria is weighted in importance from one to seven, and each actuator type is given a rating of one, two, or three, correspond to how well it fulfills that design criteria. A rating of one means that it fulfills the criteria very well, a rating of two indicates that it somewhat fulfills the criteria or it is possible for that type to fulfill the criteria, but there are better options, and a rating of three indicates that it is not able to fulfill the criteria.

Table 4.2.2 Design Criteria and Ratings for each Actuator Type

		Actuator Type		
Importance	**Design Criteria**	**Hydraulic**	**Pneumatic**	**Electric**
1	Size	1	1	1
2	Portable power supply and pumping apparatus	3	2	1
3	Accurate control	2	3	1
4	Built-in feedback	3	3	1
5	High speed/load to weight ratio	2	1	1
6	Sealable/resistant to environmental contaminations	1	1	1
7	No fluid/air leak	3	3	1

After considering the advantages and disadvantages of each prospective actuator type, as well as the main design considerations, it is clear that the type of actuator most suited to this application is the electric actuator. This decision is mainly based on the lack of portability of hydraulic and pneumatic actuators, due to the external pumping system required. Also of high importance when deciding which actuator type to choose was the low accuracy of a hydraulic and a pneumatic actuator when compared an electric actuator, as well as the lack of a built-in position feedback device on a hydraulic or pneumatic actuator. Also, implementing a hydraulic or pneumatic pumping system is a very involved, potentially messy and expensive undertaking which was not viable for this project. Now that it has been decided that an electric actuator is the type that is most suited to this application, it is important to have a general overview of all of the components that make up the robot: the actuators, the DC motor controllers, the microcontroller and the power supplies. Each of these different parts of the electrical system need to be researched, evaluated and procured, in order to have the entire robot design function as efficiently and accurately as possible. Figure 4.2.1 shows a block diagram of the entire system, as well as the specific function of each component.

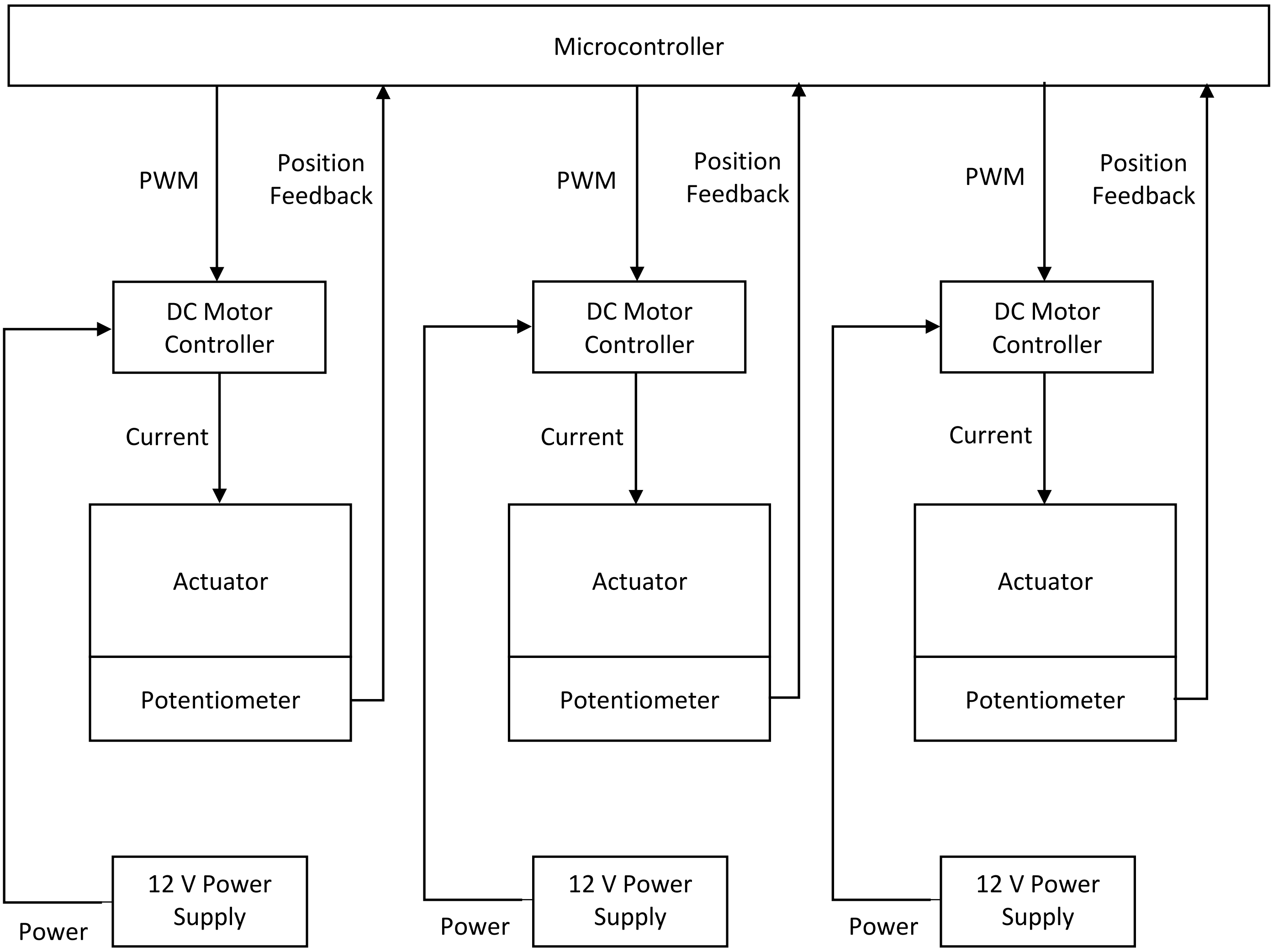

Figure 4.2.1 Overview of Electrical System Components for a 3-DOF Stewart Platform

Starting from the top of figure 4.2.1, the microcontroller is the part of the robot that is programmed to control the actuators through a PWM wave output and a position feedback input from the potentiometers attached to each actuator. The PWM wave is used by the DC motor controller to output a current to each of the actuators. This current controls the direction and the speed of the actuator arm. The 12 V power supplies provide the power to each of the DC motor controllers so that they can output the appropriate current. Although it is not shown in the figure, the microcontroller is powered and programmed through a USB cable connected to a computer. This USB cable connection is not shown because the microcontroller can run without it after it has been programmed, as long as it has an adequate power supply. Section 4.3 deals with the selection and design of the electrical system, which includes a discussion of the selection criteria for the specific actuator model, the DC motor controller and the power supply. Section 4.4 details the criteria and selection process for the microcontroller.

4.3 Electrical System Design

Once it was determined that the electric actuator was the type of actuator most suited to this application, one needed to be chosen with the proper dimensions (including an adequate stroke length), sufficient speed and load capabilities, a built-in position feedback device and a voltage and power supply requirement that could be provided for in a laboratory setting (preferably less than 24 Volts and 100 Watts). After thoroughly researching the available electric actuators and consulting with the university machine shop, the actuator that satisfied all of the above requirements was the Thomson Electrak 1 (SP12-09A-04). The dimensions of this actuator can be seen in figure 4.3.1 and a 3-D Solidworks model of the actuator can be seen in figure 4.3.2. The stroke length (S) for the SP12-09A-04 model is 10.16 cm and the retracted length (A) is 31.75 cm

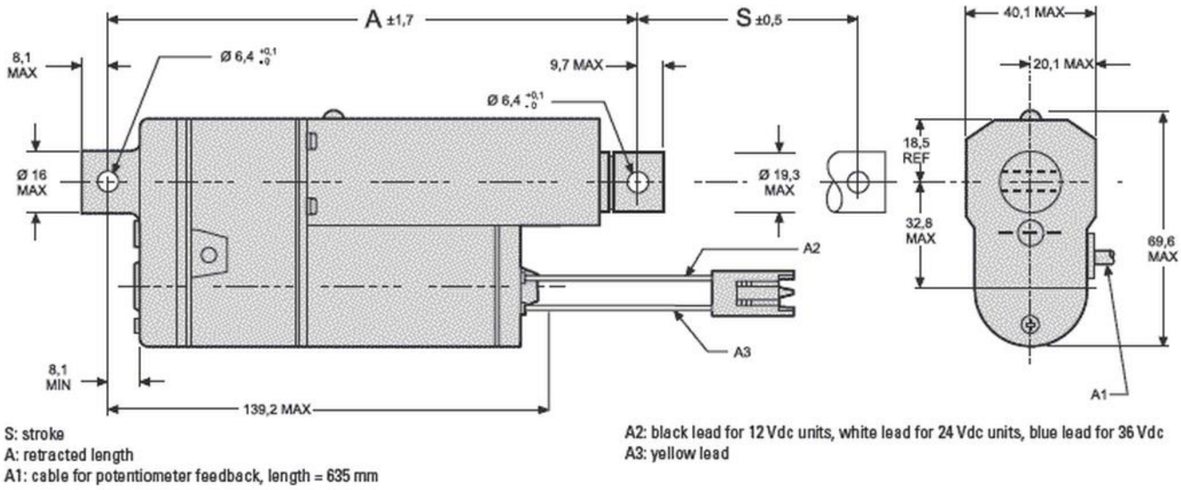

Figure 4.3.1 Thomson Electrak 1 Schematic [50]

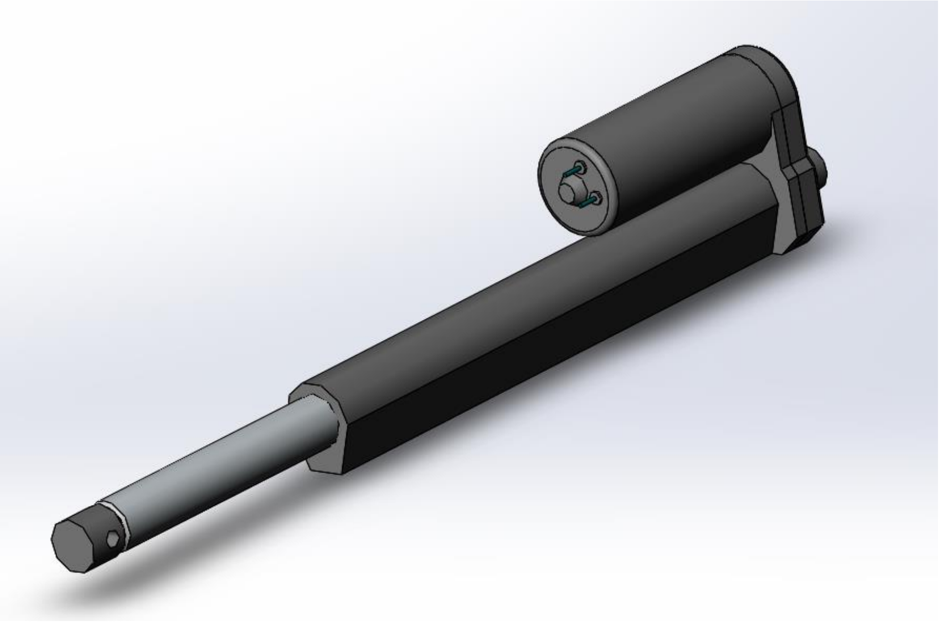

Figure 4.3.2 Thomson Electrak 1 3-D Solidworks Model

With a retracted length of 31.75 cm and a maximum height design requirement of 50.8 cm, this leaves at least 19.05 cm of additional height before the maximum is reached, and in reality more because the actuator is not mounted vertically in the robot design. This difference is available to allow for other parts of the robot, such as the platform and base plates. Additionally, the anchor points of the actuator to the platform and base are located a small distance below or above the plates, respectively, in order to allow for movement of the actuator without the actuator contacting the plates. This distance between the actuator anchor points and the plates means that there is more height added to the overall robot design that needs to be taken into account. In addition to that, the maximum stroke length of this actuator is 10.16 cm, which, by running the simulations described in chapter two of this thesis, was found to be a sufficient stroke length to allow the platform to be oriented at all the angles which would allow it to spray in the circular trajectory that was specified in the same chapter. Therefore, this actuator size is ideal for the design envelope that was specified and meets the size requirement without difficulty.

One additional feature about the construction of this actuator that was not necessarily a major part of the selection criteria, but is nonetheless beneficial to the design, is that there are anchor holes on the bottom of the actuator housing and the top of the actuator arm. This feature certainly saved time when designing and constructing the robot because it provided a simple solution to finding a way to connect the actuator to the platform and the base plates. There was no need to design or construct special anchors or connector because they were already included in the actuator design itself. These anchor holes also provide a very sturdy connection that prevents against any sliding or backlash of the actuator, which is especially important during the operation of the robot to ensure as high a degree of accuracy as possible.

The Thomson Electrak 1 SP12-09A-04 requires a 12 VDC power and has a maximum current draw of 5.6 A. This means that the maximum amount of power that could be required of the power supply is 67.2 W, although during normal operation the power requirement would be less than this. Although there are many power supplies available that meet the voltage and power requirements, the MW126 power supply (shown in figure 4.3.3 and 4.3.4) was selected for this project, and will be used as an example to demonstrate the portability of lower voltage power supplies. The reason it was chosen is because it was readily available from the university electronics shop, and because it is easy to connect and incorporate into the robot design (three identical power supplies were incorporated into the robot design – one for each actuator). This power supply has a voltage rating of 13.8 VDC and a current rating of 6 A, both of which satisfy the requirements to power the actuator. The dimensions of the power supply are 165 x 122 x 236mm (W x H x D) and the weight of the power supply is 9 lbs. Due to the relatively light weight and compact dimensions of this power supply, which mean that it is very portable, thus satisfying the second design requirement.

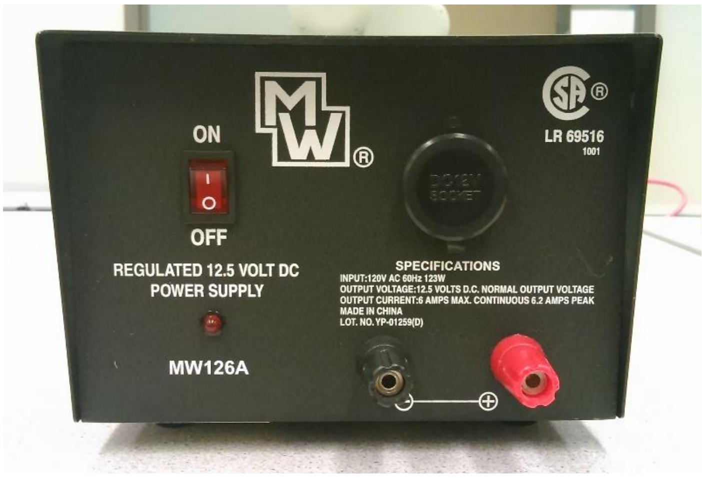

Figure 4.3.3 MW126 Power Supply (Front View)

Figure 4.3.4 MW126 Power Supply (Top/Side View)

The third design requirement is related to the fourth design requirement because the actuator cannot be accurately controlled without a position sensing feedback device. These requirements could be considered to be of equal importance, however, the built-in feedback device requirement is ranked lower because it is possible to add a feedback device into the robot design, in the case where the actuator does not have one built in. Nevertheless, choosing an actuator that lacked a built-in feedback device added unnecessary additional steps to the design process. These steps would consist of researching, procuring and subsequently integrating the feedback device into the system. For this reason the decision was made to select the SP12-09A-04 rather than the S12-09A-04 version of the Thomson Electrak 1, which includes a built-in potentiometer to receive a length feedback from the actuator.

Although the positon feedback device is required to accurately control the actuator, a brushed DC motor controller is also a necessity, in order to easily and precisely control the actuator motor at variable speeds. After researching different types of DC motor controllers such as the Jaguar, the Victor 888, and the Talon SR, as well as multiple consultations with the university electronics shop, the Talon SR motor controller (pictured in figure 4.3.5) was selected due to compact size (6.93 cm x 4.82 cm x 2.92), small deadband (4%) and linear performance curves. This DC motor controller, when used in conjunction with the built-in positon feedback potentiometer, and a microcontroller (which will be discussed in the next section), provide the hardware basis for an accurately controllable electric actuator.

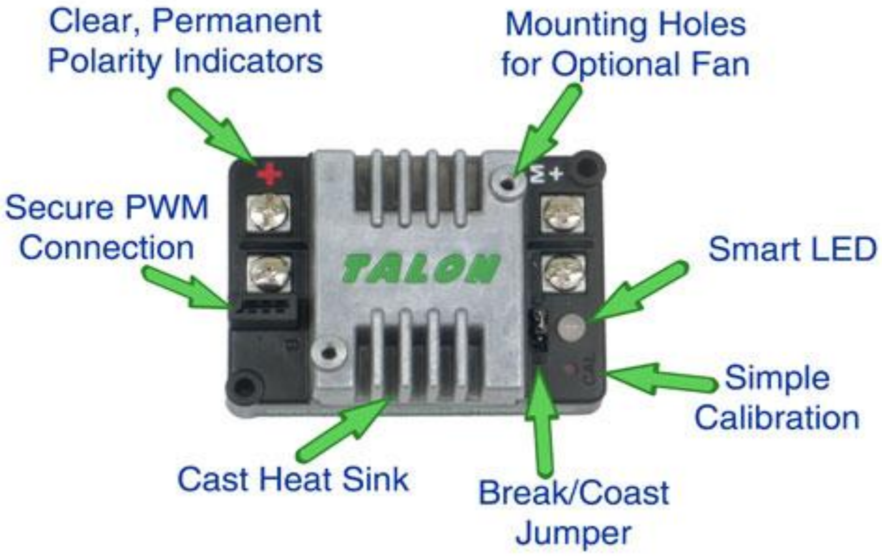

Figure 4.3.5 AndyMark Talon SR Speed Controller [51]

The graphs shown in figures 4.3.6 and 4.3.7 were the result of Motor Controller testing at the Whirlpool Motor Lab [52].

The setup for the test, according to the Whirlpool Motor Lab staff, was as follows.

- CIM motor mounted on test stand and connected to dynamometer and motor controller
- Dynamometer connected to PC with dynamometer control and data collecting software
- Motor Controller connected to FRC battery, CIM, and PWM motor command signal generator

The test procedure is as follows:

- Set dynamometer to zero load
- Set PWM command signal to fixed value
- Automatically ramp up dyno load while automatically collecting speed and torque data at 10 samples/sec

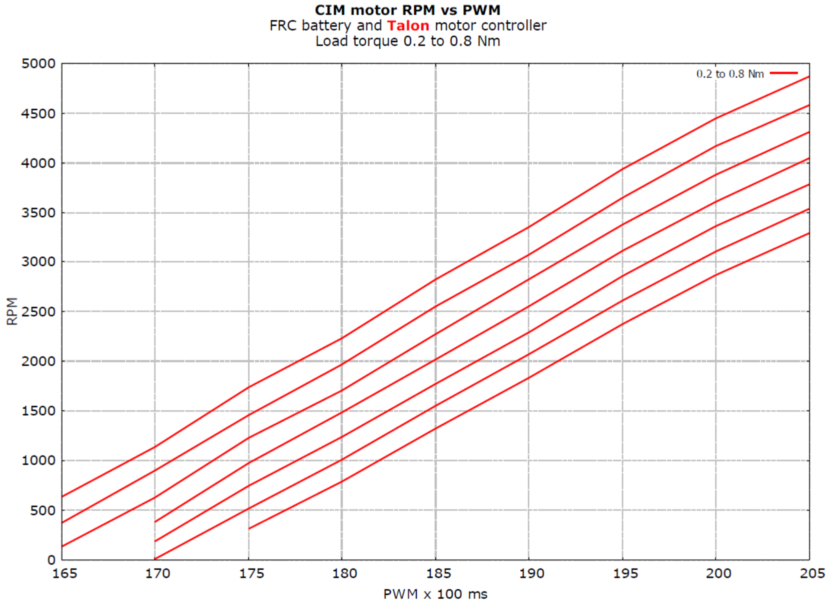

Figure 4.3.6 Talon SR motor controller CIM motor RPM vs PWM test results, courtesy of Whirlpool motor lab

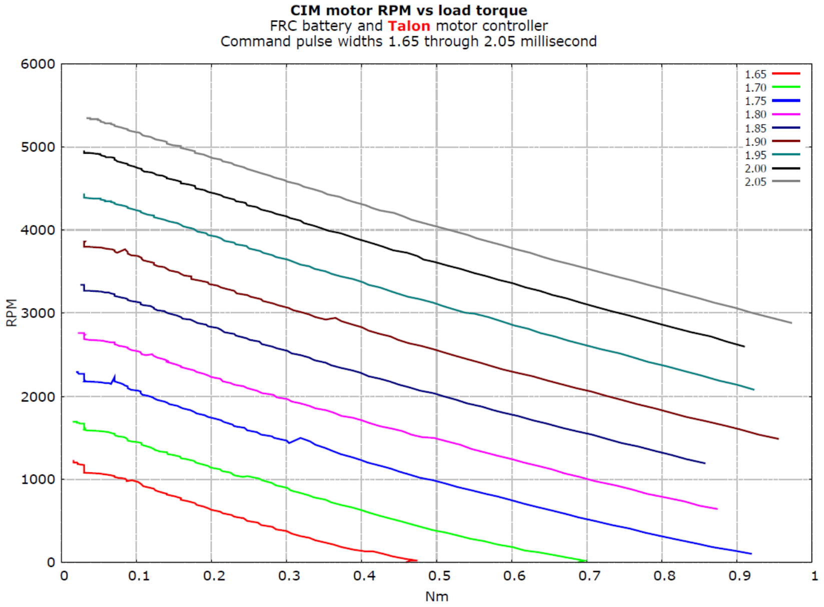

Figure 4.3.7 Talon SR motor controller CIM motor RPM vs load torque test results, courtesy of Whirlpool motor lab

From the graphs in figures 4.3.6 and 4.3.7, we can see that the Talon motor controller generally produces a fairly linear RPM response in the motor during varied load torque and constant PWM pulse widths, particularly with higher PWM pulse widths (1.95 ms and above, as seen in figure 4.3.7). The RPM response is also linear when the torque load is constant and the PWM pulse width is varied. These results are important for this application because the loads on the actuators are constantly varying due to the varying loads experienced by the actuators at different orientations of the platform. The variable loads are made to vary further by the constantly changing forces from the spraying process, which could be caused by pressure changes in the spray hose or a pause in the spraying process.

Due to the nature of spraying applications, high forces on the actuator legs are not expected. The goal of designing a spray platform is to make the spray head platform, and any connected hoses that deliver the spray itself, as mobile and lightweight as possible, while maintaining the accurate positioning of the platform. Therefore, the main loads experienced by the actuator legs comes from the weight of the platform, which should be designed to be lightweight, thereby resulting in a relatively small load on the actuators. It is assumed that reactionary forces resulting from the spraying process are small. However, while determining the magnitude of these forces is outside the scope of this project, they must be determined once the spraying process has been designed. Obviously, the actuators must be selected to ensure they are capable of operating under such loads, including high loads momentarily.

The specifications for the Thomson Electrak 1 SP12-09A-04 state that the maximum dynamic load that is supportable by the actuator is 25 lbs. While supporting a 25 lb load, the speed that the actuator can move is 5.08 cm/sec. The maximum static load that the actuator can support is 250 lbs. Additionally, The no load speed is given as 7.62 cm/sec. Since there will always be a load on the actuators from the fluid being sprayed or the weight of the platform itself, we can expect, during normal operation, that each actuator will operate at a speed somewhere between 7.62 cm/sec and 5.08 cm/sec.

Although the spraying problem is not well defined for this application, it is very unlikely that the above specifications will be exceeded, or not be sufficient, at any time during the spray process. Conceivably, an actuator could move too slowly to spray effectively, due to the forces being caused by spraying the fluid, or be unable to move at all, either because the liquid is too viscous or because it is being sprayed at too high of a velocity. However, it is important to keep in mind that one of the strengths of a Stewart platform design is its rigidity, which owes to the fact that the forces experienced by the platform are distributed between three or more platform legs.

While the force distribution to the legs may differ based on the configuration of the platform, the design itself certainly ensures that the maximum dynamic load that the robot is able to support is higher than the 25 lbs of each individual actuator. Since each actuator individually weighs approximately 1.5 lbs, they have an acceptable speed to weight ratio of 3.387 cm/s/lb at the maximum dynamic load of 25 lbs, which again would be higher when considering the combination of the three actuators. They also each have a high dynamic load to weight ratio, which is 16.67.

The sixth constraint deals with the resistance/sealability of the actuator against environmental contaminants, most commonly dirt and water. In order to make the design more robust to these contaminants, however, an actuator should be selected which is already resistant to environmental contamination, in case the sealing bag is damaged and contaminants come into contact with the actuator [53]. According to the Thomson Electrak 1 manual, these actuators are “weather protected for use in outdoor applications.” Weather protection would provide protection against contaminants, such as rain and dirt, which would be similar to the types of contaminants that the actuators would be exposed to in an underground water pipeline. For this reason, it can be safely assumed that the actuator should be well protected against the environmental contaminants that it would be exposed to in an underground pipeline.

The seventh, and final, constraint states that the actuator should not leak fluid that could potentially contaminate the environment. This criteria is generally not of concern in the selection of an electric actuator because there is not a large amount of fluid that is associated with its use, as there would be in the case of a hydraulic actuator. The only fluid in electric actuator is any fluid used to lubricate the actuator arm, which is a very small amount and of no concern in this application.

When using a hydraulic actuator, however, there is a reasonably high risk that the actuator will leak hydraulic fluid into the environment and lead to contamination. This is of particular importance in a spraying application where any leaked hydraulic fluid could react with the spray coating and interfere with its curing process. Additionally, the pipes being repaired will be used again for transporting clean water, in some cases, and any fluids that remain in the pipe after the repair process (even if they did not interfere with the curing process of the spray coating) could contaminate the clean water supply. This constraint alone effectively eliminates the possibility of using hydraulic actuators in this robot design, and was one of the constraints that could only be satisfied by using pneumatic or electric actuators.

In summary, the main reasons for choosing the Thomson Electrak 1 SP12-09A-04 are:

- The size of the actuator (retracted length of 31.75 cm) satisfies the design requirement of a 50.8 cm maximum height of the robot
- The portability of the power supply design requirement is satisfied because the Thomson Electrak 1 SP12-09A-04 requires a 12 VDC power and has a max amp draw of 5.6 A, both of which can be provided by a relatively small and portable power supply
- Each Thomson Electrak 1 SP12-09A-04 actuator has an acceptable speed to weight and dynamic load to weight ratio of 3.387 cm/s/lb and 16.67, respectively, at a maximum dynamic load of 25 lbs
- The Thomson Electrak 1 SP12-09A-04 actuators are weather protected for use in outdoor applications, thus satisfying the environmental contaminants resistance design requirement

4.4 Microprocessor Selection

The microprocessor selection process was somewhat more straightforward than the actuator selection, due to the fact that there are many microcontrollers available from many different companies which were all capable of performing the tasks that were required by this project. While slightly oversimplified, the basic requirements of microcontroller are as follows:

- Simultaneously output three separate, and variable, pulse width modulation (PWM) waves on three different channels, in order to control the actuator motor speed and direction
- Receive and convert three separate analog values from the built-in potentiometer in each actuator on three separate analog to digital (A/D) conversion channels
- Be able to perform both of the above requirements at a high speed in real-time

The F28377D Delfino Experimenter Kit (pictured in figure 4.4.1) featuring the C2000 Delfino TMS320F28377D microcontroller was selected because it fulfilled all of the requirements for the microcontroller with the additional feature of having four ADCs. The key specifications of the C2000 Delfino TMS320F28377D Table 4.4.1.

Table 4.4.1 Specifications Summary for the F28377D Delfino Experimenter Kit

	TMS320F28377D
CPU Frequency (MHz)	200
RAM (KB)	204
Flash (KB)	1024
PWM (# of channels)	24
ADC (# of modules)	4
ADC resolution	16-bit/12-bit
ADC Conversion Time (ns)	286 (12-bit) 910 (16-bit)
A/D (# of channels)	24 (12-bit) 12 (16-bit)
IO Supply (V)	3.3
Operating Temperature Range (°C)	-40 to 105

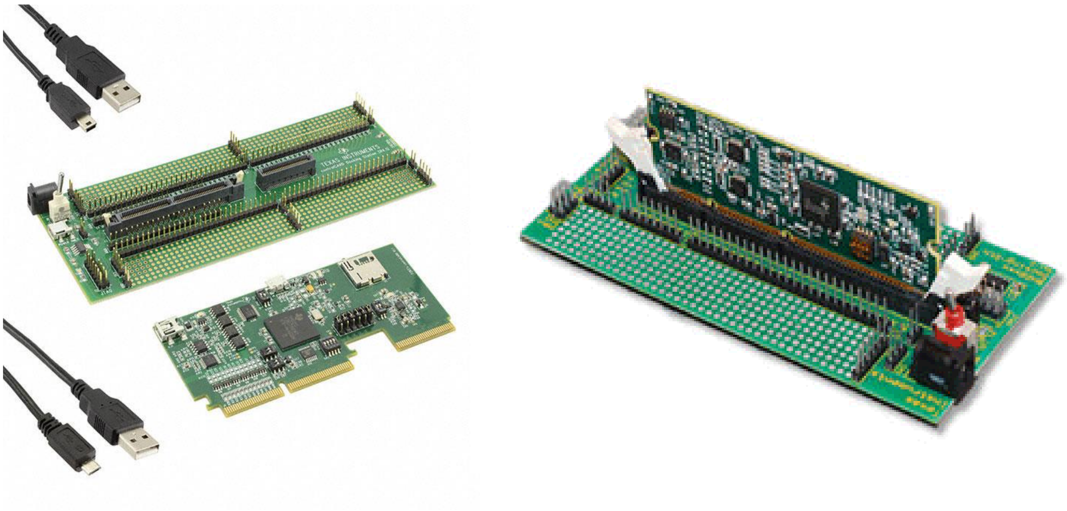

Figure 4.4.1 F28377D Delfino Experimenter Kit [54][55]

Now that all of the components that the robot will be comprised of have been chosen, a block diagram is again shown in figure 4.4.2 with all of the parts listed in their respective block.

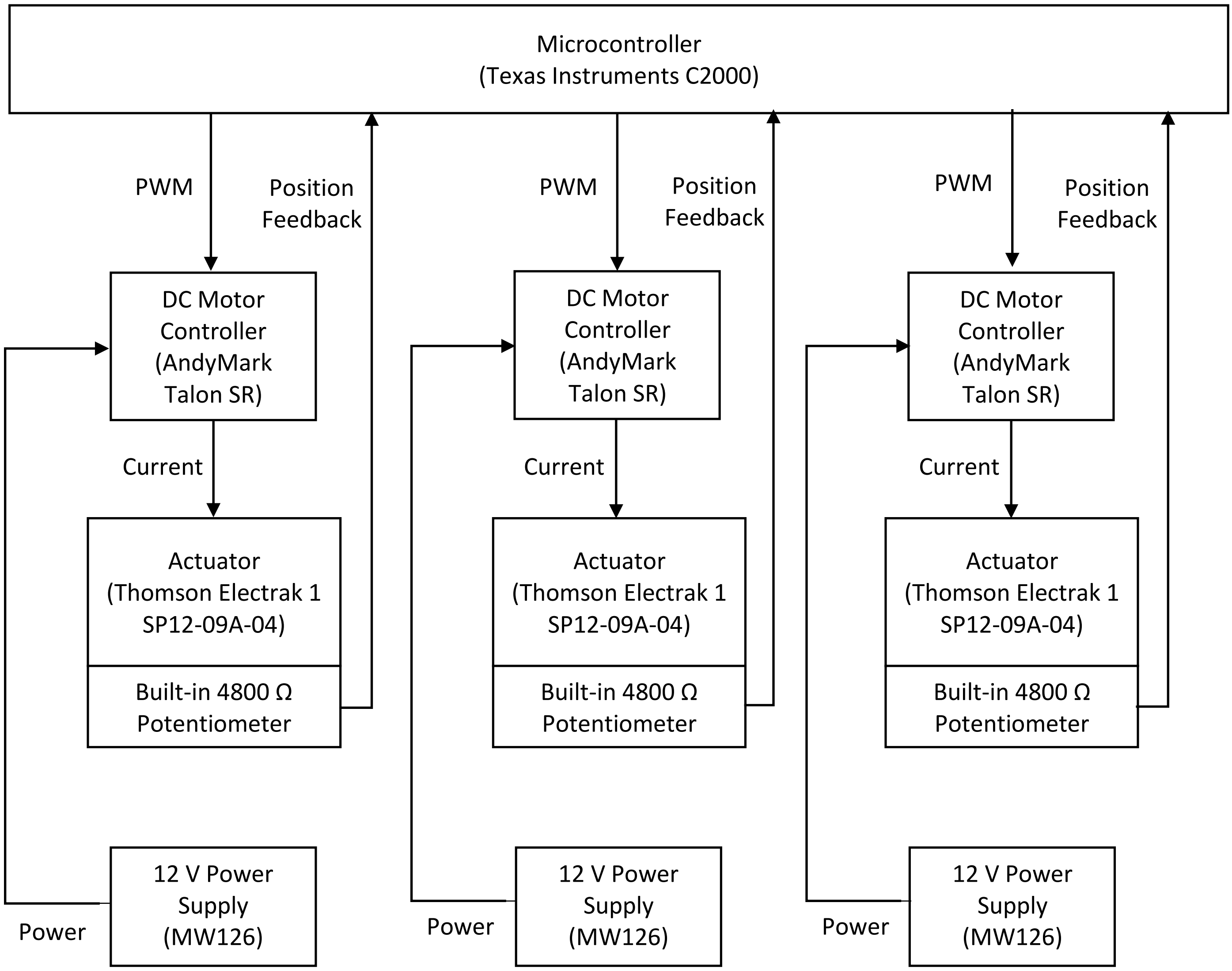
Microcontroller
(Texas Instruments C2000)
PWM
Position Feedback
PWM
Position Feedback
PWM
Position Feedback
DC Motor Controller (AndyMark Talon SR)
DC Motor Controller (AndyMark Talon SR)
DC Motor Controller (AndyMark Talon SR)
Current
Current
Current
Actuator (Thomson Electrak 1 SP12-09A-04)
Actuator (Thomson Electrak 1 SP12-09A-04)
Actuator (Thomson Electrak 1 SP12-09A-04)
Built-in 4800 Ω Potentiometer
Built-in 4800 Ω Potentiometer
Built-in 4800 Ω Potentiometer
12 V Power Supply (MW126)
12 V Power Supply (MW126)
12 V Power Supply (MW126)
Power
Power
Power

Figure 4.4.2 Electrical System Components for a 3-DOF Stewart Platform with all Parts Listed

Chapter 5

Circular Trajectory Implementation

This chapter will discuss how the circular trajectory was implemented using the Delfino TMS320F28377D microcontroller in the C language environment, the experimental set-up and how the platform orientation data will be extracted using image processing techniques from several videos.

5.1 Description of Implementation Code

In order have the platform normal vector travel in a circular trajectory, we first need to find the orientation of the platform in terms of ψ (roll), θ (pitch) and $\emptyset$ (yaw), in degrees for a specific number of points. The higher the number of points for the defined trajectory in each array, the smoother the trajectory will appear. Seventeen points was selected as the optimum number giving a smooth trajectory, while taking into account the time that it takes to find and code each trajectory. To find the numbers that need to be coded into the array (i.e. the leg lengths for each orientation), a separate MATLAB® script file was used that accepts the ψ, θ, and $\emptyset$ values in radians and returns the leg lengths in inches.

The code is written in the programming language C, and programmed in the RAM memory of the Delfino microcontroller. The user can control the lengths of each actuator leg by hardcoding (which were calculated separately and prior to running the program on the microcontroller, as described in the above paragraph, in order to reduce computing time) the leg lengths (in inches) into three separate arrays: L1Array, L2Array and L3Array. The index to each array (Index1, Index2 and Index3) are initialized at 0 and increment by 1 after all three legs have reached the leg length specified in their array. When each leg reaches its specified length for that index, a time is recorded using the CPU timers on the Delfino microcontroller. From these times, the program can determine if all three legs have reached their specified lengths for that index number, and which was the last leg to reach its length. Once the last leg has reached its specified length, all three indices increment at the same time so that each leg begins to move to the next leg length specified by their array at the same time. The digital value for each length is given by the following equations:

$$L1Digital = -515(L1Array[Index1]) + 4025 \tag{39}$$

$$L2Digital = -501(L2Array[Index2]) + 4105 \tag{40}$$

$$L3Digital = -478(L3Array[Index3]) + 4061 \tag{41}$$

These equations were calculated by measuring the stroke length of each leg at every 0.5 inches from 0 inches to 4 inches and then plotting the results in Microsoft Excel and using the equation that results from the linear line of best fit function. Since the actuators stroke length always starts at 0 inches, the first leg length always requires the actuator stroke to increase. Once it reaches its specified value, that value is stored as the last actuator length. Then, based on whether that stored number is greater or less than the next value in the array, the actuator moves forward or backward, respectively. The actuator length is controlled by the microcontroller PWM outputs. A pulse width that is more than 1.111 ms lengthens the actuator (drive motor runs clockwise), while a pulse width less than 1.111 ms shortens the actuator (drive motor runs counter clockwise). When the pulse width equals 1.111 ms, the actuator stops.

To ensure accurate control, the ADCs are continually updating the position of the actuator using the position feedback potentiometer signals being given to the microcontroller through the A/D channels. The process of all three legs reaching their specified length for the index number, and then subsequently incrementing the index number, is repeated until the last value in the array is reached, at which point all three legs return to a stroke of zero inches. The program is then run using the debugger in CCS as many times as required. It can also be run independently from the computer as long as the microcontroller has adequate power.

5.2 Experimental Set-up

The Stewart platform is placed on top of the desk and is connected as shown in figure 5.2.1. Each actuator leg is connected to its respective DC motor controller, all of which are mounted on a chassis that can be seen near the middle of the desk. These motor controllers are connected their respective DC power supply. Each PWM signal is delivered to the DC motor controller through a hobby cable (pictured in figure 5.2.1 with the red, white and black wires) connected into the top of motor controller. The PWM signals originate from the pins on microcontroller and are connected to the hobby cables through the breadboard. Five volts is applied across each potentiometer through connections on the breadboard and the potentiometer is connected to the corresponding A/D pin on the microcontroller. There are two USB cables that are connected to the computer: one powers the microcontroller with five volts and the other connects it to the computer so that it can be programmed.

To determine the orientation of the platform, a circular piece of paper with 8 different colored lines was taped to the top of the robot, as seen in figure 5.2.2. A Logitech HD webcam C525 was connected to an arm as shown in figure 5.2.3. The arm was clamped to the desk to lessen any vibration that would be transferred from the arm to the webcam while the robot was running. Videos of the platform were taken from a top down view so that they could later be processed to extract the orientation of the platform while the circular trajectory code was running.

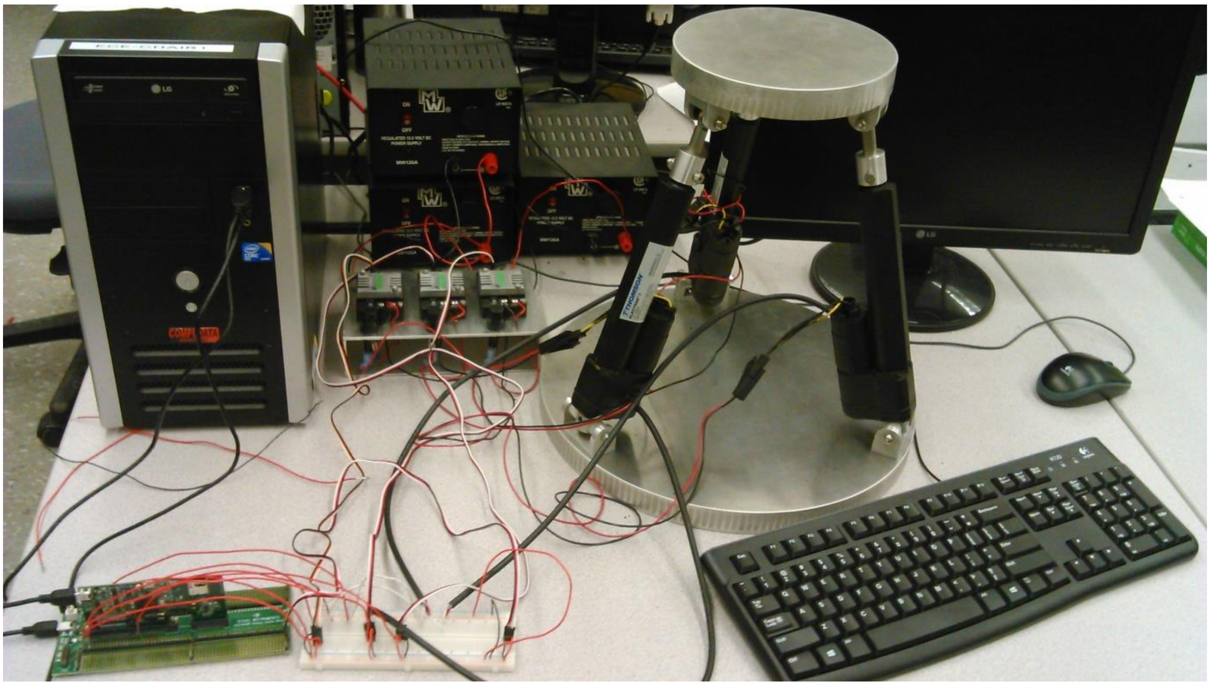

Figure 5.2.1 Experimental Set-up before Arm and Orientation Paper Added

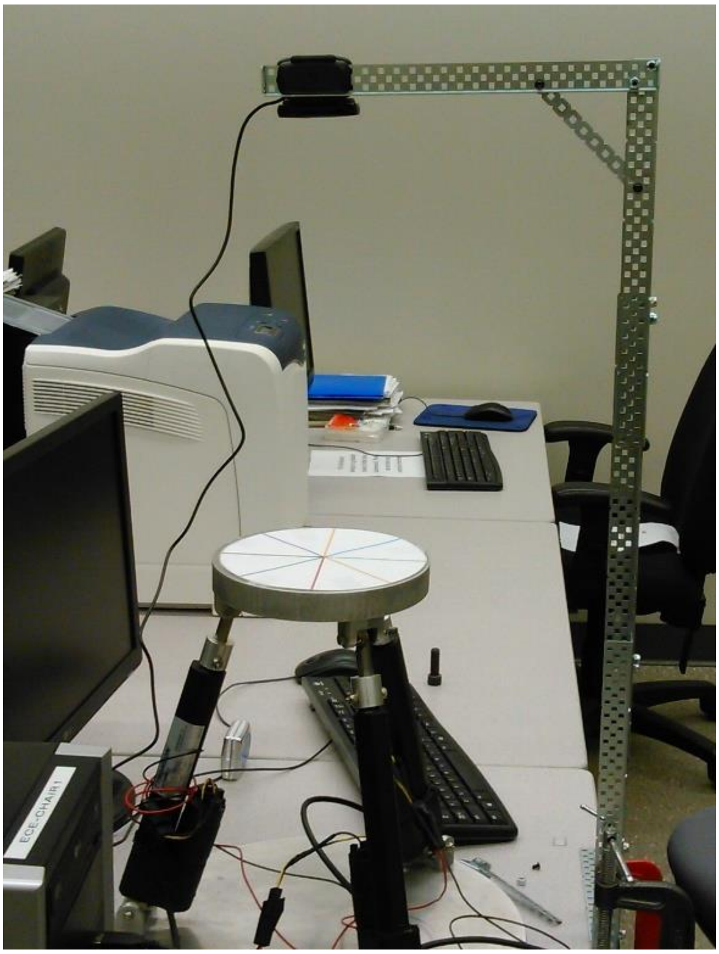

Figure 5.2.2 Experimental Set-up with Side View of Webcam and Arm

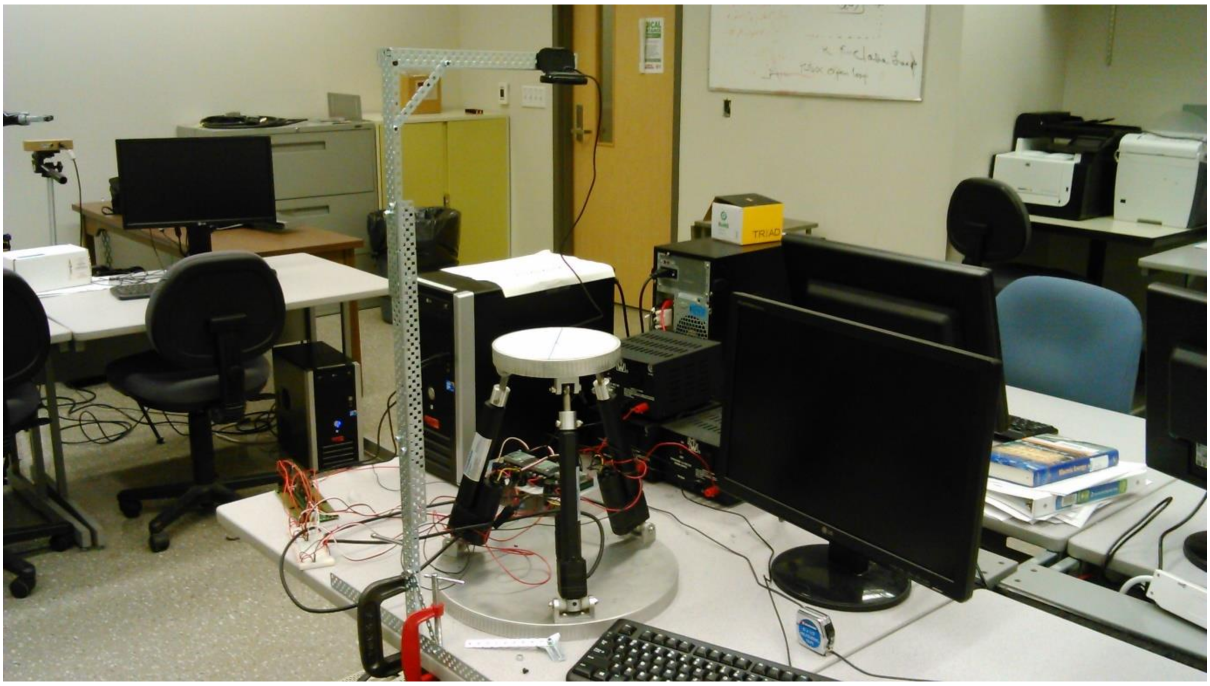

Figure 5.2.3 View of Entire Experimental Set-up after Orientation Paper and Arm Added

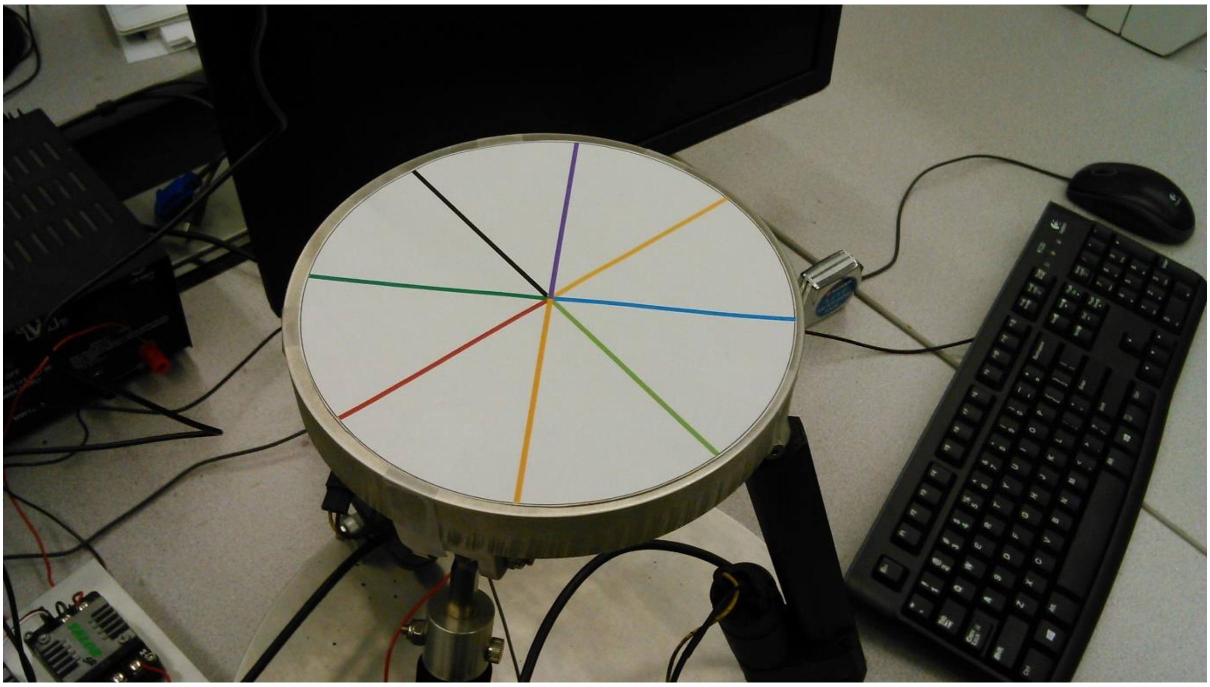

Figure 5.2.4 Close-up View of Orientation Paper

5.3 Experimental Results

Using the experimental set-up that was described in section 5.2, several videos were recorded of the platform normal vector following two different circular trajectories: one where there is an approximately 10° angle between the platform normal vector and the z-axis, and one where there is an approximately 20° between the platform normal vector and the z-axis. From these videos, it has been qualitatively verified that the platform normal vector does follow these trajectories. In order to quantitatively verify the trajectories, an algorithm will need to be developed that is able to determine the actual orientation of the platform as it reaches each of the seventeen desired orientations programmed into the arrays. This algorithm will discern the orientation of the platform based on the apparent changes in the lengths of the different colored lines that can be seen in the videos as the platform follows the trajectories. Once the actual orientation is known, the actual location of the platform joint centers (${}^{P}\overrightarrow{p_\iota} = \left[p_{ix}\ p_{iy}\ \ p_{iz}\right]^T$) can be calculated using inverse kinematics. Subsequently, the error can be determined between the desired platform joint center locations and the actual platform joint center locations. From this error, the accuracy of the platform can be quantitatively verified.

Chapter 6

Conclusions and Future Work

This chapter summarizes the work done during the course of this project. Also included is a discussion of the results, the implications of these results, as well as ideas for future work and where the project can be taken from this point on.

6.1 Summary of Project and Conclusions

At the beginning of this thesis, four research goals were listed:

1. Develop a Stewart platform through optimization techniques and by analyzing its kinematic properties.
2. Improve the speed and efficiency of existing optimization algorithms.
3. Research and choose the components that are most suited to the conditions and application of a pipeline spraying robot.
4. Build a 3-DOF Stewart platform prototype so that the trajectory path can be tested.

6.1.1 Summary of Parameter Optimization

An algorithm was proposed that uses a combination of the CRS algorithm to quickly find parameters that are in the neighbourhood of the optimum, and then use those parameters to speed up the convergence of the PSO algorithm by partially populating its swarm with those values. This proposed algorithm was first tested by optimizing a Delta robot so that it had the maximum workspace volume. The Delta robot was chosen because it is a parallel robot, but it has a simpler design than a 3-DOF Stewart platform. Five trials of the optimization were run using a PSO only algorithm and then another five trials were run using the proposed algorithm. The results of these trials were compared and it was found that the proposed algorithm converged to the optimum significantly faster overall than the PSO only algorithm.

Once the effectiveness of the proposed algorithm had been verified by optimizing the parameters of the Delta robot, five trials of optimizing the Stewart platform were run using the PSO only algorithm, and five trials of optimizing the Stewart platform were run using the proposed algorithm. Since the focus of this project was to develop a Stewart platform capable of delivering a spray coating in a circular trajectory, a different objective function had to be devised rather than simply maximizing the workspace volume, because the 3-DOF Stewart platform has only one translational DOF. Therefore, an objective function was created that maximized the radius of the largest circular trajectory that the Stewart platform was capable of spraying for each base and top plate radius that were generated. When the results of the trials were compared, it again showed that the proposed algorithm decreased the total time it took for it to converge when compared to the PSO only algorithm. An especially important result was that the convergence speed of the PSO portion of the proposed algorithm was significantly faster than the convergence speed of the PSO only algorithm.

6.1.2 Summary of Robot Design

This chapter described the selection process for the individual components that comprise the robot. The first major component selected was the actuator. It was decided that an electric actuator would be the most suitable type of actuator for this application predominantly because of its accuracy, controllability, built-in position sensor, portability, robustness and sealability against environmental contamination. Specifically, the Thomson Electrak 1 SP12-09A-04 was chosen because it included aforementioned features, and its dimensions allowed it to be easily integrated into the robot design envelope. The Talon SR speed controller was chosen as the DC motor controller based on its performance graphs, and the MW126 power supply was selected because of its portability and its adequate power output.

The second major component selected was the microprocessor, and this subsection is intended to be used as a guide for future projects that require the selection of a microprocessor based on a certain application. Three different microprocessor were tested at different parts of the design process. The Motorola 68HC12 was the first to be tested, but it suffered the drawbacks of having too few PWM output channels, a single ADC and it is programmable only in assembly language. The second microcontroller tested was the TMS320F28027 Piccolo microcontroller. This microcontroller was found to have difficulty with simultaneously sampling and converting the three positon signals from each of the potentiometers on the actuators.

Finally, the Delfino TMS320F28377D microcontroller was chosen. It featured four ADCs and it did not encounter the same problems with simultaneous signal sampling that were experienced with the Piccolo microcontroller. Therefore, this microcontroller was chosen as the one to be used in the robot prototype.

6.1.3 Summary of Trajectory Implementation

Two circular trajectories were implemented in the code that controls the Stewart platform prototype: one where there is an approximately 10° angle between the platform normal vector and the z-axis, and one where there is an approximately 20° between the platform normal vector and the z-axis. In order to determine how well the platform is able to follow these circular trajectories, several videos were recorded where a top down view of the colored lines taped onto the platform can be seen. From these videos, it has been qualitatively determined that the platform normal vector is able to follow the circular trajectories. However, an algorithm still needs to be developed that is able to extract the orientation of the platform at each of the seventeen desired orientations in the arrays. This algorithm will determine the actual orientation of the platform at each index in the array based on the apparent change in the length of the colored lines that can be seen in the videos of the platform running through the trajectories. Once the orientation at each index has been determined, the inverse kinematics can be used to calculate the each of the three platform joint centers. An error can then be calculated between the desired platform joint centers and the actual platform joint centers to quantitatively assess the accuracy of the Stewart platform prototype.

6.2 Recommendations for Future Work

There are many possibilities for future projects to build on the work present in this thesis because there are many different aspects of designing and optimizing a Stewart platform (or any robot for that matter) that still need to be explored, and there are still many diverse approaches to this design problem that are being proposed by researchers around the world. Since there were two distinct, but related, parts to this master's project, one of which being the proposed optimization algorithm and the other being the design and development of the 3-DOF Stewart platform, there are also two different directions that any future projects could take.

The proposed algorithm that combines the CRS algorithm and the PSO algorithm was shown to increase the convergence of the parameter optimization problem to an optimum workspace value and an optimal spraying trajectory radius value in the delta robot and the Stewart platform, respectively. Although both of the optimization problems that were used to test the proposed algorithm dealt with robot parameter optimization, that doesn't mean that this algorithm only performs well on problems of this type. Further research could be conducted into determining how well this algorithm performs when trying to find the optimum in other constrained or unconstrained problems. In addition to that, further research could be could be conducted into optimizing the algorithm itself by refining the code so that it runs more efficiently, or by further experimentation with the constant values (such as K_1 and K_2 in the CRS algorithm and c_1 and c_2 in the PSO algorithm) and how the initial PSO matrix is populated by the results of the CRS algorithm so that the proposed algorithm will converge even faster.

As discussed in section 5.3, an algorithm still needs to be developed that is capable of extracting the orientation of the platform from each video based on the apparent changes in the length of the colored lines that can be seen on the top of the platform. After the accuracy of the trajectory has been verified, the logical next step for the Stewart platform prototype would be to mount a spray head to the top plate. Once the spray head is attached to the top plate, further tests can be performed regarding how accurately the platform is following the circular trajectory of the spray path. By adding the spray head, the dynamics of the coating flowing through the spray head, as well as the stiffness of the hose connected to the spray head, is added to the kinematic problem that was tested in this master's project. It is likely that a PID controller will need to be incorporated into the code that implements the circular trajectory, in order to improve the accuracy of the Stewart platform now that the spray head has been added. Additionally, the implementation code should be expanded to include a section that would give a robot operator the ability to manually control the spray head of the robot (with a joystick, for example). This feature would give the operator the ability to direct the robot to spray an additional amount of coating in the larger holes, which wouldn't normally receive enough coating to fill them in during the automated spray process.

Another facet of the design that would need to be developed prior to testing this robot in the field would be the delivery system of how the robot is delivered into the pipeline or manhole and also how it moves through the pipeline or manhole so that it is able to spray it from beginning to end. This was not the focus of this project, but, nonetheless, some ideas about a possible delivery systems were discussed at points during the development of the Stewart platform prototype. Since there would need to be a hose attached to the robot to supply the coating fluid to the spray head, this hose could be stiff enough that it would be possible to push the robot through the pipe using the hose. Another more complicated, and costly idea, was to design a serial manipulator arm that would be attached to the base of the Stewart platform robot and it could be delivered more accurately using the arm. This first option is more suited to delivering the robot into a pipeline, while the second option is more suited to delivering the robot into a manhole.

Once the accuracy of the robot with the spray head is confirmed and the delivery system designed and integrated into the robot design, the final step in the testing process would be a field test where the robot is tested in an actual pipeline or manhole (or first in a pipe cut-out with similar environmental conditions). If the field test is successful the robot and all supporting equipment (computers, power supplies, spray hose, delivery system, etc.) would need to be integrated into a large truck so that it could be easily transported to and from each worksite.

References

[1] D.L. Pieper. “The Kinematics of Manipulators under Computer Control,” Ph.D. Dissertation, Dept. Mech. Eng., Stanford Univ., Stanford, CA, 1968.

[2] V. Dukovski and Z. Pandilov, “Comparison of the Characteristics between Serial and Parallel Robots,” *Acta Technica*, vol. 7, no. 1, pp. 143-160, Mar. 2014.

[4] J. Merlet, in *Parallel Robots,* 2nd ed. Dordrecht, Netherlands: Springer, 2006, pp. 2-3.

[5] J. Merlet, in *Parallel Robots,* 2nd ed. Dordrecht, Netherlands: Springer, 2006, pp. 2.

[6] J. Merlet, in *Parallel Robots,* 2nd ed. Dordrecht, Netherlands: Springer, 2006, pp. 5-12.

[7] A. Akbas. (2008, April 1). *Application of Neural Networks to Modeling and Control of Parallel Manipulators* [Online]. Available: http://www.intechopen.com/books /parallel_manipulators_new_developments/application_of_neural_networks_to_modeling_and_control_of_parallel_manipulators Aug. 1, 2015 [date accessed].

[8] EA Services. *Sprayed in place pipe – Robotic Pipe Lining* [Online]. Available: http://www.pipeliningusa.com/sipp.html Aug. 1, 2015 [date accessed].

[9] V. E. Gough and S. G. Whitehall, “Universal Tyre Test Machine,” in *Proc. 9th Int. Congr. F.I.S.I.T.A.*, May 1962, pp. 117-137.

[10] D. Stewart, “A Platform with Six Degrees of Freedom,” *Proc. of the Inst. Mech. Eng.*, vol. 180, no. 1, pp. 371-386, June 1965.

[11] C. Gosselin and J. F. Hamel, “The Agile Eye: A High Performance Three-Degree-of-Freedom Camera-Orienting Device*,”* in *IEEE Int. Conf. on Robotics and Automation*, San Diego, 1994, pp. 781-786.

[12] T. Arai et al., “Development of a Parallel Link Manipulator,” in *Proc. 5th Int. Conf. on Advanced Robotics*, Pisa, 1991, pp. 839–844.

[13] T. P. Jones and G. R. Dunlop, “Analysis of Rigid-Body Dynamics for Closed Loop Mechanisms—Its Application to a Novel Satellite Tracking Device,” *J. of Syst. & Control Eng.*, vol. 217, no. 4, pp. 285-298, June 2003.

[14] J. Merlet, and D. Lazard, “The (true) Stewart Platform has 12 Configurations,” in *Proc. IEEE Int. Conf. on Robotics and Automation*., San Diego, 1994, pp. 2160-2165.

[15] R. Ben-Horin et al. “Kinematics, Dynamics and Construction of a Planarly Actuated Parallel Robot,” *Robotics and Computer-Integrated Manufacturing*, vol. 14, pp. 163-172, 1998.

[16] L.W. Tsai et al., “Kinematics of a Novel Three DOF Translational Platform,” in *Proc. IEEE Int. Conf. Robotics and Automation*, Minneapolis, 1996, pp. 3446–3451.

[17] C.C. Ng et al., “Design and Development of 3-DOF Modular Micro Parallel Kinematic Manipulator,” *Int. J. Adv. Manuf. Technol.*, vol. 31, pp. 188-200, 2006.

[18] E. A. Dijksman, *Motion Geometry of Mechanisms*. Cambridge, UK: Cambridge Univ. Press, 1976, pp. 45-46.

[19] S. Kucuk and Z. Bingul, “Robot Kinematics: Forward and Inverse Kinematics,” in *Industrial Robotics: Theory, Modelling and Control*, S. Cubero, Ed. Mammeldorf, Germany: Pro Literatur Verlag, 2006, pp. 117–148.

[20] K. Liu et al., “Kinematic Analysis of a Stewart Platform Manipulator,” *IEEE Trans. Ind. Electron*., vol. 40, no. 2, pp. 282-293, 1993.

[21] E. Weissstein. *Euler Angles* [Online]. Available: http://mathworld.wolfram.com/ EulerAngles.html Aug. 1, 2015 [date accessed].

[22] I. Bonev and J. Ryu, “Workspace Analysis of 6-PRRS Parallel Manipulators Based on the Vertex Space Concept,” in *Proc. ASME Design Engineering Technical Conf.*, Las Vegas, 1999.

[23] C.C. Ng. “3-Axis and 5-Axis Machining with Stewart Platform,” Ph.D. Dissertation, Dept. Mech. Eng., Nat. Univ. Singapore, 2012.

[24] E. Weissstein. *Point-Plane Distance* [Online]. Available: http://mathworld.wolfram.com/Point-PlaneDistance.html Aug. 1, 2015 [date accessed].

[25] Y.J. Lou et al., “Randomized Optimal Design of Parallel Manipulators,” *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 2, pp. 223-233, April 2008.

[26] M. Guillot and C. M. Gosselin, “The Synthesis of Manipulators with Prescribed Workspace,” *Trans. ASME J. Mech. Des.*, vol. 113, no. 3, pp. 451–455, 1991.

[27] L.W. Tsai et al, “Optimization of a Three DOF Translational Platform for Well-Conditioned Workspace,” in *ASME Int. Des. Eng. Tech. Conf. Comput. Inf. Eng. Conf.*, Montreal, Quebec, Canada, 2002.

[28] M. Stock and K. Miller, “Optimal Kinematic Design of Spatial Parallel Manipulators: Application of Linear Delta Robot,” *Trans. ASME J. Mech. Des*., vol. 125, no. 2, pp. 292–301, 2003.

[29] Y.J. Lou et al., “Optimization Algorithms for Kinematically Optimal Design of Parallel Manipulators,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 574-584, April 2014.

[30] Y.J. Lou et al., “Optimization Algorithms for Kinematically Optimal Design of Parallel Manipulators,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 578, April 2014.

[31] Y.J. Lou et al., “Optimization Algorithms for Kinematically Optimal Design of Parallel Manipulators,” *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 2, pp. 579, April 2014.

[32] Y.J. Lou et al., “Randomized Optimal Design of Parallel Manipulators,” *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 2, pp. 229, April 2008.

[33] R. Goulcher and J. C. Long, “The Solution of Steady-state Chemical Engineering Optimization Problems using a Random Search Algorithm,” *Comput. Chem. Eng.*, vol. 2, no. 1, pp. 33–36, 1978.

[34] J. R. Banga and W. D. Seider, “Global Optimization of Chemical Processes using Stochastic Algorithms,” in *State of the Art in Global Optimization: Computational Methods and Applications*, C. A. Floudas and P. M. Pardalos, Eds. Norwell, MA: Kluwer, 1996, pp. 563–583.

[35] F.J. Soils and R.J.B. Wets, “Minimization by Random Search Techniques,” *Math. Oper. Res*., vol. 6, no. 1, pp. 19–30, 1981.

[36] J. Kennedy and R. C. Eberhart, “Particle Swarm Optimization,” in *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.

[37] F. Pierrot et al., “Towards a Fully-parallel 6 DOF Robot for High-Speed Applications,” in *Proc. IEEE Int. Conf. Robotics and Automation*, Sacramento, 1991, pp. 1288-1293.

[38] M. Lopez et al., “Delta Robot: Inverse, Direct, and Intermediate Jacobians,” *J. Mech. Eng. Sci.*, vol. 220, pp. 106, 2006.

[39] P. Fauchais and A. Vardelle, “Heat, Mass and Momentum Transfer in Coating Formation by Plasma Spraying,” *Int. J. of Thermal Sci.*, vol. 39, pp. 852–870, 2000.

[40] D.T.I., "Contamination Control in Fluid Power Systems, Vol 1 Field Studies", Dept. Trade and Industry, East Kilbride, Glasgow, UK, 1984.

[41] M.D. Pall et al., “Setting Control Limits for Water Contamination in Hydraulic and Lube Systems,” in *Proc. Tenth Int. Scandinavian Int. Conf. on Fluid Power,* Tampere, Finland, 2007, pp. 307-317.

[42] R. Cantley, “The Effect of Water in Lubricating Oil on Bearing Fatigue Life,” *ASLE Trans.*, vol. 20, no. 3, pp. 244-248, 1976.

[43] H. Merritt, “Introduction,” in *Hydraulic Control Systems*. Hoboken, NJ: Wiley, 1967, pp. 1-3.

[44] *Actuator, Hydraulic – Description* [Online]. Available: http://www.daerospace.com/HydraulicSystems/ActuatorHydraulicDesc.php Aug. 1, 2015 [date accessed].

[45] S.R. Majumdar, in *Pneumatic Systems Principles and Maintenance.* New Delhi, India: Tata McGraw-Hill, 1995, pp. 259-260.

[46] B. Liptak, “Control Valve Selection and Sizing,” in *Instrument Engineers’ Handbook*, 4th ed., vol. 2. Boca Raton, FL: CRC Press, 2006, pp. 1127.

[47] Appin Knowledge Solutions, “Actuators,” in *Robotics.* Hingham, MA: Infinity Science Press, 2007, pp. 126.

[48] P. Sandin, “Motor and Motion Control Systems,” in *Robot Mechanisms and Mechanical Devices*. New York, NY: McGraw-Hill, 2003, pp. 3-5.

[49] S. Mraz. (2004, Jan. 8). *Modular Actuator Meets Flexible Demands* [Online]. Available: http://machinedesign.com/archive/modular-actuator-meets-flexible-demands Aug. 1, 2015 [date accessed].

[50] Ether. (2012, Nov. 5) *Talon, Victor884, Victor888, and Jaguar Speed vs. Torque Tests* [Online]. Available: http://www.chiefdelphi.com/media/papers/2720 Aug. 1, 2015 [date accessed].

[51] Thomson Linear. *Linear Actuator – Electrak 1SP* [Online]. Available: http://www.thomsonlinear.com/ website/com/eng/products/actuators/electrak_1.php Aug. 1, 2015 [date accessed].

[52] AndyMark. *Talon SR Speed Controller Information* [Online]. Available: http://www.andymark.com/Talon-p/am-talon-discontinued.html Aug. 1, 2015 [date accessed].

[53] *Electrak 1 Series Installation & Operation Manual*, Thomson Linear, Marengo, IL, 2014, pp. 10.

[54] Digi-Key. *F28377D Delfino Experimenter Kit* [Online]. Available: http://www.digikey.com/catalog/en/partgroup/f28377d-delfino-experimenter-kit /45953 Aug. 1, 2015 [date accessed].

[55] Texas Instruments. *Experimenter* Kits [Online]. Available: http://www.ti.com/ lsds/ti/microcontrollers_16-bit_32-bit/c2000_performance/real-time_control /f2833x_f2837x/tools_software.page Aug. 1, 2015 [date accessed].

[56] Texas Instruments. *C2000 Piccolo LaunchPad* [Online]. Available: http://www.ti.com/tool/launchxl-f28027 Aug. 1, 2015 [date accessed].

[57] J. Thatcher et al. (2009, Apr.). *NAND Flash Solid State Storage for the Enterprise, an In-depth Look at Reliability* [Online]. Available: http://www.snia.org/sites/ default/files/SSSI_NAND_Reliability_White_Paper_0.pdf Aug. 1, 2015 [date accessed].

Appendix A – Microprocessor Selection

This section gives a detailed description of the selection process for a microprocessor that is suitable for controlling the robot actuators to achieve the circular trajectory required for the spraying process. The amount of detail that is given is intended to give the reader insight into the selection process to such a degree that this section can also be used as a tutorial for any future students who are undertaking a similar design process. I personally found that there was generally a lack of information regarding the selection process of the components that made up the robot and the microprocessor selection in particular, and I hope that the inclusion of this section will at least provide a starting point for those future students.

The Motorola 68HC12 microcontroller mounted on a platform (shown in figure A.1) with a built-in user controlled potentiometer, a breadboard, input/output rings, and other components that were not used for this project, was used to test the performance of the DC motor controllers and the actuators. This microcontroller was readily available from the university electronics shop and was ideal to use as a starting point to gain a better understanding of how the PWM wave controlled the motors, specifically the frequency and the duty-cycle that are required to achieve an acceptable movement speed of the actuator. This was performed by controlling the duty cycle of a PWM wave with the user controlled potentiometer. The user controlled PWM wave output is connected to the DC motor controller which causes the actuator to move forward or backward at a speed controlled by the user, or stop.

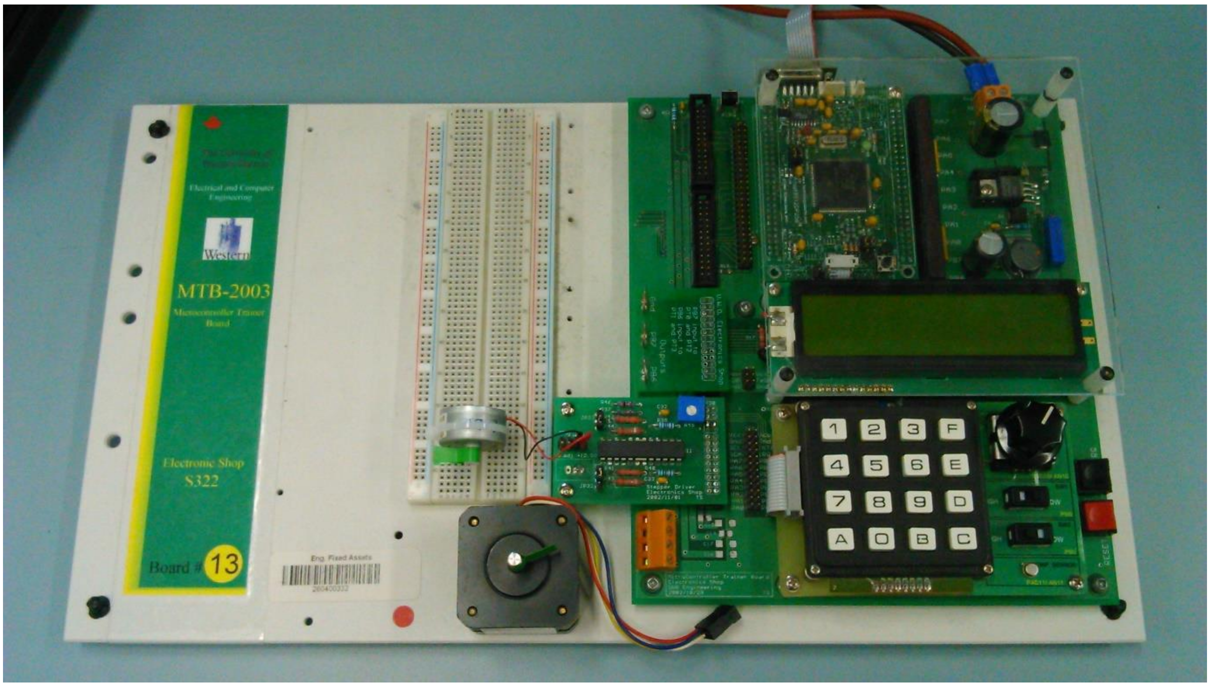

Figure A.1 68HC12 Microcontroller mounted on Platform

Although the use of the 68HC12 microcontroller was a good starting point for learning how to integrate all of the components and control the actuator, this microcontroller has many drawbacks that prevented it from being used in the later stages of the robot prototype development. The first and most important drawback for this microcontroller is that it does not have the required amount of PWM channels to control all three actuators at once. The second drawback is that it only has one analog to digital converter (ADC), which could potentially cause a significant delay between the actual position of the actuator arm and the last value that the ADC converted. This delay is the result of the difference in the time that it takes for the sample to be taken, and the conversion to occur. This delay is normally not significant, but when there are three actuator arm position values constantly needing to be converted, and only one can be done at a time, the delay can be significant enough to cause errors in the feedback loop.

The final major drawback is that the 68HC12 microcontroller is programmed using assembly language. This drawback is more of a personal preference for the person programming the microcontroller, but some programmers prefer using a high-level programming language. Although there are applications for using assembly language, many newer microcontrollers use a high-level programming language compiler, and come with sample code for PWM wave generation and ADC conversions, on which the programmer can base their code.

Newer microcontrollers typically allow the user to easily specify the frequency and duty cycle of the PWM, or how quickly the ADC channels are sampled, and which ADC channels are sampled. With these newer microcontrollers, you can simply configure the PWM wave and ADC for your particular application without having to build the code from the ground up, meaning that less time is spent writing the code, and more time spent on optimizing the code.

A second microcontroller, C2000 Piccolo Launchpad, was chosen (shown in figure A.2). The C2000 Piccolo Launchpad uses the Piccolo TMS320F28027 microcontroller, and the specifications most relevant to this project are summarized in Table A.1.

Table A.1 Specifications Summary for C2000 Piccolo Launchpad

	TMS320F28027
CPU Frequency (MHz)	60
RAM (KB)	12
Flash (KB)	64
PWM (# of channels)	8
ADC (# of modules)	1
ADC resolution	12-bit
ADC Conversion Time (ns)	217
A/D (# of channels)	7 (12-bit)
IO Supply (V)	3.3
Operating Temperature Range (°C)	-40 to 105

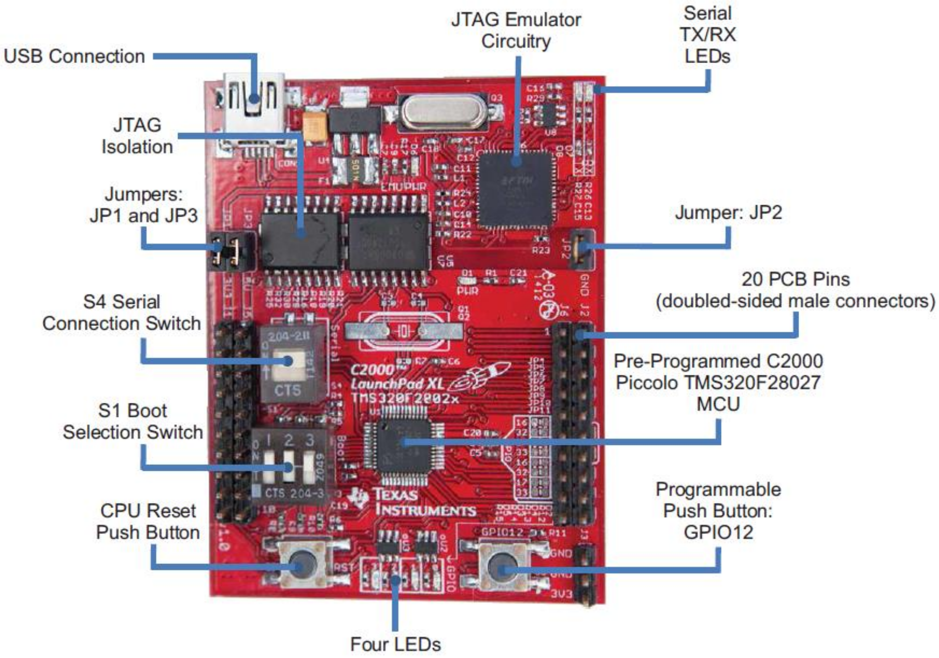

Figure A.2 C2000 Piccolo Launchpad [56]

As we can see from table A.1, the requirements for the project are exceeded by this microcontroller. To control the speed and direction of the actuator motors, only three PWM channels are required, and the Piccolo microcontroller has eight. To receive feedback from each of the three potentiometers built into the actuators, three A/D channels are required, and the Piccolo microcontroller has seven. The resolution on the channels is also more than required for accurate control, with a resolution of 12-bits.

The main reason for choosing this microcontroller is that it was inexpensive and easily incorporated into the robot design. To program the microcontroller, it first needs to be connected to a computer via a USB cable, and then programmed using a free and unrestricted version of Code Composer Studio integrated development environment (IDE). This version of Code Composer Studio (CCS) came with many examples that implement the various feature of the microcontroller, such as variable PWM wave generation, A/D conversions, interrupts, etc. By combining and modifying the available code, a program was created that allowed the user to input the required leg lengths for each platform orientation. Once the last leg had reached the required height, the cycle would repeat, moving the platform to the next orientation. This process can be repeated indefinitely, but for the purposes of testing this project, it was repeated until the vector normal to the platform had passed through the entire circular trajectory that was discussed in chapter two of this thesis. The program details are shown in appendix B of this thesis.

After running this program through many tests to ensure the accuracy of the orientations of the platform for every step, it became obvious that there was an issue with the accuracy and conversion speed of the ADC. Using the real-time debugger that is included in CCS, the results of the three in-use ADC channels were monitored. While two of the channels were updating at an acceptable rate, the third channel was updating one to two seconds slower than the other two. Also, the accuracy of this third channel was of concern. It was only capable of converting the position at certain increments (approximately every 1.3 cm) and was not able to update the ADC value if the actuator arm was in a positon that was in between these increments. This issue was significantly impaired the accuracy of the actuator positioning, as well as limiting the number of positions that it was capable of stopping at.

The issue is believed to stem from the fact that the Piccolo microcontroller has only one ADC and had a problem with the sequential conversion of the samples. A possible reason for why two of the three channels did work properly is because there are two sample and hold circuits that feed the ADC. Once the third channel was being converted a high speed, the ADC became overwhelmed and the conversion priority was with the other two channels, and therefore, only these channels continued to be converted correctly.

Since no software or hardware solution to this problem could be found using the already acquired software and hardware, a microcontroller more suited to multi-channel, high speed A/D conversions needed to be researched and procured. The solution seemed to be multiple ADCs (preferably one per channel), in order to process all of the conversions in real-time. Fortunately, Texas Instruments manufactures a line of C2000 microcontrollers, all of which are programmed using CCS. In addition to that, the code written for any C2000 microcontroller is compatible (with minor modifications) with any of the other C2000 microcontrollers. Because the code was already written for the C2000 Piccolo microcontroller, the decision was made to select a higher end member of the C2000 family to replace the C2000 Piccolo that was used previously.

As stated in section 4.4, the F28377D Delfino Experimenter Kit (pictured in figure 4.4.1) featuring the C2000 Delfino TMS320F28377D microcontroller was selected because it fulfilled all of the original requirements for the microcontroller with the additional feature of having four ADCs. A comparison of the key specifications of the C2000 Delfino TMS320F28377D and TMS320F28027 Piccolo microcontrollers is shown in Table A.2.

Table A.2 Specifications Summary for C2000 Piccolo Launchpad vs. the F28377D Delfino Experimenter Kit

	TMS320F28027	TMS320F28377D
CPU Frequency (MHz)	60	200
RAM (KB)	12	204
Flash (KB)	64	1024
PWM (# of channels)	8	24
ADC (# of modules)	1	4
ADC resolution	12-bit	16-bit/12-bit
ADC Conversion Time (ns)	217	286 (12-bit)

		910 (16-bit)
A/D (# of channels)	7 (12-bit)	24 (12-bit) 12 (16-bit)
IO Supply (V)	3.3	3.3
Operating Temperature Range (°C)	-40 to 105	-40 to 105

While it is obvious by looking at table A.2 that the Delfino TMS320F28377D is better in nearly every aspect, with the exception of ADC conversion time (although the slightly longer conversion time in the Delfino TMS320F28377D microcontroller is more than compensated for by the three additional ADCs), only the additional ADCs were absolutely necessary. However, there are other features that were also enhanced that are important for this application. The faster processing power of the CPU could be a factor in the accuracy of the of the actuator control, especially in a real-time application such as this. If the CPU is able to execute the code quicker and respond faster when the position of the actuator arm reaches the required length for a certain platform orientation, the accuracy of the robot should be greater.

The second important advantage that the Delfino TMS320F28377D microcontroller has over the TMS320F28027 Piccolo microcontroller is the larger amount of memory. This is advantageous because the program became too large for the RAM available on the TMS320F28027 Piccolo microcontroller and, consequently, the memory had to be re-mapped to write the program onto the flash memory, which takes longer to write to every time the program is run. This means that less time is spent waiting for the program to write to the memory before it can be debugged, and more time is spent optimizing the code and fixing any errors [57]. Furthermore, flash memory suffers from memory wear, meaning that it can only be erased and programmed a finite number of times. Because of the many times the program had to be re-written and debugged, there was a concern that this memory wear might affect the

TMS320F28027 Piccolo microcontroller. This was not a concern for the RAM used on the Delfino TMS320F28377D microcontroller.

Appendix B – Matlab® Script Files

```
function [l1mag, l2mag, l3mag] = Inverse_Kinematics(R, r, theta, psi, phi)
%*******************************************************************
%Inverse_Kinematics.m - This function accepts the orientation of the top
%                       platform given by theta, psi and phi and the radius
%                       of the base (r) and the radius of the platform (R)
%                       solves the inverse kinematics of a 3-DOF Stewart
%                       platform and returns the length of each leg.
%*******************************************************************

%theta - pitch - rotation about y-axis
%psi - roll - rotation about x-axis
%phi - yaw - rotation about z-axis

a11 = cos(theta)*cos(phi); %elements of the rotatation matrix
a12 = cos(theta)*sin(phi);
a13 = -sin(theta);
a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
a23 = cos(theta)*sin(psi);
a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
a33 = cos(theta)*cos(psi);

q = [0 0 31.75]'; %translation from {B} to {P} the last value is the height
                  %of the robot (top of base to top of platform) (in cm)

%rotation matrix from base to platform
Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

P1 = [R 0 0 1]'; %platform joint locations in reference frame {P}
P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

B1 = [r 0 0]'; %base joint locations in reference frame {B}
B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

u1 = Rm*P1; %vector from platform center to point 1 on platform
u2 = Rm*P2; %vector from platform center to point 2 on platform
u3 = Rm*P3; %vector from platform center to point 3 on platform

%length of legs
l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

end
```

```
function [] = Workspace_Points(R, r)
%*********************************************************************
%Workspace_Points.m - This function accepts the and the radius of the
%                     base (r) and the radius of the platform (R) and
%                     solves the inverse kinematics of a 3-DOF Stewart
%                     platform and returns a plot of all acceptable
%                     workspace points
%*********************************************************************

%theta - pitch - rotation about y-axis
%psi - roll - rotation about x-axis
%phi - yaw - rotation about z-axis

%Initialize empty arrays

%Initialize arrays that will track the theta and phi values at each step
%so that scatter plots can be made
thetatrack = [];
phitrack = [];

%track the x, y and z elements of each u1, u2 and u3 vector
u1trackx = [];
u1tracky = [];
u1trackz = [];

u2trackx = [];
u2tracky = [];
u2trackz = [];

u3trackx = [];
u3tracky = [];
u3trackz = [];

%Initialize z unit vector along z-axis
Z = [0 0 1];

%Initialize theta, phi and psi at zero to find the angles that
%the actuator legs make with the base and platform to be used in
%constraint equations later
theta = 0;
phi = 0;
psi = 0;

height = 31.75; %Initial height of the platform in cm

%Calculate original angle between base and actuator leg and
%platform and actuator leg but only runs once
if (theta == 0 && phi == 0 && psi == 0)

    %Elements of the rotatation matrix
    a11 = cos(theta)*cos(phi);
    a12 = cos(theta)*sin(phi);
    a13 = -sin(theta);
    a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
    a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
    a23 = cos(theta)*sin(psi);
    a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
    a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
    a33 = cos(theta)*cos(psi);
    %Rotation matrix from base to platform
    Rm = [a11 a12 a13;a21 a22 a23;a31 a32 a33];

    %Translation from {B} to {P} the last value is the height of the robot
    %(top of base to top of platform)
    q = [0 0 height]';

    %Rotation matrix from base to platform
    Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

    P1 = [R 0 0 1]'; %Platform joint locations in reference frame {P}
    P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
    P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

    %Base joint locations in reference frame {P}
    B1 = [r 0 0]';
    B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
    B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

    %Vector from platform center to point 1, 2 and 3 on platform
    u1 = Rm*P1;
    u2 = Rm*P2;
    u3 = Rm*P3;

    %Calculated leg lengths
    l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
    l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
    l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

    %Initial angles that are made between base and actuator leg and
    %platform and actuator
    OrigBaseAngle1 = asin(u1(3)/l1mag);
    OrigPlatAngle1 = acos(u1(3)/l1mag);

    OrigBaseAngle2 = asin(u2(3)/l2mag);
    OrigPlatAngle2 = acos(u2(3)/l2mag);

    OrigBaseAngle3 = asin(u3(3)/l3mag);
    OrigPlatAngle3 = acos(u3(3)/l3mag);
end

%Use three for loops to go through all realistic workspace orientations
%at different heights to determine which platform points satisfy all of the
%constraints and are therefore acceptable points which will be plotted

for height = 31.75:0.1:40.64
    for theta = -pi/2:0.1:pi/2
         for psi = -pi/2:0.1:pi/2

thetatrack = [thetatrack theta];
phitrack = [phitrack phi];

a11 = cos(theta)*cos(phi);
a12 = cos(theta)*sin(phi);
a13 = -sin(theta);
a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
```

```
a23 = cos(theta)*sin(psi);
a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
a33 = cos(theta)*cos(psi);

Rm = [a11 a12 a13;a21 a22 a23;a31 a32 a33];

q = [0 0 height]';

Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

P1 = [R 0 0 1]';
P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

B1 = [r 0 0]';
B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

u1 = Rm*P1;
u2 = Rm*P2;
u3 = Rm*P3;

l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

%Find the cross product between the Z-axis and the vector from the origin
%each of the points on the base. These are the normal vectors to the
%constraint planes
N1 = cross(Z,B1);
N2 = cross(Z,B2);
N3 = cross(Z,B3);

%Calculate the distance of each of the platform points form the constraint
%planes
D1 = abs(N1(1)*u1(1) + N1(2)*u1(2) + N1(3)*u1(3))/sqrt(N1(1)^2 + N1(2)^2 +
N1(3)^2);
D2 = abs(N2(1)*u2(1) + N2(2)*u2(2) + N2(3)*u2(3))/sqrt(N2(1)^2 + N2(2)^2 +
N2(3)^2);
D3 = abs(N3(1)*u3(1) + N3(2)*u3(2) + N3(3)*u3(3))/sqrt(N3(1)^2 + N3(2)^2 +
N3(3)^2);

%Check if the points satisfy all of the constraints such as the leg length
%constraint, the alpha and beta angle constraints and the maximum allowable
%distance from the constraint plane constraint

if (l1mag < 41.91 && l2mag < 41.91 && l3mag < 41.91 ...
      && l1mag > 31.75 && l2mag > 31.75 && l3mag > 31.75 ...
      && abs(asin(u1(3)/l1mag)-OrigBaseAngle1) < pi/4 && ...
      abs(acos(u1(3)/l1mag)-OrigPlatAngle1) < 0.44 ...
      && abs(asin(u2(3)/l2mag)-OrigBaseAngle2) < pi/4 && ...
      abs(acos(u2(3)/l2mag)-OrigPlatAngle2) < 0.44 ...
      && abs(asin(u3(3)/l1mag)-OrigBaseAngle3) < pi/4 && ...
      abs(acos(u3(3)/l3mag)-OrigPlatAngle3) < 0.44 ...
        && D1 < 0.25 && D2 < 0.25 && D3 < 0.25)
```

```
    %If the point satisfies all of the constraints, it is added to the
    %arrays that track each of the x, y and z values for each of the points
    u1trackx = [u1trackx u1(1)];
    u1tracky = [u1tracky u1(2)];
    u1trackz = [u1trackz u1(3)];

    u2trackx = [u2trackx u2(1)];
    u2tracky = [u2tracky u2(2)];
    u2trackz = [u2trackz u2(3)];

    u3trackx = [u3trackx u3(1)];
    u3tracky = [u3tracky u3(2)];
    u3trackz = [u3trackz u3(3)];

end

        end
    end
end

%Plot all of the acceptable points on the same scatter plot and include the
%base points as a visual reference
scatter3(u1trackx,u1tracky,u1trackz)
hold
scatter3(u2trackx,u2tracky,u2trackz)
scatter3(u3trackx,u3tracky,u3trackz)
scatter3(B1(1),B1(2),B1(3),'MarkerEdgeColor','k','MarkerFaceColor',...
[0 .75 .75])
scatter3(B2(1),B2(2),B2(3),'MarkerEdgeColor','k','MarkerFaceColor',...
[0 .75 .75])
scatter3(B3(1),B3(2),B3(3),'MarkerEdgeColor','k','MarkerFaceColor',...
[0 .75 .75])
```

```
function [lviol,O_Alph_Viol,O_Beta_Viol,dviol] = Workspace_Vector(R, r)
%*******************************************************************
%WorkspaceVector.m - This function accepts the and the radius of the
%                    base (r) and the radius of the platform (R) and
%                    solves the inverse kinematics of a 3-DOF Stewart
%                    platform and and finds the vector normal to the
%                    platform for each acceptable orientation and
%                    returns a plot of all acceptable vectors
%                    that are normal to the platform for all acceptable
%                    orientations
%*******************************************************************

%theta - pitch - rotation about y-axis
%psi - roll - rotation about x-axis
%phi - yaw - rotation about z-axis

%Initialize arrays that will track the x,y and z components of each
%acceptable vector that is normal to the platform so that they can be
%plotted using quiver
Cx = [];
Cy = [];
Cz = [];

%Initialize z unit vector along z-axis
Z = [0 0 1];

%Initialize the violation counts. These variables will track the number of
%violations that occur for the leg length, the alpha and beta angles and
%the constraint plane distance
lviol = 0;
O_Beta_Viol = 0;
O_Alph_Viol = 0;
dviol = 0;

%Initialize theta, phi and psi at zero to find the angles that
%the actuator legs make with the base and platform to be used in
%constraint equations later
theta = 0;
phi = 0;
psi = 0;

height = 31.75; %Initial height of the platform in cm

%Calculate initial angle between base and actuator leg and platform
%and actuator leg (alpha and beta)
if (theta == 0 && phi == 0 && psi == 0)

    %Elements of the rotatation matrix
    a11 = cos(theta)*cos(phi);
    a12 = cos(theta)*sin(phi);
    a13 = -sin(theta);
    a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
    a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
    a23 = cos(theta)*sin(psi);
    a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
    a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
    a33 = cos(theta)*cos(psi);

    %Rotation matrix from base to platform
    Rm = [a11 a12 a13;a21 a22 a23;a31 a32 a33];

    %Translation from {B} to {P} the last value is the height of the robot
    %(top of base to top of platform)
    q = [0 0 height]';

    %Rotation matrix from base to platform
    Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

    P1 = [R 0 0 1]'; %Platform joint locations in reference frame {P}
    P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
    P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

    B1 = [r 0 0]';%base joint locations in reference frame {B}
    B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
    B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

    u1 = Rm*P1; %vector from platform center to point 1 on platform
    u2 = Rm*P2;
    u3 = Rm*P3;

    l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
    l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
    l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

   OrigBaseAngle1 = asin(u1(3)/l1mag);
   OrigPlatAngle1 = acos(u1(3)/l1mag);

   OrigBaseAngle2 = asin(u2(3)/l2mag);
   OrigPlatAngle2 = acos(u2(3)/l2mag);

   OrigBaseAngle3 = asin(u3(3)/l3mag);
   OrigPlatAngle3 = acos(u3(3)/l3mag);
end

%Plot the vector normal to the platform for its initial configuration
quiver3(0,0,31.75,0,0,20)
hold

%Use three for loops to step through the orientational workspace and find
%all of the acceptable points using the constraint equations

for height = 32:0.2:42
    for theta = -pi/2:0.01:pi/2
        for phi = 0
            for psi =  -pi/2:0.01:pi/2

thetatrack = [thetatrack theta];
phitrack = [phitrack phi];

a11 = cos(theta)*cos(phi);
a12 = cos(theta)*sin(phi);
a13 = -sin(theta);
a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
a23 = cos(theta)*sin(psi);
a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
```

```
a33 = cos(theta)*cos(psi);

Rm = [a11 a12 a13;a21 a22 a23;a31 a32 a33]; %rotation matrix from base to
platform

q = [0 0 height]'; %translation from {B} to {P} the last value is the
height of the robot (top of base to top of platform)

Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];
%rotation matrix from base to platform

P1 = [R 0 0 1]'; %platform joint locations in reference frame {P}
P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

B1 = [r 0 0]';%base joint locations in reference frame {P}
B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

u1 = Rm*P1; %vector from platform center to point 1 on platform
u2 = Rm*P2;
u3 = Rm*P3;

l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

%Find the cross product between the Z-axis and the vector from the origin
%each of the points on the base. These are the normal vectors to the
%constraint planes
N1 = cross(Z,B1);
N2 = cross(Z,B2);
N3 = cross(Z,B3);

%Calculate the distance of each of the platform points form the constraint
%planes
D1 = abs(N1(1)*u1(1) + N1(2)*u1(2) + N1(3)*u1(3))/sqrt(N1(1)^2 + N1(2)^2 +
N1(3)^2);
D2 = abs(N2(1)*u2(1) + N2(2)*u2(2) + N2(3)*u2(3))/sqrt(N2(1)^2 + N2(2)^2 +
N2(3)^2);
D3 = abs(N3(1)*u3(1) + N3(2)*u3(2) + N3(3)*u3(3))/sqrt(N3(1)^2 + N3(2)^2 +
N3(3)^2);

%Check if the points satisfy all of the constraints such as the leg length
%constraint, the alpha and beta angle constraints and the maximum allowable
%distance from the constraint plane constraint
if (l1mag < 41.91 && l2mag < 41.91 && l3mag < 41.91 && l1mag > 31.75 &&
l2mag > 31.75 && l3mag > 31.75 ...
        && abs(asin(u1(3)/l1mag)-OrigBaseAngle1) < pi/4 && ...
        abs(acos(u1(3)/l1mag)-OrigPlatAngle1) < 0.44 ...
        && abs(asin(u2(3)/l2mag)-OrigBaseAngle2) < pi/4 && ...
        abs(acos(u2(3)/l2mag)-OrigPlatAngle2) < 0.44 ...
        && abs(asin(u3(3)/l1mag)-OrigBaseAngle3) < pi/4 && ...
        abs(acos(u3(3)/l3mag)-OrigPlatAngle3) < 0.44 ...
        && D1 < 0.25 && D2 < 0.25 && D3 < 0.25)

    %Use the platform points to find two vectors that lie on the platform
    crossvec1 = u1-u2;
    crossvec2 = u1-u3;
    crossvec1(4) = []; %prune the fourth element of the vector
    crossvec2(4) = [];

    %Calculate the two normal vectors and determine which one is the
    %postiive vector i.e. z element > 0
    C1 = cross(crossvec1,crossvec2);
    C2 = cross(crossvec2,crossvec1);

    if (C1(3) > 0)
        C = C1;
    elseif (C2(3) > 0)
        C = C2;
    end

    %Normalize the vector
    C_Norm = C/norm(C);
    %Multiply the normalized vector by 20 so that it is easily visible on
    %the plot
    C = 20*C_Norm;

    %Store the x, y and z elements of each normal vector for each
    %orientation
    Cx = [Cx C(1)];
    Cy = [Cy C(2)];
    Cz = [Cz C(3)];

end

    %For the platform points that were determined to be in violation of the
    %constraints, find which constraint was violated and increase the
    %number of violations by one for each time it is violated. This way we
    %can track how many violations there were of each constraint

if (l1mag > 41.91 || l2mag > 41.91 || l3mag > 41.91 || ...
      l1mag < 31.75 || l2mag < 31.75 || l3mag < 31.75)

    lviol = lviol+1;

end

if (abs(asin(u1(3)/l1mag)-OrigBaseAngle1) > pi/4 || ...
      abs(asin(u2(3)/l2mag)-OrigBaseAngle2) > pi/4 ...
        || abs(asin(u3(3)/l1mag)-OrigBaseAngle3) > pi/4)

        O_Beta_Viol = O_Beta_Viol+1;
end

if (abs(acos(u1(3)/l1mag)-OrigPlatAngle1) > 0.44 || ...
      abs(acos(u2(3)/l2mag)-OrigPlatAngle2) > 0.44 ...
    || abs(acos(u3(3)/l3mag)-OrigPlatAngle3) > 0.44)
        O_Alph_Viol = O_Alph_Viol+1;
end

if (D1 > 0.25 || D2 > 0.25 || D3 > 0.25)
    dviol = dviol+1;
end

            end
        end
    end

    %Check that there are values in the normal vector arrays and then
    %replicate the origin points (qx, qy, qz) so that they match the size
    %of the normal vector array. Then, use quiver 3 to make the quiver plot
    %and clear out the Cx Cy and Cz arrays and then repeat for each height
    if any(Cx) && any(Cy) && any(Cz)
        [Row,Col] = size(Cx);
        qx = repmat(0,1,Col);
        qy = repmat(0,1,Col);
        qz = repmat(height,1,Col);

        quiver3(qx,qy,qz,Cx,Cy,Cz)

        Cx = [];
        Cy = [];
        Cz = [];
    else
        Cx = [];
        Cy = [];
        Cz = [];
    end
end

%Plot all of the acceptable points on the same scatter plot and include the
%base points as a visual reference
scatter3(B1(1),B1(2),B1(3), 'MarkerEdgeColor','k','MarkerFaceColor', ...
[0 .75 .75])
scatter3(B2(1),B2(2),B2(3), 'MarkerEdgeColor','k','MarkerFaceColor', ...
[0 .75 .75])
scatter3(B3(1),B3(2),B3(3), 'MarkerEdgeColor','k','MarkerFaceColor', ...
[0 .75 .75])
```

```
function [alph, bestmaxworkspace] = CRS_Delta
%********************************************************************
%CRS_Delta.m - This script generates 4 parameters that will be optimized
%              in order to maximize the workspace volume of the delta
%              robot. It performs 10 generations of the controlled random
%              search to optimize these values and then returns the best
%              values that are found and the largest workspace found.
%********************************************************************

a = 0.5*rand; %generate a random number for a that is no higher than 0.5

d = 0.01*rand; %generate a random number for d that is no higher than 0.01

b = 1-a-d; %keep normalization constant a+b+d=1

%Set the phi values for the 120 degree evenly space top arm joint of the
%delta robot in the {O} reference frame
phi1 = 0;
phi2 = 2*pi/3;
phi3 = 4*pi/3;

%Initialize the number of generations to zero
gens = 0;

%Set the K1 and K2 constants according the Lou et al. (K2 not used)
K1 = 1/3;
K2 = 1/2;

%Initialize the maximum workspace variable
maxworkspace = 0;

%Initialize the besta, bestb and bestd variable to be equal to the first
%generated number
besta = a;
bestb = b;
bestd = d;

%Set the initial step direction variable
sigma_a = 1;
sigma_zc = 1;

%Initialize the best maximum workspace variable to zero
bestmaxworkspace = 0;

%Set x and y to zero so that an acceptable zc can be found
x = 0;
y = 0;

%Set zc to zero so that the while loop will not be skipped
zc = 0;

while zc == 0
            zc = -rand; %Generate a zc value in the interval [-1,0]

            alph = [a b d zc] %Put initial alpha guess population together

            %Variables from the equation: Xi*cos(theta1) + z*sin(theta1) =
Qi;
```

```
Xi =  x - d;
Qi = (Xi^2 + a^2 + y^2 + zc^2 - b^2)/2*a;

%Separate the variables so that a quadratic eqn is formed
ElemA = Xi^2 + zc^2;
ElemB = -2*Qi;
ElemC = Qi^2 - zc^2;

%Use roots to solve for the roots fo the polynomial
poly1 = [ElemA ElemB ElemC];

r1 = roots(poly1);

%Check if and which roots are real, if none are real then set
%to -40 so that the constraints are not satisfied and the point
%is discarded
if isreal(acosd(r1(1)))
    theta = acosd(r1(1));
elseif isreal(acosd(r1(2)))
    theta = acosd(r1(2));
else
    theta = -40;
end

%Calculate beta based on geometry
beta = acosd((a*cos(theta)+d)/b);

%Calculate the constraint
constraint = theta+beta;

%Calculate the three different gamma values for each arm
gamma1 = acosd((x*sin(phi1)+y*cos(phi1))/b) - 90;
gamma2 = acosd((x*sin(phi2)+y*cos(phi2))/b) - 90;
gamma3 = acosd((x*sin(phi3)+y*cos(phi3))/b) - 90;

%The next section of code calculates the manipulability
%constraint
h1 = (x*cos(phi1)+y*sin(phi1)-d)*a*sin(theta)+zc*a*cos(theta);
h2 = (x*cos(phi2)+y*sin(phi2)-d)*a*sin(theta)+zc*a*cos(theta);
h3 = (x*cos(phi3)+y*sin(phi3)-d)*a*sin(theta)+zc*a*cos(theta);

Jtheta = -[h1 0 0;0 h2 0;0 0 h3];

v1 = [x-(d+a*cos(theta))*cos(phi1), ...
       y-(d+a*cos(theta))*sin(phi1), zc+a*sin(theta)];
v2 = [x-(d+a*cos(theta))*cos(phi2), ...
       y-(d+a*cos(theta))*sin(phi2), zc+a*sin(theta)];
v3 = [x-(d+a*cos(theta))*cos(phi3), ...
       y-(d+a*cos(theta))*sin(phi3), zc+a*sin(theta)];

Jx = [v1;v2;v3];

J = inv(Jx)*Jtheta;

singmin = svds(J,1,0);
singmax = svds(J,1);

kai = singmin/singmax;
```

```
            %Check that all values are real and check to see if the
            %constraints are satisfied. If they are all satisfied, break
            %out of the loop and use that value as zc
            if isreal(gamma1) && isreal(gamma2) && isreal(gamma3) ...
               && isreal(kai) && isreal(constraint) ...
               && isreal(theta) && isfinite(constraint) ...
               && isfinite(theta) && isempty(kai) == 0 ...
                if constraint <= 180 && gamma1 >= -40 && gamma1 <= 40 ...
                   && gamma2 >= -40 && gamma2 <= 40 ...
                   && gamma3 >= -40 && gamma3 <= 40 ...
                   && kai >= 0.4

                    break

                else
                    zc = 0;
                end
            end
end

%Set the best zc to the current zc
bestzc = zc;

%Terminate the algorithm after 10 generations
while gens < 10

    gens = gens + 1;

%For different side lengths of the cube, calculate the vertices of the of
%the cube and determine if the constraints are satisfied for each workspace
%volume as defined by the cube. If they are satisfied, then that is the
%largest workspace for the parameters that were generated
for r = 1:-0.01:0
    for i = 1:1:8
            vc1 = [-r -r zc-r]; %cube vertices
            vc2 = [r -r zc-r];
            vc3 = [r r zc-r];
            vc4 = [-r r zc-r];
            vc5 = [-r -r zc+r];
            vc6 = [r -r zc+r];
            vc7 = [r r zc+r];
            vc8 = [-r r zc+r];

            cube = {vc1 vc2 vc3 vc4 vc5 vc6 vc7 vc8};

            x = cube{i}(1);
            y = cube{i}(2);
            z = cube{i}(3);

            Xi =  x - d;
            Qi = (Xi^2 + a^2 + y^2 + z^2 - b^2)/2*a;

            ElemA = Xi^2 + z^2;
            ElemB = -2*Qi;
            ElemC = Qi^2 - z^2;

            poly1 = [ElemA ElemB ElemC];
```

```
        r1 = roots(poly1);

        if isreal(acosd(r1(1)))
            theta = acosd(r1(1));
        elseif isreal(acosd(r1(2)))
            theta = acosd(r1(2));
        else
            theta = -40;
        end

        beta = acosd((a*cos(theta)+d)/b);

        constraint = theta+beta;

        gamma1 = acosd((x*sin(phi1)+y*cos(phi1))/b) - 90;
        gamma2 = acosd((x*sin(phi2)+y*cos(phi2))/b) - 90;
        gamma3 = acosd((x*sin(phi3)+y*cos(phi3))/b) - 90;

        h1 = (x*cos(phi1)+y*sin(phi1)-d)*a*sin(theta)+z*a*cos(theta);
        h2 = (x*cos(phi2)+y*sin(phi2)-d)*a*sin(theta)+z*a*cos(theta);
        h3 = (x*cos(phi3)+y*sin(phi3)-d)*a*sin(theta)+z*a*cos(theta);

        Jtheta = -[h1 0 0;0 h2 0;0 0 h3];

        v1 = [x-(d+a*cos(theta))*cos(phi1), ...
               y-(d+a*cos(theta))*sin(phi1), z+a*sin(theta)];
        v2 = [x-(d+a*cos(theta))*cos(phi2), ...
               y-(d+a*cos(theta))*sin(phi2), z+a*sin(theta)];
        v3 = [x-(d+a*cos(theta))*cos(phi3), ...
               y-(d+a*cos(theta))*sin(phi3), z+a*sin(theta)];

        Jx = [v1;v2;v3];

        J = inv(Jx)*Jtheta;

        singmin = svds(J,1,0);
        singmax = svds(J,1);

        kai = singmin/singmax;

          if isreal(gamma1) && isreal(gamma2) && isreal(gamma3) ...
               && isreal(kai) && isreal(constraint) ...
              && isreal(theta) && isfinite(constraint)&& ...
              ...isfinite(theta) && isempty(kai) == 0 ...
            if constraint <= 180 && gamma1 >= -40 && gamma1 <= 40 ...
               && gamma2 >= -40 && gamma2 <= 40 ...
               && gamma3 >= -40 && gamma3 <= 40 ...
               && kai >= 0.4

           %Calculate the volume of the cube based on the r value and
           %if that workspace is the largest that has been found, store
           %that as the max workspace
               workspace = (2*r)^3;
                        if workspace >= maxworkspace
                            maxworkspace = workspace;
                        end
            end
        end
end
```

```
end

%if the max workspace found during this run is larger than the
%current best maximum workspace then make that the new best and
%record the a b and d values for the maximum
if maxworkspace > bestmaxworkspace
      bestmaxworkspace = maxworkspace;
      besta = a;
      bestb = b;
      bestd = d;
      bestzc = zc;
end

%Use the controlled random search algorithm to find the next points that
%will be generated
   zeta = normrnd(0,1);

   a = besta + sigma_a*zeta;

   zc = bestzc + sigma_zc*zeta;

   b = 1-a-d;

   if a < 0.2 || b < 0
       a = besta;
       b = 1-a-d;
   end

   if zc > 0 || zc < -1
       zc = bestzc;
   end

   upperbound_a = 1 - a;
   lowerbound_a = a - 0;

   if lowerbound_a > upperbound_a
        deltasigma_a = upperbound_a;
   else
        deltasigma_a = lowerbound_a;
   end

   sigma_a = K1*deltasigma_a;

   upperbound_zc = 1 - z;
   lowerbound_zc = z - 0;

   if lowerbound_zc > upperbound_zc
        deltasigma_zc = upperbound_zc;
   else
        deltasigma_zc = lowerbound_zc;
   end

   sigma_zc = K1*deltasigma_zc;
end

%Use the best values found to return to the user and to subsequently be
%used for the proposed algorithm
```

```
alph = [besta bestb bestd bestzc]

function [optimumworkspace,alpha] = PSO_Delta
%*******************************************************************
%PSO_Delta.m - This script generates 4 paramters that will be optimized
%              in order to maximize the workspace volume of the delta
%              robot. It performs 10 generations of the controlled random
%              search to optimize these values and then returns the best
%              values that are found and the largest workspace found.
%*******************************************************************

%The majority of this code is the same as the CRS_Delta.m code except that
%instead of one point for each generation, there is a swarm of 20 particles
%that are continually be compared against one another and updated according
%to the PSO algorithm. Please refer to CRS_Delta for any clarification of
%the code through comments.

a = 0.5*rand(20,1);

d = 0.01*rand(20,1);

b = 1-d-a;

z = zeros(20,1);
zc = zeros(20,1);

maxworkspace = zeros(20,1);
oldmaxworkspace = zeros(20,1);
bestmaxworkspace = 0;
Z_Max = zeros(20,1);

pbest = zeros(20,3);

phi1 = 0;
phi2 = 2*pi/3;
phi3 = 4*pi/3;

gens = 0;

x = 0;
y = 0;
Z_Max = zeros(20,1);

while max(maxworkspace) < 0.125

            gens = gens + 1;
            for i = 1:1:20
                while z(i) == 0

                    z(i) = -rand;

                    Xi =  x - d(i);
                    Qi = (Xi^2 + a(i)^2 + y^2 + z(i)^2 - b(i)^2)/2*a(i);


                    ElemA = Xi^2 + z(i)^2;
                    ElemB = -2*Qi;
                    ElemC = Qi^2 - z(i)^2;
```

```
    poly1 = [ElemA ElemB ElemC];

    r1 = roots(poly1);

    if isreal(acosd(r1(1)))
        theta = acosd(r1(1));
    elseif isreal(acosd(r1(2)))
        theta = acosd(r1(2));
    else
        theta = -40;
    end

    beta = acosd((a(i)*cos(theta)+d(i))/b(i));

    constraint = theta+beta;

    gamma1 = acosd((x*sin(phi1)+y*cos(phi1))/b(i)) - 90;
    gamma2 = acosd((x*sin(phi2)+y*cos(phi2))/b(i)) - 90;
    gamma3 = acosd((x*sin(phi3)+y*cos(phi3))/b(i)) - 90;

    h1 = (x*cos(phi1)+y*sin(phi1) ...
        -d(i))*a(i)*sin(theta)+z(i)*a(i)*cos(theta);
    h2 = (x*cos(phi2)+y*sin(phi2) ...
        -  d(i))*a(i)*sin(theta)+z(i)*a(i)*cos(theta);
    h3 = (x*cos(phi3)+y*sin(phi3) ...
        -d(i))*a(i)*sin(theta)+z(i)*a(i)*cos(theta);

    Jtheta = -[h1 0 0;0 h2 0;0 0 h3];

    v1 = [x-(d(i)+a(i)*cos(theta))*cos(phi1), ...
        y-(d(i)+a(i)*cos(theta))*sin(phi1), ...
        z(i)+a(i)*sin(theta)];
    v2 = [x-(d(i)+a(i)*cos(theta))*cos(phi2), ...
        y-(d(i)+a(i)*cos(theta))*sin(phi2), ...
        z(i)+a(i)*sin(theta)];
    v3 = [x-(d(i)+a(i)*cos(theta))*cos(phi3), ...
              y-(d(i)+a(i)*cos(theta))*sin(phi3), ...
              z(i)+a(i)*sin(theta)];

    Jx = [v1;v2;v3];

    J = inv(Jx)*Jtheta;

    singmin = svds(J,1,0);
    singmax = svds(J,1);

    kai = singmin/singmax;

  if isreal(gamma1) && isreal(gamma2) && isreal(gamma3)...
        && isreal(kai) && isreal(constraint) ...
        && isreal(theta) && isfinite(constraint) && ...
        isfinite(theta) && isempty(kai) == 0 ...

        if constraint <= 180 && gamma1 >= -40 && ...
              gamma1 <= 40 ...
             && gamma2 >= -40 && gamma2 <= 40 ...
             && gamma3 >= -40 && gamma3 <= 40 ...
```

```
                            && kai >= 0.4

                            zc(i) = z(i);
                            break
                        else
                            z(i) = 0;
                        end
                    end
                end
            end

alpha = [a b d zc] %%put initial alpha guess population together

for i = 1:1:20
    for r = 1:-0.01:0
        for j = 1:1:8
                v1 = [-r -r zc(i)-r]; %cube vertices
                v2 = [r -r zc(i)-r];
                v3 = [r r zc(i)-r];
                v4 = [-r r zc(i)-r];
                v5 = [-r -r zc(i)+r];
                v6 = [r -r zc(i)+r];
                v7 = [r r zc(i)+r];
                v8 = [-r r zc(i)+r];

                cube = {v1 v2 v3 v4 v5 v6 v7 v8};

                x = cube{j}(1);
                y = cube{j}(2);
                z = cube{j}(3);

                Xi =  x - d(i);
                Qi = (Xi^2 + a(i)^2 + y^2 + z^2 - b(i)^2)/2*a(i);

                ElemA = Xi^2 + z^2;
                ElemB = -2*Qi;
                ElemC = Qi^2 - z^2;

                poly1 = [ElemA ElemB ElemC];

                r1 = roots(poly1);

                if isreal(acosd(r1(1)))
                    theta = acosd(r1(1));
                elseif isreal(acosd(r1(2)))
                    theta = acosd(r1(2));
                else
                    theta = -40;
                end

                beta = acosd((a(i)*cos(theta)+d(i))/b(i));

                constraint = theta+beta;

                gamma1 = acosd((x*sin(phi1)+y*cos(phi1))/b(i)) - 90;
                gamma2 = acosd((x*sin(phi2)+y*cos(phi2))/b(i)) - 90;
                gamma3 = acosd((x*sin(phi3)+y*cos(phi3))/b(i)) - 90;
                h1 = (x*cos(phi1)+y*sin(phi1)-
d(i))*a(i)*sin(theta)+z*a(i)*cos(theta);
                h2 = (x*cos(phi2)+y*sin(phi2)-
d(i))*a(i)*sin(theta)+z*a(i)*cos(theta);
                h3 = (x*cos(phi3)+y*sin(phi3)-
d(i))*a(i)*sin(theta)+z*a(i)*cos(theta);

                Jtheta = -[h1 0 0;0 h2 0;0 0 h3];

                v1 = [x-(d(i)+a(i)*cos(theta))*cos(phi1) ...
                  ,y-(d(i)+a(i)*cos(theta))*sin(phi1), z+a(i)*sin(theta)];
                v2 = [x-(d(i)+a(i)*cos(theta))*cos(phi2), ...
                   y-(d(i)+a(i)*cos(theta))*sin(phi2), z+a(i)*sin(theta)];
                v3 = [x-(d(i)+a(i)*cos(theta))*cos(phi3), ...
                  y-(d(i)+a(i)*cos(theta))*sin(phi3), z+a(i)*sin(theta)];

                Jx = [v1;v2;v3];

                J = inv(Jx)*Jtheta;

                singmin = svds(J,1,0);
                singmax = svds(J,1);

                kai = singmin/singmax;

                if isreal(gamma1) && isreal(gamma2) && isreal(gamma3)
                   && isreal(kai) && isreal(constraint) ...
                       && isreal(theta) && isfinite(constraint) ...
                       && isfinite(theta) && isempty(kai) == 0
                    if theta >= -30 && theta <= 100 && constraint >= 45 ...
                        && constraint <= 180 && gamma1 >= -40 && ...
                        gamma1 <= 40 ...
                       && gamma2 >= -40 && gamma2 <= 40 ...
                       && gamma3 >= -40 && gamma3 <= 40 ...
                       && kai >= 0.4

                       workspace(i) = (2*r)^3;
                                if workspace(i) >= maxworkspace(i)
                                    maxworkspace(i) = workspace(i);
                                end
                    end
                end
        end
    end
end

%Stores the best workspace found overall and stores the parameters that
%give that workspace
for i=1:1:20
    if maxworkspace(i) > bestmaxworkspace && a(i) > 0 && b(i) > 0 ...
      && d(i) > 0
        bestmaxworkspace = maxworkspace(i);
        gbest = [a(i) b(i) d(i) zc(i)];
    end
end

%Resize gbest to 20 columns to make calculations easier
gbestorig = gbest;
gbest = repmat(gbest,20,1);
```

```
for i = 1:1:20
    if maxworkspace(i) >= oldmaxworkspace(i)
            pbest(i,1) = a(i);
            pbest(i,2) = b(i);
            pbest(i,3) = d(i);
            pbest(i,4) = zc(i);
    end
end

c1 = 0.5; %C1 and C2 are both values that are used by Lou et al.
c2 = 1.25;
vmax = [];

%No intial velocity given in paper so it is set to be very small
v_a = repmat(0.0001,20,1);
v_b = repmat(0.0001,20,1);
v_zc = repmat(0.0001,20,1);

%Calculates the velocities of each of the paramters to update them for the
%next generation according to the PSO algorithm

for i = 1:1:20
    v_a(i) = v_a(i) + c1*rand*(pbest(i,1)-alpha(i,1)) ...
      +c2*rand*(gbest(i,1)-alpha(i,1));
    v_b(i) = v_b(i) + c1*rand*(pbest(i,2)-alpha(i,2)) ...
      +c2*rand*(gbest(i,2)-alpha(i,2));
    v_zc(i) = v_zc(i) + c1*rand*(pbest(i,4)-alpha(i,4)) ...
      +c2*rand*(gbest(i,4)-alpha(i,4));
end

%Resize gbest to one column
gbest = gbestorig;

%Store the previous values for each parameter and workspace in case the
%values generated by the velocities are not within in the bounds of the
%parameters or if they violate the normalization
olda = a;
oldb = b;
oldd = d;
oldzc = zc;
oldalpha = alpha;
oldmaxworkspace = maxworkspace;

newa = a+v_a;

%Make sure that the velocities aren't causing the bounds of the parameters
%or the normalization to be violated

for i=1:1:20
    if newa(i) < 0 || newa(i) > 1
        a(i) = olda(i);
    else
        a(i) = newa(i);
    end
end

newzc = zc+v_zc;
```

```
for i=1:1:20
    if newzc(i) < -1 || newzc(i) > 0
        z(i) = oldzc(i);
    else
        z(i) = newzc(i);
    end
end

newb = b+v_b;

for i=1:1:20
    if newb(i) < 0 || newb(i) > 1
        b(i) = oldb(i);
    else
        b(i) = newb(i);
    end
end

d = 1-a-b;
b = newb;

newalpha = [a b d zc];
alpha = newalpha;

end

%Output the best values found and maximum workspace found
alpha = gbest(1,1:4);

optimumworkspace = max(maxworkspace);

function [optimumworkspace,alph] = Project_Delta
%*********************************************************************
%Project_Delta.m - This script calls CRS_delta and then uses the value that
%                    returned by this function to populate 5 of the 20
%                    values in the initial swarm array. The other 15 values
%                    are randomly generated within the bound of the
%                    parameters. Then PSO is used to optimize the
%                    parameters. The function returns the best parameters
%                    found and the optimum workspace found
%*********************************************************************

%Get the output of the CRS algorithm
alph = CRS_Delta();

%Replicate the values so that there are 5 rows of the same value for each
%parameter
a = alph(1);
b = alph(2);
d = alph(3);
z = alph(4);

a1 = repmat(a,5,1);
a2 = 0.5*rand(15,1);

%Put the randomly generated values and the CRS values together to make one
%matrix of 20 particles
a = [a1; a2];
```

```
d1 = repmat(d,5,1);
d2 = 0.01*rand(15,1);

d = [d1; d2];

b = 1-d-a;

z1 = repmat(z,5,1);
z2 = -rand(15,1);

zc = [z1; z2];

maxworkspace = zeros(20,1);
oldmaxworkspace = zeros(20,1);
bestmaxworkspace = 0;

pbest = zeros(20,3);

phi1 = 0;
phi2 = 2*pi/3;
phi3 = 4*pi/3;

gens = 0;

alph = [a b d zc]

while max(maxworkspace) < 0.125

gens = gens + 1;

for i = 1:1:20
    for r = 1:-0.01:0
        for j = 1:1:8
                v1 = [-r -r zc(i)-r]; %cube vertices
                v2 = [r -r zc(i)-r];
                v3 = [r r zc(i)-r];
                v4 = [-r r zc(i)-r];
                v5 = [-r -r zc(i)+r];
                v6 = [r -r zc(i)+r];
                v7 = [r r zc(i)+r];
                v8 = [-r r zc(i)+r];

                cube = {v1 v2 v3 v4 v5 v6 v7 v8};

                x = cube{j}(1);
                y = cube{j}(2);
                z = cube{j}(3);

                Xi =  x - d(i);
                Qi = (Xi^2 + a(i)^2 + y^2 + z^2 - b(i)^2)/2*a(i);

                ElemA = Xi^2 + z^2;
                ElemB = -2*Qi;
                ElemC = Qi^2 - z^2;

                poly1 = [ElemA ElemB ElemC];

                r1 = roots(poly1);
                if isreal(acosd(r1(1)))
                    theta = acosd(r1(1));
                elseif isreal(acosd(r1(2)))
                    theta = acosd(r1(2));
                else
                    theta = -40;
                end

                beta = acosd((a(i)*cos(theta)+d(i))/b(i));

                constraint = theta+beta;

                gamma1 = acosd((x*sin(phi1)+y*cos(phi1))/b(i)) - 90;
                gamma2 = acosd((x*sin(phi2)+y*cos(phi2))/b(i)) - 90;
                gamma3 = acosd((x*sin(phi3)+y*cos(phi3))/b(i)) - 90;

                h1 = (x*cos(phi1)+y*sin(phi1)-d(i))*a(i)*sin(theta) ...
                    +z*a(i)*cos(theta);
                h2 = (x*cos(phi2)+y*sin(phi2)-d(i))*a(i)*sin(theta) ...
                    +z*a(i)*cos(theta);
                h3 = (x*cos(phi3)+y*sin(phi3)-d(i))*a(i)*sin(theta) ...
                    +z*a(i)*cos(theta);

                Jtheta = -[h1 0 0;0 h2 0;0 0 h3];

                v1 = [x-(d(i)+a(i)*cos(theta))*cos(phi1), ...
                    y-(d(i)+a(i)*cos(theta))*sin(phi1), z+a(i)*sin(theta)];
                v2 = [x-(d(i)+a(i)*cos(theta))*cos(phi2), ...
                    y-(d(i)+a(i)*cos(theta))*sin(phi2), z+a(i)*sin(theta)];
                v3 = [x-(d(i)+a(i)*cos(theta))*cos(phi3), ...
                    y-(d(i)+a(i)*cos(theta))*sin(phi3), z+a(i)*sin(theta)];

                Jx = [v1;v2;v3];

                J = inv(Jx)*Jtheta;

                singmin = svds(J,1,0);
                singmax = svds(J,1);

                kai = singmin/singmax;

                if isreal(gamma1) && isreal(gamma2) && isreal(gamma3) ...
                        && isreal(kai) && isreal(constraint) ...
                        && isreal(theta) && isfinite(constraint) ...
                        && isfinite(theta) && isempty(kai) == 0 ...
                    if constraint <= 180 && gamma1 >= -40 && gamma1 <= 40
...

                       && gamma2 >= -40 && gamma2 <= 40 ...
                       && gamma3 >= -40 && gamma3 <= 40 ...
                       && kai >= 0.4

                       workspace(i) = (2*r)^3;
                                if workspace(i) >= maxworkspace(i)
                                    maxworkspace(i) = workspace(i);
                                end
                    end
                end
        end
```

```
    end
end

for i=1:1:20
    if maxworkspace(i) > bestmaxworkspace && a(i) > 0 && b(i) > 0 ...
            && d(i) > 0
        bestmaxworkspace = maxworkspace(i);
        gbest = [a(i) b(i) d(i) zc(i)];
    end
end

gbestorig = gbest;
gbest = repmat(gbest,20,1);

for i = 1:1:20
    if maxworkspace(i) >= oldmaxworkspace(i)
            pbest(i,1) = a(i);
            pbest(i,2) = b(i);
            pbest(i,3) = d(i);
            pbest(i,4) = zc(i);
    end
end

c1 = 0.5;
c2 = 1.25;
vmax = [];
v_a = repmat(0.0001,20,1);
v_b = repmat(0.0001,20,1);
v_zc = repmat(0.0001,20,1);

for i = 1:1:20
    v_a(i) = v_a(i) + c1*rand*(pbest(i,1)-alph(i,1))+ ...
        c2*rand*(gbest(i,1)-alph(i,1));
    v_b(i) = v_b(i) + c1*rand*(pbest(i,2)-alph(i,2))+ ...
        c2*rand*(gbest(i,2)-alph(i,2));
    v_zc(i) = v_zc(i) + c1*rand*(pbest(i,4)-alph(i,4))+ ...
        c2*rand*(gbest(i,4)-alph(i,4));
end

gbest = gbestorig;

olda = a;
oldb = b;
oldd = d;
oldzc = zc;
oldalpha = alph;
oldmaxworkspace = maxworkspace;

newa = a+v_a;

for i=1:1:20
    if newa(i) < 0 || newa(i) > 1
        a(i) = olda(i);
    else
        a(i) = newa(i);
    end
end

newzc = zc+v_zc;
```

```
for i=1:1:20
    if newzc(i) < -1 || newzc(i) > 0
        z(i) = oldzc(i);
    else
        z(i) = newzc(i);
    end
end

newb = b+v_b;

for i=1:1:20
    if newb(i) < 0 || newb(i) > 1
        b(i) = oldb(i);
    else
        b(i) = newb(i);
    end
end

b = newb;

d = 1-a-b;

newalpha = [a b d zc];
alph = newalpha

maxworkspace

end

alph = gbest(1,1:4)

optimumworkspace = max(maxworkspace)

function [alph, bestcircle_r] = CRS_Stewart

%*******************************************************************
%CRS_Stewart.m - This script generates 2 parameters that will be optimized
%               in order to maximize the orientational workspace of the
%               Stewart represented as a the maximum radius of a circular
%               trajectory that the platform can follow. It performs
%               10 generations of the controlled random search to optimize
%               these values and then returns the best values that are
%               found and largest circular trajectory radius found.
%*******************************************************************

%Most of this code is a combination of the CRS_Delta function and the
%Workspace_Points function. Therefore, much of it has already been
%commented. However, anything that is unique to this function will be
%explained in the comments in this code

%Generate a random number between 5 and 10 as the initial guess for the
%radius of the base plate
R = 5 + (12-5).*rand;

%Generate a random number between 20 and 25 as the initial guess for the
%radius of the platform or top plate
r = 18 + (25-18).*rand;
```

```
gens = 0;

K1 = 1/3;
K2 = 1/2;

maxworkspace = 0;
bestmaxworkspace = 0;

%The bestR and bestr values are initialized to be the initial parameters
%generated
bestR = R;
bestr = r;

sigma_R = 1;
sigma_r = 1;

theta = 0;
phi = 0;
psi = 0;

%The maximum height of the robot is 41.91 cm and the minimum height of the
%robot is 31.75 cm, as dictated by the length of the actuator housing and
%the stroke of the actuators
max_h = 41.91;
min_h = 31.75;

%The orientational workspace occurs near the middle of the height range.
%So, the height of the robot was fixed at this level
mid_h = (max_h + min_h)/2;

%Spray circle trajectory center 50 cm above platform is located 50 cm
%above the mid_h height of the platform
circle_h = mid_h + 50;

%Values for the circle trajectory are initialized
circle_r = 0;
bestcircle_r = 0;

Z = [0 0 1];

alpha = [R r]

%The algorithm terminates after 10 generations or when the radius of the
%largest trajectory exceeds 26.7 cm

while gens < 10 || bestcircle_r < 26.7

    P1 = [R 0 0 1]';
    P2 = [(-1/2)*R (sqrt(3)/2)*R 0 1]';
    P3 = [(-1/2)*R -(sqrt(3)/2)*R 0 1]';

    B1 = [r 0 0]';
    B2 = [(-1/2)*r (sqrt(3)/2)*r 0]';
    B3 = [(-1/2)*r -(sqrt(3)/2)*r 0]';

    a11 = cos(theta)*cos(phi);
    a12 = cos(theta)*sin(phi);
    a13 = -sin(theta);
    a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
    a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
    a23 = cos(theta)*sin(psi);
    a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
    a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
    a33 = cos(theta)*cos(psi);

    q = [0 0 mid_h]';

    Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

    u1 = Rm*P1;
    u2 = Rm*P2;
    u3 = Rm*P3;

    l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
    l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
    l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

    OrigBaseAngle1 = asin(u1(3)/l1mag);
    OrigPlatAngle1 = acos(u1(3)/l1mag);

    OrigBaseAngle2 = asin(u2(3)/l2mag);
    OrigPlatAngle2 = acos(u2(3)/l2mag);

    OrigBaseAngle3 = asin(u3(3)/l3mag);
    OrigPlatAngle3 = acos(u3(3)/l3mag);

    gens = gens + 1;

%The radius of the circular trajectory starts at an adequately large size
%of 30 cm and is decreased by 0.1 cm at each step. The radius is determined
%to be the largest acceptable radius if the normal vector originating from
%the center of the platform contacts the circular trajectory at all eight
%points on the circle, which are spaced 45 degrees apart from one another
for circle_r = 30:-0.1:0

    valid1 = 0;
    valid2 = 0;
    valid3 = 0;
    valid4 = 0;
    valid5 = 0;
    valid6 = 0;
    valid7 = 0;
    valid8 = 0;

    for theta = -pi/2:0.025:pi/2
            for psi =  -pi/2:0.025:pi/2
                V1 = [circle_r 0 circle_h]'; %Points on the circle
                V2 = [circle_r*1/sqrt(2) circle_r*1/sqrt(2) circle_h]';
                V3 = [0 circle_r circle_h]';
                V4 = [-circle_r*1/sqrt(2) circle_r*1/sqrt(2) circle_h]';
                V5 = [-circle_r 0 circle_h]';
                V6 = [-circle_r*1/sqrt(2) -circle_r*1/sqrt(2) circle_h]';
                V7 = [0 -circle_r circle_h]';
                V8 = [circle_r*1/sqrt(2) -circle_r*1/sqrt(2) circle_h]';

                V = {V1 V2 V3 V4 V5 V6 V7 V8};
                a11 = cos(theta)*cos(phi);
                a12 = cos(theta)*sin(phi);
                a13 = -sin(theta);
                a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
                a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
                a23 = cos(theta)*sin(psi);
                a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
                a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
                a33 = cos(theta)*cos(psi);

                q = [0 0 mid_h]';

                Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2); ...
                    a31 a32 a33 q(3);0 0 0 1];

                u1 = Rm*P1;
                u2 = Rm*P2;
                u3 = Rm*P3;

                l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-
B1(3))^2);
                l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-
B2(3))^2);
                l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-
B3(3))^2);

                %The initial base angle is calculated at this point
                if theta == 0 && psi == 0
                    OrigBaseAngle1 = asin(u1(3)/l1mag);
                    OrigPlatAngle1 = acos(u1(3)/l1mag);

                    OrigBaseAngle2 = asin(u2(3)/l2mag);
                    OrigPlatAngle2 = acos(u2(3)/l2mag);

                    OrigBaseAngle3 = asin(u3(3)/l3mag);
                    OrigPlatAngle3 = acos(u3(3)/l3mag);
                end

                N1 = cross(Z,B1);
                N2 = cross(Z,B2);
                N3 = cross(Z,B3);

                D1 = abs(N1(1)*u1(1) + N1(2)*u1(2) + ...
                    N1(3)*u1(3))/sqrt(N1(1)^2 + N1(2)^2 + N1(3)^2);
                D2 = abs(N2(1)*u2(1) + N2(2)*u2(2) + ...
                    N2(3)*u2(3))/sqrt(N2(1)^2 + N2(2)^2 + N2(3)^2);
                D3 = abs(N3(1)*u3(1) + N3(2)*u3(2) + ...
                    N3(3)*u3(3))/sqrt(N3(1)^2 + N3(2)^2 + N3(3)^2);

                    if (l1mag < 41.91 && l2mag < 41.91 && ...
                            l3mag < 41.91 && l1mag > 31.75 && ...
                            l2mag > 31.75 && l3mag > 31.75 ...
                            && abs(asin(u1(3)/l1mag)-OrigBaseAngle1) < pi/4
...
                            && abs(acos(u1(3)/l1mag)-OrigPlatAngle1) < 0.44
...
                            && abs(asin(u2(3)/l2mag)-OrigBaseAngle2) < pi/4
...
```

```
                            && abs(acos(u2(3)/l2mag)-OrigPlatAngle2) < 0.44
...
                            && abs(asin(u3(3)/l1mag)-OrigBaseAngle3) < pi/4
...
                            && abs(acos(u3(3)/l3mag)-OrigPlatAngle3) < 0.44
...
                            && D1 < 0.25 && D2 < 0.25 && D3 < 0.25)

                        crossvec1 = u1-u2; %find the vector normal to the
platform

                        crossvec2 = u1-u3;
                        crossvec1(4) = [];
                        crossvec2(4) = [];
                        C1 = cross(crossvec1,crossvec2);
                        C2 = cross(crossvec2,crossvec1);

                        if (C1(3) > 0) %check which vector normal is the
positive vector

                             C = C1;
                        elseif (C2(3) > 0)
                             C = C2;
                        end

                        C_Norm1 = C/norm(C);
                        C_Norm2 = 2*C_Norm1;

                        %D_v is a formula which calculates the distance
from

                        %a line to a point. This is used because the
likelihood

                        %of a line contacting one of the points defining
                        %the circle at a step size of 0.025 radians is
                        %extremely low and even lower for 8 points.
                        %Therefore, if the distance is within an acceptable
                        %amount, then the normal vector is within an
                        %acceptable distance from the circular trajectory
                        %to consider it to have contacted it.
                        for j = 1:1:8
                            d_v = norm(cross(V{j}-C_Norm1,V{j}-C_Norm2))...
                                /norm(C_Norm2-C_Norm1);
                            if d_v <= 1.5

                                if j == 1
                                    valid1 = 1;
                                elseif j == 2
                                    valid2 = 1;
                                elseif j == 3
                                    valid3 = 1;
                                elseif j == 4
                                    valid4 = 1;
                                elseif j == 5
                                    valid5 = 1;
                                elseif j == 6
                                    valid6 = 1;
                                elseif j == 7
                                    valid7 =1;
                                elseif j == 8
                                    valid8 = 1;
                                end
                            end
                        end
                    end
            end
    end

    valid = valid1 + valid2 + valid3 + valid4 + valid5 + valid6 + valid7 +
valid8;

    if valid == 8
        valid1 = 0;
        valid2 = 0;
        valid3 = 0;
        valid4 = 0;
        valid5 = 0;
        valid6 = 0;
        valid7 = 0;
        valid8 = 0;
        break
    end

end


%If the radius of the circle found during this generation is larger than
%the best one previously found then store that value and store the
%paramters R and r that go with that radius
if circle_r > bestcircle_r
   bestcircle_r = circle_r;
   bestr = r;
   bestR = R;
end

   zeta = normrnd(0,1);

   r = bestr + sigma_r*zeta;

   R = bestR + sigma_R*zeta;

   upperbound_r = 25 - r;
   lowerbound_r = r - 18;

   if lowerbound_r > upperbound_r
        deltasigma_r = upperbound_r;
   else
        deltasigma_r = lowerbound_r;
   end

   sigma_r = K1*deltasigma_r;

   upperbound_R = 12 - R;
   lowerbound_R = R - 5;

   if lowerbound_R > upperbound_R
        deltasigma_R = upperbound_R;
   else
        deltasigma_R = lowerbound_R;
   end
   sigma_R = K1*deltasigma_R;

   circle_r

   alpha = [R r]

end

alph = [bestR bestr];


function [alpha, bestcircle_r] = PSO_Stewart

%*******************************************************************
%PSO_Stewart.m - This script generates 2 parameters that will be optimized
%              in order to maximize the orientational workspace of the
%              Stewart represented as a the maximum radius of a circular
%              trajectory that the platform can follow. It performs
%              10 generations of the Particle Swarm Optimization to
optimize
%              these values and then returns the best values that are
%              found and largest circular trajectory radius found.
%*******************************************************************

%Most of this code is a combination of the PSO_Delta function and the
%Workspace_Points function. Therefore, much of it has already been
%commented. However, anything that is unique to this function will be
%explained in the comments in this code

R = 5 + (12-5).*rand(20,1);

r = 18 + (25-18).*rand(20,1);

gens = 0;

theta = 0;
phi = 0;
psi = 0;
max_h = 41.91;
min_h = 31.75;
mid_h = (max_h + min_h)/2;
circle_h = mid_h + 50;

maxcircle_r = zeros(20,1);
bestcircle_r = 0;
oldcircle_r = zeros(20,1);

Z = [0 0 1];

alpha = [R r]

valid1 = 0;
valid2 = 0;
valid3 = 0;
valid4 = 0;
valid5 = 0;
valid6 = 0;
valid7 = 0;
```

```
valid8 = 0;


while bestcircle_r < 26.7

    gens = gens + 1;

for i=1:1:20

    P1 = [R(i) 0 0 1]';
    P2 = [(-1/2)*R(i) (sqrt(3)/2)*R(i) 0 1]';
    P3 = [(-1/2)*R(i) -(sqrt(3)/2)*R(i) 0 1]';

    B1 = [r(i) 0 0]';
    B2 = [(-1/2)*r(i) (sqrt(3)/2)*r(i) 0]';
    B3 = [(-1/2)*r(i) -(sqrt(3)/2)*r(i) 0]';

    a11 = cos(theta)*cos(phi);
    a12 = cos(theta)*sin(phi);
    a13 = -sin(theta);
    a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
    a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
    a23 = cos(theta)*sin(psi);
    a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
    a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
    a33 = cos(theta)*cos(psi);

    q = [0 0 mid_h]';

    Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];

    u1 = Rm*P1;
    u2 = Rm*P2;
    u3 = Rm*P3;

    l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
    l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
    l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

    OrigBaseAngle1 = asin(u1(3)/l1mag);
    OrigPlatAngle1 = acos(u1(3)/l1mag);

    OrigBaseAngle2 = asin(u2(3)/l2mag);
    OrigPlatAngle2 = acos(u2(3)/l2mag);

    OrigBaseAngle3 = asin(u3(3)/l3mag);
    OrigPlatAngle3 = acos(u3(3)/l3mag);


    for circle_r = 30:-0.1:0

        valid1 = 0;
        valid2 = 0;
        valid3 = 0;
        valid4 = 0;
        valid5 = 0;
        valid6 = 0;
        valid7 = 0;
```

```
valid8 = 0;

for theta = -pi/3:0.025:pi/3
        for psi =  -pi/3:0.025:pi/3

            V1 = [circle_r 0 circle_h]'; %points on the circle
            V2 = [circle_r*1/sqrt(2) circle_r*1/sqrt(2) circle_h]';
            V3 = [0 circle_r circle_h]';
            V4 = [-circle_r*1/sqrt(2) circle_r*1/sqrt(2) ...
                circle_h]';
            V5 = [-circle_r 0 circle_h]';
            V6 = [-circle_r*1/sqrt(2) -circle_r*1/sqrt(2) ...
                circle_h]';
            V7 = [0 -circle_r circle_h]';
            V8 = [circle_r*1/sqrt(2) -circle_r*1/sqrt(2) ...
                circle_h]';

            V = {V1 V2 V3 V4 V5 V6 V7 V8};

            a11 = cos(theta)*cos(phi);
            a12 = cos(theta)*sin(phi);
            a13 = -sin(theta);
            a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
            a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
            a23 = cos(theta)*sin(psi);
            a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
            a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
            a33 = cos(theta)*cos(psi);

            q = [0 0 mid_h]';

            Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2); ...
                a31 a32 a33 q(3);0 0 0 1];

            u1 = Rm*P1;
            u2 = Rm*P2;
            u3 = Rm*P3;

            l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+ ...
                      (u1(3)-B1(3))^2);
            l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+ ...
                      (u2(3)-B2(3))^2);
            l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+ ...
                      (u3(3)-B3(3))^2);

            if theta == 0 && phi == 0
                OrigBaseAngle1 = asin(u1(3)/l1mag);
                OrigPlatAngle1 = acos(u1(3)/l1mag);

                OrigBaseAngle2 = asin(u2(3)/l2mag);
                OrigPlatAngle2 = acos(u2(3)/l2mag);

                OrigBaseAngle3 = asin(u3(3)/l3mag);
                OrigPlatAngle3 = acos(u3(3)/l3mag);
            end

            N1 = cross(Z,B1);
            N2 = cross(Z,B2);
            N3 = cross(Z,B3);
```

```
D1 = abs(N1(1)*u1(1) + N1(2)*u1(2) ...
     + N1(3)*u1(3))/sqrt(N1(1)^2 + N1(2)^2 + N1(3)^2);
D2 = abs(N2(1)*u2(1) + N2(2)*u2(2) ...
     + N2(3)*u2(3))/sqrt(N2(1)^2 + N2(2)^2 + N2(3)^2);
D3 = abs(N3(1)*u3(1) + N3(2)*u3(2) ...
     + N3(3)*u3(3))/sqrt(N3(1)^2 + N3(2)^2 + N3(3)^2);

    if (l1mag < 41.91 && l2mag < 41.91 && ...
     l3mag < 41.91 && l1mag > 31.75 && ...
     l2mag > 31.75 && l3mag > 31.75 ...
    && abs(asin(u1(3)/l1mag)-OrigBaseAngle1) < pi/4...
    && abs(acos(u1(3)/l1mag)-OrigPlatAngle1) < 0.44...
    && abs(asin(u2(3)/l2mag)-OrigBaseAngle2) < pi/4...
    && abs(acos(u2(3)/l2mag)-OrigPlatAngle2) < 0.44 ...
    && abs(asin(u3(3)/l1mag)-OrigBaseAngle3) < pi/4 ...
    && abs(acos(u3(3)/l3mag)-OrigPlatAngle3) < 0.44 ...
    && D1 < 0.25 && D2 < 0.25 && D3 < 0.25)

        crossvec1 = u1-u2;
        crossvec2 = u1-u3;
        crossvec1(4) = [];
        crossvec2(4) = [];
        C1 = cross(crossvec1,crossvec2);
        C2 = cross(crossvec2,crossvec1);

        if (C1(3) > 0)
             C = C1;
        elseif (C2(3) > 0)
             C = C2;
        end

        C_Norm1 = C/norm(C);
        C_Norm2 = 2*C_Norm1;

        for j = 1:1:8
            d_v = norm(cross(V{j}-C_Norm1, ...
         V{j}-C_Norm2))/norm(C_Norm2-C_Norm1);
            if d_v <= 1.5

                if j == 1
                    valid1 = 1;
                elseif j == 2
                    valid2 = 1;
                elseif j == 3
                    valid3 = 1;
                elseif j == 4
                    valid4 = 1;
                elseif j == 5
                    valid5 = 1;
                elseif j == 6
                    valid6 = 1;
                elseif j == 7
                    valid7 =1;
                elseif j == 8
                    valid8 = 1;
                end
            end
        end
    end
```

```
                end
        end

        valid = valid1 + valid2 + valid3 + valid4 + valid5 + valid6 ...
       + valid7 + valid8;

        if valid == 8
            valid1 = 0;
            valid2 = 0;
            valid3 = 0;
            valid4 = 0;
            valid5 = 0;
            valid6 = 0;
            valid7 = 0;
            valid8 = 0;
            break
        end

    end

    maxcircle_r(i) = circle_r;
end

for i=1:1:20
    if maxcircle_r(i) > bestcircle_r
        bestcircle_r = maxcircle_r(i);
        gbest = [R(i) r(i)];
    end
end

gbestorig = gbest;
gbest = repmat(gbest,20,1);

for i = 1:1:20
    if maxcircle_r(i) >= oldcircle_r(i)
            pbest(i,1) = R(i);
            pbest(i,2) = r(i);
    end
end

c1 = 0.5;
c2 = 1.25;
vmax = [];
v_R = repmat(0.0001,20,1);
v_r = repmat(0.0001,20,1);

for i = 1:1:20
    v_R(i) = v_R(i) + c1*rand*(pbest(i,1)-alpha(i,1))+ ...
            c2*rand*(gbest(i,1)-alpha(i,1));
    v_r(i) = v_r(i) + c1*rand*(pbest(i,2)-alpha(i,2))+ ...
            c2*rand*(gbest(i,2)-alpha(i,2));
end

gbest = gbestorig;
oldr = r;
oldR = R;

oldalpha = alpha;
oldcircle_r = maxcircle_r;
```

```
newr = r+v_r;

for i=1:1:20
    if newr(i) < 18 || newr(i) > 25
        r(i) = oldr(i);
    else
        r(i) = newr(i);
    end
end

newR = R+v_R;

for i=1:1:20
    if newR(i) < 5 || newR(i) > 12
        R(i) = oldR(i);
    else
        R(i) = newR(i);
    end
end

newalpha = [R r];
alpha = newalpha;

end

alpha = gbest(1,1:2);

function [optimumworkspace,alph] = project_stewart

%*********************************************************************
%Project_Stewart.m - This script calls CRS_Stewart and then uses the value
%                    returned by this function to populate 5 of the 20
%                    values in the initial swarm array. The other 15 values
%                    are randomly generated within the bound of the
%                    parameters. Then PSO is used to optimize the
%                    parameters. The function returns the best parameters
%                    found and the optimum workspace found
%*********************************************************************

%Most of this code is a combination of the PSO_Delta function,
%Project_Delta and the Workspace_Points function. Therefore, much of it
%has already been commented. However, anything that is unique to this
%function will be explained in the comments in this code

alph = crs_stewart();

R = alph(1);
r = alph(2);

R1 = repmat(R,5,1);
R2 = 5 + (12-5).*rand(15,1);

R = [R1; R2];

r1 = repmat(r,5,1);
r2 = 18 + (25-18).*rand(15,1);
```

```
r = [r1; r2];

gens = 0;

theta = 0;
phi = 0;
psi = 0;
max_h = 41.91;
min_h = 31.75;
mid_h = (max_h + min_h)/2;
circle_h = mid_h + 50;

maxcircle_r = zeros(20,1);
bestcircle_r = 0;
oldcircle_r = zeros(20,1);

Z = [0 0 1];

alpha = [R r]

valid1 = 0;
valid2 = 0;
valid3 = 0;
valid4 = 0;
valid5 = 0;
valid6 = 0;
valid7 = 0;
valid8 = 0;


while bestcircle_r < 26.7

    gens = gens + 1;

for i=1:1:20

    P1 = [R(i) 0 0 1]';
    P2 = [(-1/2)*R(i) (sqrt(3)/2)*R(i) 0 1]';
    P3 = [(-1/2)*R(i) -(sqrt(3)/2)*R(i) 0 1]';

    B1 = [r(i) 0 0]';
    B2 = [(-1/2)*r(i) (sqrt(3)/2)*r(i) 0]';
    B3 = [(-1/2)*r(i) -(sqrt(3)/2)*r(i) 0]';

    a11 = cos(theta)*cos(phi);
    a12 = cos(theta)*sin(phi);
    a13 = -sin(theta);
    a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
    a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
    a23 = cos(theta)*sin(psi);
    a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
    a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
    a33 = cos(theta)*cos(psi);

    q = [0 0 mid_h]';

    Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 q(3);0 0 0 1];m
```

```
u1 = Rm*P1;
u2 = Rm*P2;
u3 = Rm*P3;

l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+(u1(3)-B1(3))^2);
l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+(u2(3)-B2(3))^2);
l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+(u3(3)-B3(3))^2);

OrigBaseAngle1 = asin(u1(3)/l1mag);
OrigPlatAngle1 = acos(u1(3)/l1mag);

OrigBaseAngle2 = asin(u2(3)/l2mag);
OrigPlatAngle2 = acos(u2(3)/l2mag);

OrigBaseAngle3 = asin(u3(3)/l3mag);
OrigPlatAngle3 = acos(u3(3)/l3mag);


for circle_r = 30:-0.1:0

    valid1 = 0;
    valid2 = 0;
    valid3 = 0;
    valid4 = 0;
    valid5 = 0;
    valid6 = 0;
    valid7 = 0;
    valid8 = 0;

    for theta = -pi/3:0.025:pi/3
            for psi =  -pi/3:0.025:pi/3

                V1 = [circle_r 0 circle_h]';
                V2 = [circle_r*1/sqrt(2) circle_r*1/sqrt(2) ...
                    circle_h]';
                V3 = [0 circle_r circle_h]';
                V4 = [-circle_r*1/sqrt(2) circle_r*1/sqrt(2) ...
                    circle_h]';
                V5 = [-circle_r 0 circle_h]';
                V6 = [-circle_r*1/sqrt(2) -circle_r*1/sqrt(2) ...
                    circle_h]';
                V7 = [0 -circle_r circle_h]';
                V8 = [circle_r*1/sqrt(2) -circle_r*1/sqrt(2) ...
                    circle_h]';

                V = {V1 V2 V3 V4 V5 V6 V7 V8};

                a11 = cos(theta)*cos(phi);
                a12 = cos(theta)*sin(phi);
                a13 = -sin(theta);
                a21 = sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
                a22 = sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi);
                a23 = cos(theta)*sin(psi);
                a31 = cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
                a32 = cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi);
                a33 = cos(theta)*cos(psi);

                q = [0 0 mid_h]';
```

```
Rm = [a11 a12 a13 q(1);a21 a22 a23 q(2);a31 a32 a33 ...
    q(3);0 0 0 1];

u1 = Rm*P1;
u2 = Rm*P2;
u3 = Rm*P3;

l1mag = sqrt((u1(1)-B1(1))^2+(u1(2)-B1(2))^2+ ...
    (u1(3)-B1(3))^2);
l2mag = sqrt((u2(1)-B2(1))^2+(u2(2)-B2(2))^2+ ...
    (u2(3)-B2(3))^2);
l3mag = sqrt((u3(1)-B3(1))^2+(u3(2)-B3(2))^2+ ...
    (u3(3)-B3(3))^2);

if theta == 0 && phi == 0
    OrigBaseAngle1 = asin(u1(3)/l1mag);
    OrigPlatAngle1 = acos(u1(3)/l1mag);

    OrigBaseAngle2 = asin(u2(3)/l2mag);
    OrigPlatAngle2 = acos(u2(3)/l2mag);

    OrigBaseAngle3 = asin(u3(3)/l3mag);
    OrigPlatAngle3 = acos(u3(3)/l3mag);
end

N1 = cross(Z,B1);
N2 = cross(Z,B2);
N3 = cross(Z,B3);

D1 = abs(N1(1)*u1(1) + N1(2)*u1(2) + ...
    N1(3)*u1(3))/sqrt(N1(1)^2 + N1(2)^2 + N1(3)^2);
D2 = abs(N2(1)*u2(1) + N2(2)*u2(2) + ...
    N2(3)*u2(3))/sqrt(N2(1)^2 + N2(2)^2 + N2(3)^2);
D3 = abs(N3(1)*u3(1) + N3(2)*u3(2) + ...
    N3(3)*u3(3))/sqrt(N3(1)^2 + N3(2)^2 + N3(3)^2);

    if (l1mag < 41.91 && l2mag < 41.91 && ...
        l3mag < 41.91 && l1mag > 31.75 && ...
        l2mag > 31.75 && l3mag > 31.75 ...
        && abs(asin(u1(3)/l1mag)-OrigBaseAngle1) ...
         < pi/4 ...
        && abs(acos(u1(3)/l1mag)-OrigPlatAngle1) ...
         < 0.44 ...
        && abs(asin(u2(3)/l2mag)-OrigBaseAngle2) ...
         < pi/4 ...
        && abs(acos(u2(3)/l2mag)-OrigPlatAngle2) ...
         < 0.44 ...
        && abs(asin(u3(3)/l1mag)-OrigBaseAngle3) ...
         < pi/4 ...
        && abs(acos(u3(3)/l3mag)-OrigPlatAngle3) ...
         < 0.44 ...
        && D1 < 0.25 && D2 < 0.25 && D3 < 0.25)

        crossvec1 = u1-u2;
        crossvec2 = u1-u3;
        crossvec1(4) = [];
        crossvec2(4) = [];
        C1 = cross(crossvec1,crossvec2);
        C2 = cross(crossvec2,crossvec1);
```

```
                            if (C1(3) > 0)
                                 C = C1;
                            elseif (C2(3) > 0)
                                 C = C2;
                            end

                            C_Norm1 = C/norm(C);
                            C_Norm2 = 2*C_Norm1;

                            for j = 1:1:8
                                d_v = norm(cross(V{j}-C_Norm1,V{j}-
C_Norm2)) ...

                                /norm(C_Norm2-C_Norm1);
                                if d_v <= 1.5

                                    if j == 1
                                        valid1 = 1;
                                    elseif j == 2
                                        valid2 = 1;
                                    elseif j == 3
                                        valid3 = 1;
                                    elseif j == 4
                                        valid4 = 1;
                                    elseif j == 5
                                        valid5 = 1;
                                    elseif j == 6
                                        valid6 = 1;
                                    elseif j == 7
                                        valid7 =1;
                                    elseif j == 8
                                        valid8 = 1;
                                    end
                                end
                            end
                        end
                end
        end

        valid = valid1 + valid2 + valid3 + valid4 + valid5 + valid6 ...
            + valid7 + valid8;

        if valid == 8
            valid1 = 0;
            valid2 = 0;
            valid3 = 0;
            valid4 = 0;
            valid5 = 0;
            valid6 = 0;
            valid7 = 0;
            valid8 = 0;
            break
        end

    end

    maxcircle_r(i) = circle_r;
end
```

```
for i=1:1:20
    if maxcircle_r(i) > bestcircle_r
        bestcircle_r = maxcircle_r(i);
        gbest = [R(i) r(i)];
    end
end

gbestorig = gbest;
gbest = repmat(gbest,20,1);

for i = 1:1:20
    if maxcircle_r(i) >= oldcircle_r(i)
            pbest(i,1) = R(i);
            pbest(i,2) = r(i);
    end
end

c1 = 0.5;
c2 = 1.25;
vmax = [];
v_R = repmat(0.0001,20,1);
v_r = repmat(0.0001,20,1);

for i = 1:1:20
    v_R(i) = v_R(i) + c1*rand*(pbest(i,1)-alpha(i,1))+...
        c2*rand*(gbest(i,1)-alpha(i,1));
    v_r(i) = v_r(i) + c1*rand*(pbest(i,2)-alpha(i,2))+...
        c2*rand*(gbest(i,2)-alpha(i,2));
end

gbest = gbestorig;

oldr = r;
oldR = R;

oldalpha = alpha;
oldcircle_r = maxcircle_r;

newr = r+v_r;

for i=1:1:20
    if newr(i) < 18 || newr(i) > 25
        r(i) = oldr(i);
    else
        r(i) = newr(i);
    end
end

newR = R+v_R;

for i=1:1:20
    if newR(i) < 5 || newR(i) > 12
        R(i) = oldR(i);
    else
        R(i) = newR(i);
    end
end

newalpha = [R r];
```

```
alpha = newalpha;

end

alpha = gbest(1,1:2);
```

Appendix C – Microcontroller Code

```
//#####################################################################################
#
//RobotCode.c
//This code accepts length of the three actuator legs in L1Array, L2Array
//and L3Array. Each individual leg will go to its length for that index in the
//array and stop once it reaches it. It will not move to the next length until
//all legs have reached their respective length for that index value. A number
//of files need to be included in the project code that are not shown in this
//code in order for it to compile properly, but they are mentioned in the
//comments.
//#####################################################################################
#

#include "F28x_Project.h"     // Device header file

//Definitions for selecting ADC resolution (0 means that it is selected)
#define RESOLUTION_12BIT 0 //12-bit resolution
#define RESOLUTION_16BIT 1 //16-bit resolution

//Definitions for selecting ADC signal mode (0 means that it is selected)
#define SIGNAL_SINGLE 0 //single-ended channel conversions (12-bit mode only)
#define SIGNAL_DIFFERENTIAL 1 //differential pair channel conversions

//Define the structure that is used to give information about the PWM
//wave to its function call (mainly the length of its duty cycle)
typedef struct
{
   volatile struct EPWM_REGS *EPwmRegHandle;
   Uint32 EPwm_CMPA_Direction;
   Uint32 EPwm_CMPB_Direction;
   Uint32 EPwmTimerIntCount;
```

```
    Uint32 EPwmMaxCMPA;
    Uint32 EPwmMinCMPA;
    Uint32 EPwmMaxCMPB;
    Uint32 EPwmMinCMPB;
}EPWM_INFO;

//Function declarations
void InitEPwm1(void);
void InitEPwm2(void);
void InitEPwm3(void);

void ConfigureADC(void);
void SetupADCSoftware(void);

void update_compare1(EPWM_INFO*);
void update_compare2(EPWM_INFO*);
void update_compare3(EPWM_INFO*);

void forward1(EPWM_INFO *epwm_info);
void forward2(EPWM_INFO *epwm_info);
void forward3(EPWM_INFO *epwm_info);

void backward1(EPWM_INFO *epwm_info);
void backward2(EPWM_INFO *epwm_info);
void backward3(EPWM_INFO *epwm_info);

//Interrupt function definitions
__interrupt void epwm1_isr(void);
__interrupt void epwm2_isr(void);
__interrupt void epwm3_isr(void);

__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);

// Global variables declarations
EPWM_INFO epwm1_info;
EPWM_INFO epwm2_info;
EPWM_INFO epwm3_info;

//Configure the period for each timer and set the initial
//duty cycle values
//Most of these values are just intialization values that
//don't effect much of the function of the wave later on
//EPWMX_MIN_CMPA is the important definition because it
//is the high part of the wave, and this is the part of
//wave that controls the direction and speed of the leg
//8500 is backwards at a relatively high speed
//10500 is forwards at a relatively high speed
//Somewhere in the middle of those numbers makes the
//actuator halt, but this can be variable so its best
//to use 0 as the value instead
#define EPWM1_TIMER_TBPRD   18750  // Period register
#define EPWM1_MAX_CMPA      10500
#define EPWM1_MIN_CMPA       9000   //Initial ON part of the wave
#define EPWM1_MAX_CMPB       1950
#define EPWM1_MIN_CMPB       50

#define EPWM2_TIMER_TBPRD  18750 // Period register
#define EPWM2_MAX_CMPA     10500
#define EPWM2_MIN_CMPA     9000        //Initial ON part of the wave
#define EPWM2_MAX_CMPB     1950
#define EPWM2_MIN_CMPB     50
```

```
#define EPWM3_TIMER_TBPRD  18750  // Period register
#define EPWM3_MAX_CMPA     10500
#define EPWM3_MIN_CMPA     9000        //Initial ON part of the wave
#define EPWM3_MAX_CMPB     1950
#define EPWM3_MIN_CMPB     50

//Declare index variables to be used in arrays
int Index1 = 0;
int Index2 = 0;
int Index3 = 0;

//Declare that flag that is incremented if a leg length
//reaches its length for the current index
int flag = 3;

//Declare time values that will be used to determine
//which leg is the last leg to reach its length for the
//current index so that the index can be incremented
int time1 = 0;
int time2 = 0;
int time3 = 0;

//Declare the size of the leg lengths arrays so the program knows
//when to return the actuators to there 0 position (all they way down)
int ArraySize = 18;

//The arras that contain the lenght values required for each actuator to
//follow the circular trajectory
double L1Array[18] = {0.05, 0.903, 0.444, 0.200, 0.300, 0.517, 0.783, 1.052, 1.596, 2.164,
2.737, 3.297, 2.974, 2.648, 2.330, 2.031, 1.466, 0.903}; //L2 (left leg)
double L2Array[18] = {0.10, 2.829, 2.750, 2.800, 2.138, 1.530, 0.960, 0.453, 0.392, 0.371,
0.392, 0.453, 0.960, 1.530, 2.137, 2.761, 2.812, 2.829}; //L1 (back leg)
```

```
double L3Array[18] = {0.40, 0.903, 1.466, 2.031, 2.330, 2.648, 2.974, 3.297, 2.737, 2.164,
1.596, 1.052, 0.783, 0.517, 0.262, 0.500, 0.444, 0.903}; //L3 (right leg)


//Set the initial value of the position to be very low so that the actuator
//always goes forward initially
int L1Digital_Old = 4000;
int L2Digital_Old = 4000;
int L3Digital_Old = 4000;


//Declare the variables that will contain the digital value
//of the potentiometer voltage (i.e. the length of the
//actuator)
int L1Digital = 0;
int L2Digital = 0;
int L3Digital = 0;



void main(void)
{


//Initialize System Control:
//PLL, WatchDog, enable Peripheral Clocks
//This function is called from the F28M36x_SysCtrl.c


        InitSysCtrl();


//Initialize GPIO:
// This function is called from the F28M36x_Gpio.c
// The GPIO to it's default state.
   InitGpio();


//Allocate PWM1, PWM2 and PWM3 to CPU1
```

```
//    CpuSysRegs.CPUSEL0.bit.EPWM1 = 0;
//    CpuSysRegs.CPUSEL0.bit.EPWM2 = 0;
//    CpuSysRegs.CPUSEL0.bit.EPWM3 = 0;


//Enable PWM1, PWM2 and PWM3
   CpuSysRegs.PCLKCR2.bit.EPWM1=1;
   CpuSysRegs.PCLKCR2.bit.EPWM2=1;
   CpuSysRegs.PCLKCR2.bit.EPWM3=1;

//Initialize GPIO pins for ePWM1, ePWM2, ePWM3
//These functions are called from the F28M36x_EPwm.c file
        InitEPwm1Gpio();
        InitEPwm2Gpio();
        InitEPwm3Gpio();

// Initialize the PIE control registers to their default state.
// This means that all PIE interrupts are disabled and flags
// are cleared.
// This function is called from the F28M36x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   DINT;
   IER = 0x0000;
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table
// The shell ISR routines are found in F28M36x_DefaultIsr.c.
// This function is called from in F28M36x_PieVect.c.
```

```
    InitPieVectTable();

// PWM Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
            EALLOW; // This is needed to write to EALLOW protected registers
            PieVectTable.EPWM1_INT = &epwm1_isr;
            PieVectTable.EPWM2_INT = &epwm2_isr;
            PieVectTable.EPWM3_INT = &epwm3_isr;
            EDIS;   // This is needed to disable write to EALLOW protected registers

// Initialize all the Device Peripherals:
// This function is found in F28M36x_InitPeripherals.c
// InitPeripherals();  // Not required for this example



            EALLOW;
            CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0;

            EDIS;

            InitEPwm1();
            InitEPwm2();
            InitEPwm3();

            EALLOW;
            CpuSysRegs.PCLKCR0.bit.TBCLKSYNC =1;

            EDIS;

// Enable CPU INT3 which is connected to EPWM1-3 INT:
            IER |= M_INT3;
```

```
// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
            PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
            PieCtrlRegs.PIEIER3.bit.INTx2 = 1;
            PieCtrlRegs.PIEIER3.bit.INTx3 = 1;



// Enable global Interrupts and higher priority real-time debug events:
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM

//Initialize CPU timers
//This function is called from F2806x_CpuTimers.h
    InitCpuTimers();

// CPU Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
    EALLOW;  // This is needed to write to EALLOW protected registers
    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
    PieVectTable.TIMER1_INT = &cpu_timer1_isr;
    PieVectTable.TIMER2_INT = &cpu_timer2_isr;
    EDIS;    // This is needed to disable write to EALLOW protected registers

// Configure CPU-Timer 0, 1, and 2 to interrupt every second:
// 200MHz CPU Freq, 0.01 second Period (in uSeconds)


    ConfigCpuTimer(&CpuTimer0, 200, 10000);
    ConfigCpuTimer(&CpuTimer1, 200, 10000);
    ConfigCpuTimer(&CpuTimer2, 200, 10000);

// To ensure precise timing, use write-only instructions to write to the entire register.
    CpuTimer0Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0
    CpuTimer1Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0
```

```
   CpuTimer2Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 0

//Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
// which is connected to CPU-Timer 1, and CPU int 14, which is connected
// to CPU-Timer 2:
   IER |= M_INT1;
   IER |= M_INT13;
   IER |= M_INT14;

// Enable TINT0 in the PIE: Group 1 interrupt 7
   PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

//Configure the ADCs and power them up
   ConfigureADC();

//Setup the ADCs for software conversions
   SetupADCSoftware();

//IDLE loop.
   for(;;)
   {
         asm ("         NOP");

   }
}

//Write ADC configurations and power up the ADC for both ADC A and ADC B
void ConfigureADC(void)
{
         EALLOW;
         //write configurations
         AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
```

```
        AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
        AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4

        AdcaRegs.ADCCTL2.bit.RESOLUTION = RESOLUTION_12BIT;
        AdcbRegs.ADCCTL2.bit.RESOLUTION = RESOLUTION_12BIT;
        AdcdRegs.ADCCTL2.bit.RESOLUTION = RESOLUTION_12BIT;

        AdcaRegs.ADCCTL2.bit.SIGNALMODE = SIGNAL_SINGLE;
        AdcbRegs.ADCCTL2.bit.SIGNALMODE = SIGNAL_SINGLE;
        AdcdRegs.ADCCTL2.bit.SIGNALMODE = SIGNAL_SINGLE;

        //Set pulse positions to late
        AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
        AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;

        //Power up the ADCs
        AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
        AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
        AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;

        //Delay for 1ms to allow ADC time to power up
        DELAY_US(1000);

        EDIS;
}

void SetupADCSoftware(void)
{

//Select the channels to convert and end of conversion flag
   //ADCA
```

```
    EALLOW;
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 0;  //SOC0 will convert pin A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK
                                                         //cycles
    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //EOC0 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    //ADCB
    AdcbRegs.ADCSOC0CTL.bit.CHSEL = 0;  //SOC1 will convert pin B0
    AdcbRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK
                                                         //cycles
    AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //EOC1 will set INT1 flag
    AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    //ADCC
    AdcdRegs.ADCSOC0CTL.bit.CHSEL = 0;  //SOC2 will convert pin C2
    AdcdRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK
                                                         //cycles
    AdcdRegs.ADCINTSEL1N2.bit.INT1SEL = 0; //EOC2 will set INT1 flag
    AdcdRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
    AdcdRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared

    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 1;     //set SOC0 start trigger on CPUTIMER1
    AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 2;     //set SOC1 start trigger on CPUTIMER2
    AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 3;     //set SOC2 start trigger on CPUTIMER3

    EDIS;
}

__interrupt void epwm1_isr(void)
{
    // Update the CMPA
    update_compare1(&epwm1_info);
    // Clear INT flag for this timer
    EPwm1Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

__interrupt void epwm2_isr(void)
{

    // Update the CMPA
    update_compare2(&epwm2_info);

    // Clear INT flag for this timer
    EPwm2Regs.ETCLR.bit.INT = 1;

    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

__interrupt void epwm3_isr(void)
{

    // Update the CMPA
    update_compare3(&epwm3_info);

    // Clear INT flag for this timer
    EPwm3Regs.ETCLR.bit.INT = 1;
    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}
```

```
void InitEPwm1()
{
        // Setup TBCLK
        EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
        EPwm1Regs.TBPRD = EPWM1_TIMER_TBPRD;       // Set timer period
        EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Disable phase loading
        EPwm1Regs.TBPHS.half.TBPHS = 0x0000;       // Phase is 0
        EPwm1Regs.TBCTR = 0x0000;                  // Clear counter
        EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;   //Clock ratio
        EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV4;

        // Setup shadow register load on ZERO
        EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
        EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
        EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
        EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

        // Set Compare values
        EPwm1Regs.CMPA.half.CMPA = EPWM1_MIN_CMPA;   // Set compare A value
        EPwm1Regs.CMPB.half.CMPB = EPWM1_MIN_CMPB;   // Set Compare B value

        // Set actions
        EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;           // Set PWM1A on Zero
        EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;     // Clear PWM1A on event A,

        EPwm1Regs.AQCTLB.bit.ZRO = AQ_SET;           // Set PWM1B on Zero
        EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;      // Clear PWM1B on event B

        // Interrupt where we will change the Compare Values
        EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
        EPwm1Regs.ETSEL.bit.INTEN = 1;                      // Enable INT
```

```
	EPwm1Regs.ETPS.bit.INTPRD = ET_3RD;		// Generate INT on 3rd event

	// Keeps track of the direction that
	// the CMPA/CMPB values are
	// moving, the min and max allowed values and
	// a pointer to the correct ePWM registers
	epwm1_info.EPwm_CMPA_Direction = 1; // Start by increasing CMPA & CMPB
	epwm1_info.EPwm_CMPB_Direction = 1;
	epwm1_info.EPwmTimerIntCount = 0;		// Zero the interrupt counter
	epwm1_info.EPwmRegHandle = &EPwm1Regs;	// Set the pointer to ePWMregs
	epwm1_info.EPwmMaxCMPA = EPWM1_MAX_CMPA;
	epwm1_info.EPwmMinCMPA = EPWM1_MIN_CMPA;
	epwm1_info.EPwmMaxCMPB = EPWM1_MAX_CMPB;
	epwm1_info.EPwmMinCMPB = EPWM1_MIN_CMPB;

}
//The other two ePWM initializations are the same as for ePWM1, so no comments will be
//included
void InitEPwm2()
{
	EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
	EPwm2Regs.TBPRD = EPWM1_TIMER_TBPRD;
	EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;
	EPwm2Regs.TBPHS.half.TBPHS = 0x0000;
	EPwm2Regs.TBCTR = 0x0000;
	EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;
	EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV4;

	EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
	EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
	EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
	EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
```

```
            EPwm2Regs.CMPA.half.CMPA = EPWM1_MIN_CMPA;
            EPwm2Regs.CMPB.half.CMPB = EPWM1_MIN_CMPB;

            EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;
            EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;

            EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;
            EPwm2Regs.AQCTLB.bit.CBU = AQ_CLEAR;

            EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;
            EPwm2Regs.ETSEL.bit.INTEN = 1;
            EPwm2Regs.ETPS.bit.INTPRD = ET_3RD;

            epwm2_info.EPwm_CMPA_Direction = 1;
            epwm2_info.EPwm_CMPB_Direction = 1;
            epwm2_info.EPwmTimerIntCount = 0;
            epwm2_info.EPwmRegHandle = &EPwm2Regs;
            epwm2_info.EPwmMaxCMPA = EPWM1_MAX_CMPA;
            epwm2_info.EPwmMinCMPA = EPWM1_MIN_CMPA;
            epwm2_info.EPwmMaxCMPB = EPWM1_MAX_CMPB;
            epwm2_info.EPwmMinCMPB = EPWM1_MIN_CMPB;

}
void InitEPwm3(void)
{
            EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP;
            EPwm3Regs.TBPRD = EPWM1_TIMER_TBPRD;
            EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;
            EPwm3Regs.TBPHS.half.TBPHS = 0x0000;
            EPwm3Regs.TBCTR = 0x0000;
            EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV4;
            EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV4;

            EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
            EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
            EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
            EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

            EPwm3Regs.CMPA.half.CMPA = EPWM1_MIN_CMPA;
            EPwm3Regs.CMPB.half.CMPB = EPWM1_MIN_CMPB;

            EPwm3Regs.AQCTLA.bit.ZRO = AQ_SET;
            EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;

            EPwm3Regs.AQCTLB.bit.ZRO = AQ_SET;
            EPwm3Regs.AQCTLB.bit.CBU = AQ_CLEAR
            EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;
            EPwm3Regs.ETSEL.bit.INTEN = 1;
            EPwm3Regs.ETPS.bit.INTPRD = ET_3RD;

            epwm3_info.EPwm_CMPA_Direction = 1;
            epwm3_info.EPwm_CMPB_Direction = 1;
            epwm3_info.EPwmTimerIntCount = 0;
            epwm3_info.EPwmRegHandle = &EPwm3Regs;
            epwm3_info.EPwmMaxCMPA = EPWM1_MAX_CMPA;
            epwm3_info.EPwmMinCMPA = EPWM1_MIN_CMPA;
            epwm3_info.EPwmMaxCMPB = EPWM1_MAX_CMPB;
            epwm3_info.EPwmMinCMPB = EPWM1_MIN_CMPB;
}

//CpuTimerX.InterruptCount will be used as a clock to determine
//which the last leg to reach its length for the current index is
__interrupt void cpu_timer0_isr(void)
```

```
{
  CpuTimer0.InterruptCount++;
  // Acknowledge this interrupt to receive more interrupts from group 1
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

__interrupt void cpu_timer1_isr(void)
{
  CpuTimer1.InterruptCount++;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

__interrupt void cpu_timer2_isr(void)
{
  CpuTimer2.InterruptCount++;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

//These functions accept the info for their respective PWM and controls
//that leg accordingly
void update_compare1(EPWM_INFO *epwm_info)
{

//If all of the actuators have reached their destination and
//its the last actuator to get there or they all reached their destination
//and its the first run through, then calculate the next length to go to for
//each leg and rest all of the timer values and flag value
	if ((flag == 3 && (time1 > time2 && time1 > time3 && time2 != 0 && time3 != 0))
|| (flag == 3 && (time1 < 10 && time2 < 10 && time3 < 10)))
	{
		flag = 0;
		time1 = 0;
```

```
			time2 = 0;
			time3 = 0;

//The LXDigital values convert length (in inches) from the index array to a digital value
//so that it can be compared against the value received from the
//potentiometer
			L1Digital = -515*L1Array[Index1]+4095;
			L2Digital = -501*L2Array[Index2]+4105;
			L3Digital = -478*L3Array[Index3]+4061;
		}

//If somehow the flag count exceeds what it should, then set it to 3
		if (flag > 3)
		{
			flag = 3;
		}

//If the last value that the leg went to was larger than the next value it
//needs to go to, then go forward. This may seem backwards but its because
//the 0 position for the actuator has a digital value of 4095 (approximately,
//but it varies quite a bit in reality) and as the actuator length increases
//the value decreases to around 2000 at the end of its stroke length
		if(L1Digital_Old > L1Digital)
		{
			forward1(&epwm1_info);
		}
//If the old value was larger than the new value then the leg needs to go
//backwards to reach its new value
		if(L1Digital_Old < L1Digital)
		{
			backward1(&epwm1_info);
		}
```

```
//If the values are the same then stop moving the actuator
//This is how the actuator stops and waits for the last
//leg to reach its position
        if(L1Digital_Old == L1Digital)
        {
                epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
        }
//If the index of the current leg exceeds the array size
//or both of the indices of the other legs exceed the array size
//then bring all actuators back down to the 0 position

        if(Index1 > ArraySize || (Index2 > ArraySize && Index3 > ArraySize))
        {
                epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;
        }

        return;
}

//The other two update_compare functions are the same as the first one but their
//each control their respective PWM wave and, therefore, actuator leg

void update_compare2(EPWM_INFO *epwm_info)
{
        if ((flag == 3 && (time2 > time1 && time2 > time3 && time1 != 0 && time3 != 0))
|| (flag == 3 && (time1 < 10 && time2 < 10 && time3 < 10)))
                flag = 0;
                time1 = 0;
                time2 = 0;
                time3 = 0;
                L1Digital = -515*L1Array[Index1]+4095;
                L2Digital = -501*L2Array[Index2]+4105;
                L3Digital = -478*L3Array[Index3]+4061;
        }

        if (flag > 3)
        {
                flag = 3;
        }

        if(L2Digital_Old > L2Digital)
        {
                forward2(&epwm2_info);
        }

        if(L2Digital_Old < L2Digital)
        {
                backward2(&epwm2_info);
        }

        if(L2Digital_Old == L2Digital)
        {
                epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
        }

        if(Index2 > ArraySize || (Index1 > ArraySize && Index3 > ArraySize))
        {
                epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;
        }
        return;
}
```

```
void update_compare3(EPWM_INFO *epwm_info)
{
        if ((flag == 3 && (time3 > time2 && time3 > time1 && time1 != 0 && time2 != 0))
|| (flag == 3 && (time1 < 10 && time2 < 10 && time3 < 10)))
        {
                flag = 0;
                time1 = 0;
                time2 = 0;
                time3 = 0;

                L1Digital = -530*L1Array[Index1]+4025;
                L2Digital = -501*L2Array[Index2]+4105;
                L3Digital = -478*L3Array[Index3]+4061;
        }



        if (flag > 3)
        {
                flag = 3;
        }

        if(L3Digital_Old > L3Digital)
        {
                forward3(&epwm3_info);
        }

        if(L3Digital_Old < L3Digital)
        {
                backward3(&epwm3_info);
        }

        if(L3Digital_Old == L3Digital)
```

```
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
	}

	if(Index3 > ArraySize || (Index1 > ArraySize && Index2 > ArraySize))
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;
	}
	return;

}

//Each leg has its own forward and backward function so
//the it receives the correct PWM info as its input.
//So now that we know that the leg needs to go forward
//to reach its next position, we set the PWM wave for that
//leg to go drive the arm forward. The if statement
//checks the ADC register for that leg to see if it has
//reached its new position, and if it has then
//the actuator leg stops and the positon it just
//reached is now the previously position value
//and is stored as such. The time that this leg reached
//its position is stored in time1 and the index for this leg
//is increased and the flag value is also increased
void forward1(EPWM_INFO *epwm_info)
{
	epwm_info->EPwmRegHandle->CMPA.half.CMPA = 10500;
	if (AdcaResultRegs.ADCRESULT0 < L1Digital)
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
		L1Digital_Old = L1Digital;
		time1 = CpuTimer0.InterruptCount;
		flag++;
		Index1++;
	}
	return;
}

//The backwards function works exactly the same as the forward
//function excpet that it changes the PWM so that the actuator
//is driven backward instead of forward
void backward1(EPWM_INFO *epwm_info)
{
	epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;

	if (AdcaResultRegs.ADCRESULT0 > L1Digital)
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
		L1Digital_Old = L1Digital;
		time1 = CpuTimer0.InterruptCount;
		flag++;
		Index1++;
	}

	return;
}
//All the other functions are the same as the above two forward
//and backward functions, but they control the other two PWM waves
void forward2(EPWM_INFO *epwm_info)
{
	epwm_info->EPwmRegHandle->CMPA.half.CMPA = 10500;

	if (AdcbResultRegs.ADCRESULT0 < L2Digital)
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
		L2Digital_Old = L2Digital;
		time2 = CpuTimer0.InterruptCount;
		flag++;
		Index2++;
	}

	return;
}

void backward2(EPWM_INFO *epwm_info)
{
	epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;

	if (AdcbResultRegs.ADCRESULT0 > L2Digital)
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
		L2Digital_Old = L2Digital;
		time2 = CpuTimer0.InterruptCount;
		flag++;
		Index2++;
	}
	return;
}

void forward3(EPWM_INFO *epwm_info)
{
	epwm_info->EPwmRegHandle->CMPA.half.CMPA = 10500;

	if (AdcdResultRegs.ADCRESULT0 < L3Digital)
	{
		epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
```

```
        L3Digital_Old = L3Digital;
        time3 = CpuTimer0.InterruptCount;
        flag++;
        Index3++;
    }
    return;
}

void backward3(EPWM_INFO *epwm_info)
{
    epwm_info->EPwmRegHandle->CMPA.half.CMPA = 8500;

    if (AdcdResultRegs.ADCRESULT0 > L3Digital)
    {
        epwm_info->EPwmRegHandle->CMPA.half.CMPA = 0;
        L3Digital_Old = L3Digital;
        time3 = CpuTimer0.InterruptCount;
        flag++;
        Index3++;
    }
    return;
}
```

Curriculum Vitae

Name:	Derek Brecht
Place of birth:	Chatham, Ontario
Year of birth:	1990
Post-secondary education and degrees:	2009-2013 B. Sc. (Eng.) Electrical Engineering The University of Western Ontario London, Ontario, Canada
Honours and awards:	NSERC Undergraduate Student Research Award 2013
	Victor Hangan Global Opportunities Award 2013
	University of Western Ontario Dean's Honour List 2013
Related work experience:	2014-2015 Teaching Assistant Department of Electrical & Computer Engineering' The University of Western Ontario London, Ontario, Canada