DESIGN AND IMPLEMENTATION OF STEWART PLATFORM ROBOT FOR ROBOTICS COURSE LABORATORY

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Trent Robert Peterson

March 2020

COMMITTEE MEMBERSHIP

TITLE: Design and Implementation of Stewart Platform Robot for Robotics Course Laboratory

AUTHOR: Trent Robert Peterson

DATE SUBMITTED: March 2020

COMMITTEE CHAIR: Saeed B. Niku, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: William R. Murray, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: John R. Ridgely, Ph.D.
Professor of Mechanical Engineering

ABSTRACT

Design and Implementation of Stewart Platform Robot for Robotics Course Laboratory

Trent Robert Peterson

A Stewart Platform robot was designed, constructed, and programmed for use in Cal Poly's ME 423 Robotics: Fundamentals and Applications laboratory section. A Stewart Platform is a parallel manipulator robot with six prismatic joints that has six degrees of freedom, able to be defined in both position and orientation. Its purpose is to supplement parallel robot material covered in lecture. Learning objectives include applying and verifying the Stewart Platform inverse kinematics and investigating the Stewart Platform's operation, range of motion, and limitations. The Stewart Platform geometry and inverse kinematics were modeled and animated using MATLAB. The platform was then built using linear actuators, magnetic spherical bearings, and acrylic plates. Control of the Stewart Platform is achieved using an Arduino Due and a custom HexaMoto shield. Users interact with the system using a GUI created with MATLAB's App Designer.

Keywords: Stewart Platform, Parallel Robot, Linear Actuators, Robot, Inverse Kinematics, Arduino

ACKNOWLEDGMENTS

I would like to thank Dr. Saeed Niku for providing me the opportunity to create a robot and give back to the university, and for the guidance and patience along the way.

I would like to thank the members of my thesis committee, Dr. William Murray and Dr. John Ridgely, for imparting onto me the knowledge to make this robot and thesis possible.

I would like to thank Charlie Refvem for reviewing my board and giving me advice as I assembled my electronics hardware, saving me lots of time and potentially wasted effort.

I would like to thank the Cal Poly Mechanical Engineering Department for funding this robot, and to Christine Haas for working with me through the process.

I would like to thank Progressive Automations for the (pending) donation of the linear actuators.

I would like to thank Helical Products Company for supplying flexible couplings for use with the robot.

Finally, I would like to thank my family and friends for all their support during my time at Cal Poly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION AND BACKGROUND

1.1 Robots: A General Overview

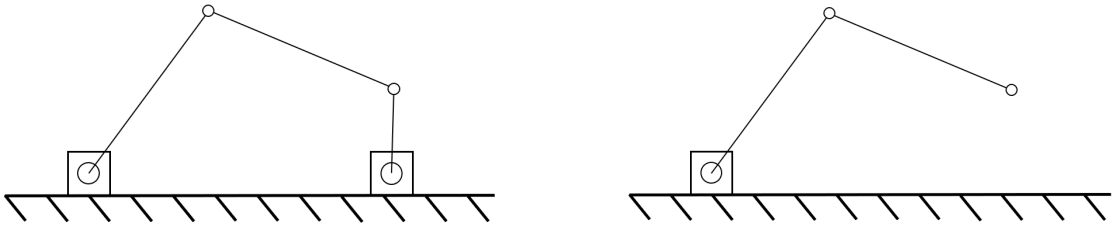
A robot is a machine that can be easily commanded, typically by a computer, to perform a set of tasks automatically. Robots are extraordinarily diverse in size, performance, cost, and operation. They typically have multiple links and actuators that create kinematic chains, and connect to an end effector. An end effector is a tool that is connected to the end of a robotic mechanism. The end effector may move in space in up to six ways called degrees of freedom (DOF). Three degrees of freedom are translation which occur along mutually-orthogonal axes, typically referred to as X, Y, and Z. Another three DOF may be rotation, which occur about these three axis. These six DOF are necessary to completely define the position and orientation of an object in space.

A joint is the connection between links that allows motion. This motion can be linear, rotary, or spherical. Examples of linear joints include linear motors, fluid-powered cylinders, and electric linear actuators. An electric linear actuator is articulated by a screw mechanism, powered by a rotary motor through gearing. A linear motor has the same operating principle as a rotary motor, but its rotor and stator are 'unfolded' allowing for linear but finite motion. It is important to note that a joint does not necessitate complete control. Spherical joints are difficult to control in practice. In addition, pneumatic joints are excellent for actuating to the extended and retracted positions of the cylinder, but controlling their actuation to intermediate positions

accurately is more difficult. Links are rigid members that connect the joints of a robot.

Forward kinematics and inverse kinematics are an integral segment of robotic study and analysis. Forward kinematics allows the position and orientation of the end effector to be known, given all joint values. Inverse kinematics determines the joint values given a known or desired end effector position and orientation.

The combination of joints and links comprise a kinematic loop, which can be either closed or open. An open kinematic loop occurs when links are connected in serial form via joints. The kinematic loop is closed when each link, which includes ground, is connected to a minimum of two other links. A four-bar linkage is a common mechanism that has a closed kinematic loop. The arm of an excavator is an example of an open loop mechanism (Figure 1.1).



(a) Closed Kinematic Loop (b) Open Kinematic Loop

Figure 1.1: Closed and Open Kinematic Loop Mechanisms

1.2 Parallel Robots

J. P. Merlet, author of *Parallel Robots*, defines a parallel general manipulator as a "closed-loop kinematic chain mechanism whose end effector is linked to the base by several independent kinematic chains" [15]. The manipulator becomes an n-degree-of-freedom robot when controlled actuation occurs through quantity n actuators. This

geometry gives a parallel robot its distinct characteristics, analysis, and behavior. Parallel robots can have up to six DOF and have various constructions.

1.2.1 Advantages and Disadvantages

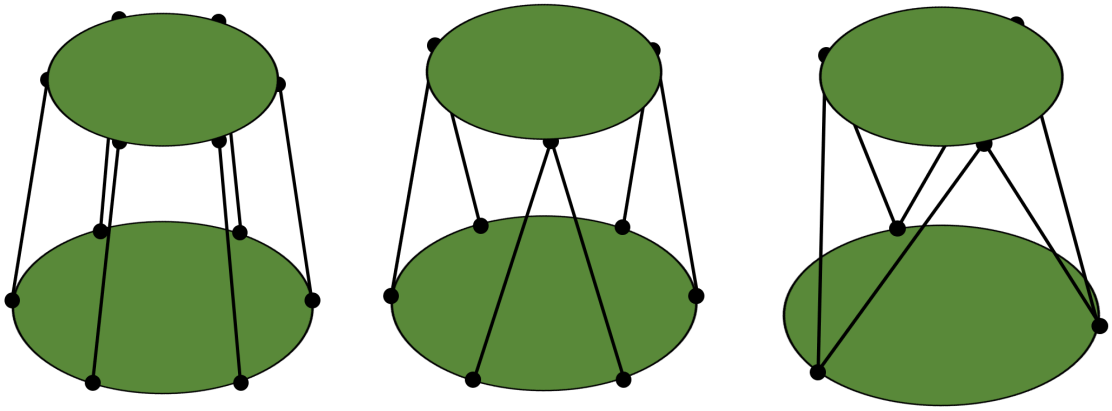
It is important to compare the differences between serial and parallel robots and illustrate the areas in which each type of robot excels. An important characteristic of a robot is its payload-to-mass ratio, which is typically higher for parallel robots. This is due to the closed-loop construction of a parallel robot. By distributing the loading of the robot between its actuators, this construction supports greater forces that comes with increased mass or acceleration. In comparison, the open-loop construction of a serial robot successively increases the load seen by each joint. The outermost joint, the wrist, needs to support the mass and inertial forces from the end effector only. The next joint, the elbow, must support the end effector, the wrist joint, and the forearm link. This continues to the base of the robot. Loading of an open-loop mechanism can cause deformation that is not detected by sensors which reduces positional accuracy. To compensate, the serial robot must be made very stiff, and this over-designed geometry increases robot weight. High stiffness and payload-to-mass ratios in parallel robots is generally accompanied by a reduction in workspace compared to similarly-sized serial robots.

Solving the forward kinematics equations for a serial robot is relatively straightforward, but their inverse kinematic solutions are much more involved. Conversely, the inverse kinematics solution for a parallel robot is much less effort than its forward kinematics solution, which is an ongoing area of research for some types of parallel robots.

1.3 Stewart Platform

The Stewart Platform is a common type of parallel manipulator and possesses six degrees of freedom. In their paper, Cruz, Ferreira, and Sequeira [5] state that "the generic Stewart-Gough platform is composed of two rigid bodies connected through a number of prismatic actuators as in a parallel arrangement of kinematic chains. Usually six actuators are used, pairing arbitrary points in the two bodies."

The actuators are connected to the rigid bodies by spherical or universal joints. This would mean the Stewart Platform is a spherical-prismatic-spherical, or SPS, robot. Although the joint positions can be arbitrary, evenly spacing them results in special cases. When the joints are spaced 60° around the base and the moving platform, this is called a 6-6 configuration. When a pair of actuators share the same joint spaced 120° on the moving platform, this is called a 6-3 configuration. 120° spacing on top and bottom platforms results in a 3-3 configuration as shown in Figure 1.2.



(a) Type 6-6 (b) Type 6-3 (c) Type 3-3

Figure 1.2: Stewart Platform Configurations

1.3.1 Geometry Stability

In a literature survey, a true type 6-6 Stewart Platform was never found to be used in practice. Typically, platforms were designed as a mix between the 6-6 and 3-3 configuration. Instead of pairs of actuators sharing the same joint, each actuator had its own joint like in a 6-6 configuration; pairs of joints were separated by a small gap, resulting in a geometry visually similar to Type 3-3. A diagram of this geometry is shown in Figure 1.3 and example platforms are shown in Section 1.3.4.

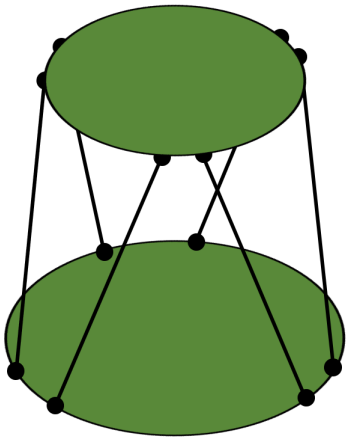


Figure 1.3: Typical Stewart Platform Joint Configuration

Further analysis reveals that a true 6-6 platform with 60° spacing between all joints is unstable because there is no force balance, allowing the top platform to rotate about its axis. This is illustrated in Figure 1.4. When a small disturbance occurs, the links cannot exert forces in opposite directions resulting in a couple about the links. Spherical joints cannot resist this moment, leading to a collapse. This shortcoming of a true 6-6 configuration shows the value of 6-3, 3-3, and comparable geometries.

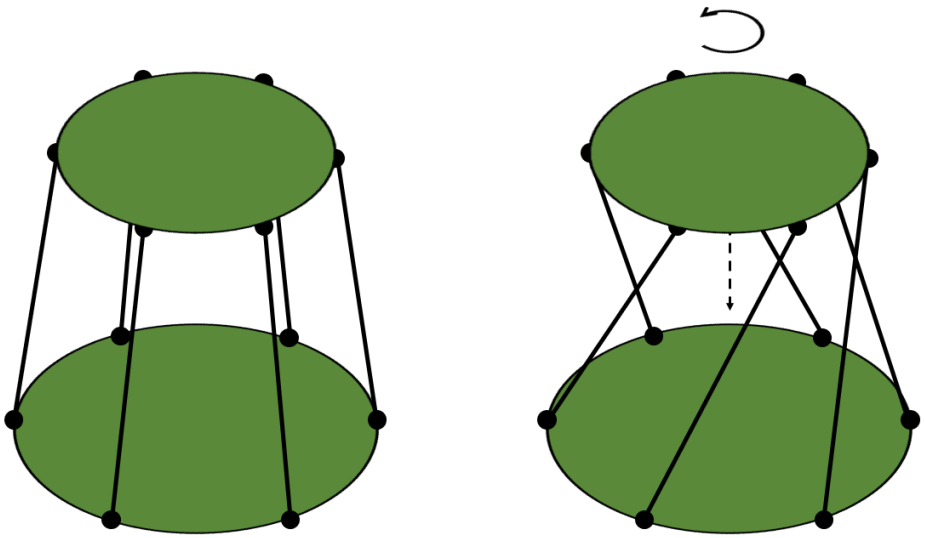


Figure 1.4: Unstable 6-6 Configuration

1.3.2 History

The early stages of robotics were dominated by serial robots, which drew heavy inspiration from the human arm and its utility. It was the contributions by Stewart and Gough that brought parallel robots under the scope of further research and development, especially in industry in recent years. According to Merlet [15], Eric Gough established a closed-loop kinematic mechanism and its principles in 1947 and built a prototype for tire testing in 1955. In the 1960s, the rise of aeronautics created a demand for flight simulators. D. Stewart published "A Platform with Six Degrees of Freedom" in 1965, discussing a triangular parallel mechanism design approach for the simulators. While Stewart had important contributions to flight simulator development, his published design was not used in practice. On the other hand the tire-testing mechanism designed by Gough, who was one of the reviewers of Stewart's paper, found heavy usage in industry. Gough's Tire Testing Machine is shown in Figure 1.5 and used six jacks to exert combined loads to test Dunlop tires. Regardless,

the platform has become known as the Stewart-Gough Platform, or more informally as the Stewart Platform.

Figure 1.5: Tire Testing Machine Application of Stewart Platform [13]

1.3.3 Kinematic Equations

The Stewart Platform kinematics are derived from analysis of its six kinematic chains. The following derivation is drawn from Dr. Saeed Niku's inverse kinematic derivation of the Stewart Platform, found in the 3rd edition of his book, *Introduction to Robotics: Analysis, Control, Applications* [17].

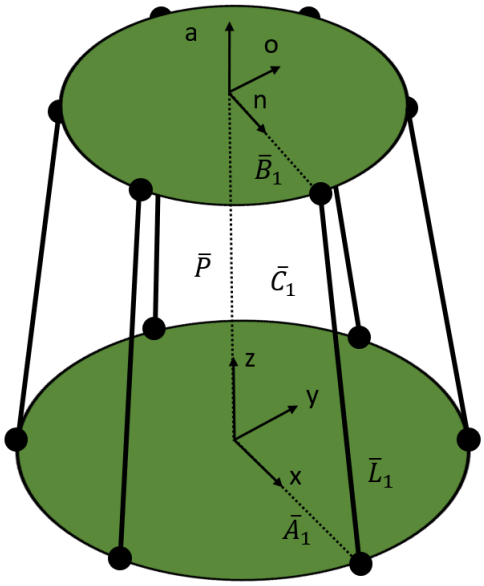


Figure 1.6: Home Position of Stewart Platform Robot

A fixed reference frame xyz is placed at the center of the platform base. The z-axis is normal to the base and the x-axis is parallel to it, pointed towards a spherical joint. A moving reference frame noa is similarly placed at the center of the moving platform with the a-axis normal to the moving platform face and the n-axis pointed towards a spherical joint. The spherical joints are assumed to be connected through the linear actuator. Frame noa rotates by θ,ϕ,ψ respectively.

Four parts compose each kinematic chain $\overline{C}$: $\overline{A}$ connects the origin of the fixed reference frame and base spherical joint at an angle from x. $\overline{B}$ connects the origin of the moving reference frame and moving platform spherical joint at an angle from n. $\overline{L}$ connects the spherical joints. $\overline{P}$ connects the origins of the two frames, and is common for all six chains. The purpose of the inverse kinematics is to determine the length L of each linear actuator, given the position P and orientation of frame noa. The vector equation to determine each actuator length is

$$\overline{L_i} = \overline{P} + \overline{B_i} - \overline{A_i} \tag{1.1}$$

for i = 1..6.

For chain $\overline{C_1}$, $\overline{A_1}$ and $\overline{B_1}$ lie along the x-axis and n-axis respectively and their angles are zero. In a Type 6-6 Stewart Platform, subsequent chains will be multiples of 60° from the x and n axes. This is not always the case, and the remaining angles can be arbitrary values; in practice, Stewart Platforms will have some degree of symmetry.

Following the vector convention of *Introduction to Robotics* by Dr. Saeed Niku [16], $\overline{A_i}$ for the general case can be written as

$$\overline{A_i} = \begin{bmatrix} Acos(i_{th}) \\ Asin(i_{th}) \\ 0 \\ 1 \end{bmatrix} \tag{1.2}$$

where the i_th term is the angle that $\overline{A_i}$ makes with the x axis.

$\overline{A_i}$ for i=1..6 will remain constant throughout the motion of the Stewart Platform for a given geometry.

Similarly, $\overline{B_i}$ can be written as

$$\overline{B_i} = \begin{bmatrix} Bcos(i_{th}) \\ Bsin(i_{th}) \\ 0 \\ 1 \end{bmatrix} \tag{1.3}$$

for the moving platform when it is in the home position as shown in Figure 1.6. The home position is achieved after the robot is calibrated, and all the actuators are at their minimum length.

Because $\overline{B_i}$ is attached to moving frame *noa*, its components will change as the platform rotates. To account for this rotation, $\overline{B_i}$ must be pre-multiplied by rotation matrices. Using the variables from the moving reference frame, the rotation matrices are:

$$Rot(x,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix} \tag{1.4}$$

$$Rot(y,\phi) = \begin{bmatrix} cos\phi & 0 & sin\phi \\ 0 & 1 & 0 \\ -sin\phi & 0 & cos\phi \end{bmatrix} \tag{1.5}$$

$$Rot(z,\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

Equations 1.4, 1.5, and 1.6 are from equations 2.20 and 2.21 in *Introduction to Robotics* [16]. These 3x3 rotation matrices only account for rotation. They can be expanded to 4x4 to include position, in which the fourth row and column are zero, except for a value of one located in element 4,4.

In order to determine the value of $\overline{B_i}$ after rotation for each kinematic chain, it must be pre-multiplied by the above rotation matrices, yielding:

$$\begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{rot} = \Big[Rot(z,\psi)\Big] \Big[Rot(y,\psi)\Big] \Big[Rot(x,\theta)\Big] \begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{home} \tag{1.7}$$

Multiplying $Rot(z,\psi)$ and $Rot(y,\phi)$ yields

$$\Big[Rot(z,\psi)\Big] \Big[Rot(y,\psi)\Big] = \begin{bmatrix} cos\phi cos\psi & -sin\psi & cos\psi sin\phi \\ cos\phi sin\psi & cos\psi & sin\phi sin\psi \\ -sin\phi & 0 & cos\phi \end{bmatrix} \tag{1.8}$$

Multiplying the result of Equation 1.8 by $Rot(x,\theta)$ yields

$$\begin{bmatrix} cos\phi cos\psi & -sin\psi & cos\psi sin\phi \\ cos\phi sin\psi & cos\psi & sin\phi sin\psi \\ -sin\phi & 0 & cos\phi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix} =$$

$$\begin{bmatrix} cos\phi cos\psi & cos\psi sin\phi sin\theta - cos\theta sin\psi & sin\psi sin\theta + cos\psi cos\theta sin\phi \\ cos\phi sin\psi & cos\psi cos\theta + sin\phi sin\psi sin\theta & cos\theta sin\phi sin\psi - cos\psi sin\theta \\ -sin\phi & cos\phi sin\theta & cos\phi cos\theta \end{bmatrix} \tag{1.9}$$

After combining the rotation matrices, Equation 1.7 becomes

$$\begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{rot} = \begin{bmatrix} cos\phi cos\psi & cos\psi sin\phi sin\theta - cos\theta sin\psi & sin\psi sin\theta + cos\psi cos\theta sin\phi & 0 \\ cos\phi sin\psi & cos\psi cos\theta + sin\phi sin\psi sin\theta & cos\theta sin\phi sin\psi - cos\psi sin\theta & 0 \\ -sin\phi & cos\phi sin\theta & cos\phi cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{home} \tag{1.10}$$

$[\overline{B_i}]_{home}$ is known from the Stewart Platform geometry, and θ, ϕ, and ψ are known upon specifying the orientation of the moving platform. Therefore, $[\overline{B_i}]_{rot}$ can be solved. Expanding the vectors Equation 1.1 into component form yields

$$\begin{bmatrix} L_{ix} \\ L_{iy} \\ L_{iz} \end{bmatrix} = \begin{bmatrix} P_{ix} \\ P_{iy} \\ P_{iz} \end{bmatrix} + \begin{bmatrix} B_{ix} \\ B_{iy} \\ B_{iz} \end{bmatrix} - \begin{bmatrix} A_{ix} \\ A_{iy} \\ A_{iz} \end{bmatrix} \tag{1.11}$$

for i=1..6.

The linear actuator lengths are the desired quantity from the kinematics. The magnitude of each length is determined by

$$|L_i| = \sqrt{(L_{ix})^2 + (L_{iy})^2 + (L_{iz})^2} \tag{1.12}$$

for i=1..6.

These calculated lengths become the setpoints for controlling the Stewart Platform.

1.3.4 Applications

There are several applications suited for a Stewart Platform. Flight simulators make use of the platforms 6 DOF to simulate orientation and motion. The Stewart Platform can accurately orient to the correct roll, pitch and yaw, and their high-dynamic capability can create linear and nonlinear forces, like a bump. Shake tables make use of this capability to create high-acceleration, low-displacement moves. Fiber-optic equipment can be aligned using a Stewart Platform because they must be accurately positioned and oriented, enabling the light to refract and the optic signal to operate

properly. In astronomy, Stewart Platforms are used in spherical radio telescopes and trajectory tracking, such as the one shown in Figure 1.7.

Figure 1.7: AMiBA Radio Telescope in Hawaii [12]

1.3.5 Stewart Platform Specifications

The application of this Stewart Platform is for education and experimentation to promote the "Learn By Doing" experience in regards to parallel robots. Thus, there are not external deterministic requirements or specifications that the Stewart Platform must meet. Instead, an internal specification will be set. Quantitative specifications are found in Table 1.1.

Table 1.1: Stewart Platform Specifications

Metric	Specification	Actual Value
Mechanical Properties		
Weight	< 40 lb	23.7 lb
Length	< 24 in	21 in
Width	< 24 in	21 in
Height	< 36 in	31 in
Range of Motion		
θ	$> 30^\circ$	44°
ϕ	$> 30^\circ$	54°
ψ	$> 60^\circ$	90°
X	6 in	13.5 in
Y	6 in	14.5 in
Z	6 in	25.0 in
Accuracy		
Angular	$< 2^\circ$	1°
Positional	0.25 in	0.25 in

The mechanical properties were set to allow the robot to fit comfortably on a desk surface, and be moved by one person. The specified range of motion demonstrates that the Stewart Platform can properly utilize each of its six degrees of freedom. The robot is not being used for tooling or processing applications, and does not need to be extremely accurate; it must be accurate enough to pass visual observation by the user.

There are also some qualitative specifications set for the Stewart Platform. The platform should be safe for students to operate and experiment with. It should be designed to be easy to manufacture using Cal Poly Machine Shop resources. Chosen vendor and custom parts used should be readily available or easy to replace, keeping the Stewart Platform serviceable. It would be educationally beneficial to reconfigure the robot. Typically, the geometry of a robot is optimized to solve a problem or complete a task. For this lab purpose, exploring multiple configurations and their benefits and limitations.

Chapter 2

MECHANICAL DESIGN

There were several important foci of the mechanical design of the Stewart Platform. The first was to create a platform that was easy to assemble and disassemble into various configurations. As mentioned previously in Section 1.3, the Stewart Platform has three primary configurations, 6-6, 6-3, and 3-3. To avoid limiting the Stewart Platform to just one configuration, it was designed to be re-configurable within minutes with just a hex key.

Another design focus was ability to manufacture and assemble the Stewart Platform with only university resources. This focus was coupled with budget considerations, as special tooling and outsourcing manufacturing could significantly increase cost. Therefore, all components were required to be manufactured in-house using machine shop resources.

2.1 Linear Actuators

At its core, a linear actuator is a device that causes movement along an axis. This can typically be achieved with hydraulic, pneumatic, or electrical power. These mediums of actuation were considered with regards to the Stewart Platform, specifically its performance requirements and intended lab environment. The linear actuators that operate the Stewart Platform must have a form of positional feedback, because the joint lengths are the controlled variables by which the Stewart Platform is actuated. The actuators are primarily oriented vertically in the Stewart Platform, and must be capable of both supporting and moving a payload.

There are three common characteristics in linear actuator terminology: stroke, force, and speed. The stroke is the difference between the actuator length when fully extended and fully retracted. The force is the maximum load the linear actuator can handle and this can be rated for both static and dynamic conditions. Speed is simply how fast the actuator extends or retracts.

Hydraulic cylinders with servo-valves under position control would provide sufficient positional accuracy for a Stewart Platform to function properly. Additionally, under a typical hydraulic pressure of 3000 psi, a single piston with a 0.5 inch cylinder could actuate with a force up to 589 lb. While hydraulics satisfies the force and positional control requirements, it requires dedicated infrastructure and relatively high cost, as hydraulic systems require pumps, filters, valves, hoses, and tubing. Therefore, hydraulic actuation is inappropriate for this Stewart Platform.

Pneumatic systems, generally speaking, can be created with less cost and infrastructure than hydraulic systems, and provide sufficient force within the defined scope of this Stewart Platform. However, the positional control of pneumatic cylinders is difficult due to the compressibility of air and its nonlinear behavior. Even industrial, purpose-built pneumatic positioning controllers [24] after optimized tuning, can achieve as as little as ± 1 mm (or ± 0.039 inches) positional accuracy. Products that can achieve these specifications are beyond the budget for the project, and pneumatic actuation is also inappropriate for the Stewart Platform.

Electric linear actuators are another common means of linear actuation. Firgelli Automations [10] gives an overall introduction to electric linear actuators. They convert the rotational motion of a motor into linear motion through a gear train, which connects to a lead screw and nut as shown in Figure 2.1.

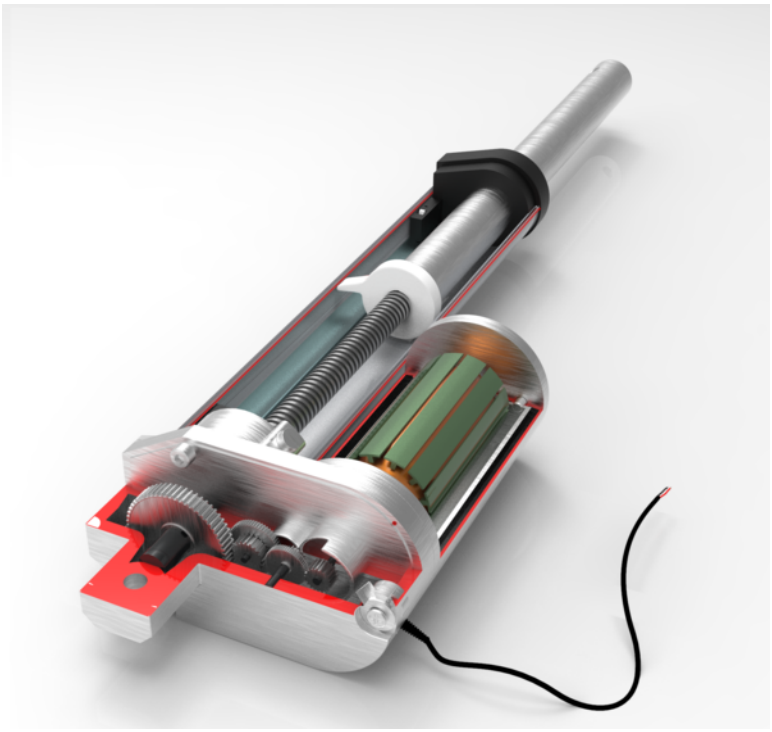


Figure 2.1: Section View of an Electric Linear Actuator [10]

Most electric linear actuators have factory-installed limit switches that cut the power to the motor once the actuator has reached its maximum or minimum travel distance. Typically, a 12 VDC motor is used to drive linear actuators, but DC motors at other voltages and AC motors are available as well. Some electric linear actuators also have position feedback, usually provided from a potentiometer.

Progressive Automations, Thomson, and Firgelli Automations offer linear actuators in their product line that range from miniature with light payload ratings, around 5 lb, to heavy duty models able to carry several hundred pounds.

The electric linear actuator is lower cost, relatively easy to integrate, can exert moderate forces, and is very suited to position control. These factors make electric actuators the best choice for this Stewart Platform.

2.1.1 Selection Guidelines

As described above, there are several specifications available for choosing electric linear actuators. The scope of the project helped dictate these specifications. The physical envelope of the robot was established to have an approximate 1.5 ft. by 1.5 ft footprint, with a vertical range between 1.5 - 2 feet. Using these guidelines, a stroke between 6 inches to 10 inches was desired. It is important to note that when fully extended, the overall length of the linear actuator will be at least twice the stroke. The intended usage of the Stewart Platform is for experimentation with kinematics and motion, and will not be carrying heavy loads or aligning precise equipment. Therefore, high-speed, low-load actuators are the best fit for this design. Non-hardware design decisions include low cost and ease of use and implementation, as excessive cost or time to implement would be harmful to the project completion and cause delays if spare parts are needed or installed. After looking through several vendors, the PA-14P-8-35 linear actuator offered by Progressive Automations best fits the design parameters.

2.1.2 Progressive Automation PA-14P-8-35

The PA-14P-8-35 is an electric linear actuator that has an 8-inch stroke and exerts a dynamic force of 35 lb. At no load, the actuator moves at 2 in/sec, drawing 1.0 A. At full load, the actuator moves at 1.38 in/sec and draws its maximum 5.0 A. It has a potentiometer that provides the positional feedback required to control the Stewart Platform. As of this writing, these actuators cost $138.99 each. While there are cheaper linear actuators, Progressive Automations provides a sponsorship opportunity which would retroactively reduce the price of the actuators if approved, making them budget-friendly. Finally, Progressive Automations has sample Arduino

code provided for controlling multiple actuators and their timing, monitoring their current, and more. Further information can be found in the PA-14P Datasheet [19] in Appendix A.

2.2 Magnetic Spherical Joints

The Stewart Platform has 6, 9, or 12 passive spherical joints, depending on the configuration of the platform. The contribution of spherical joints to the degrees of freedom of the platform was discussed in Section 1.3. A common hardware choice for a spherical joint is a ball joint, shown in Figure 2.2. The swivel angle is a characteristic of a ball joint that specifies the amount of rotation possible from the axis normal to the axis of the rod. This is an important characteristic because the range of the ball swivel affects the range of the platform operation. On McMaster Carr's website, most ball joint rod ends have a maximum ball swivel between 20°-30°[14].

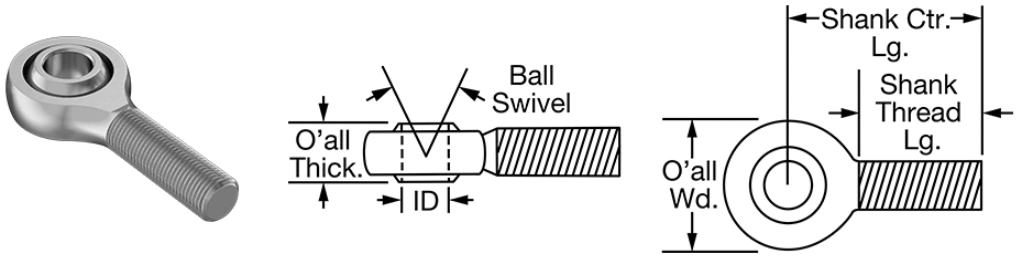


Figure 2.2: Ball Joint Rod Ends [14]

Another choice that satisfies the spherical joint requirement of the Stewart Platform is a magnetic spherical joint. These spherical joints are comprised of a steel ball with a protruding threaded rod, coupled with a Neodymium Iron Boron (NdFeB) permanent magnet housed in cylindrical base as shown in Figure 2.3a. The magnetic joints come in a range of sizes and holding force ratings. Table 2.1, adapted from AliExpress [1],

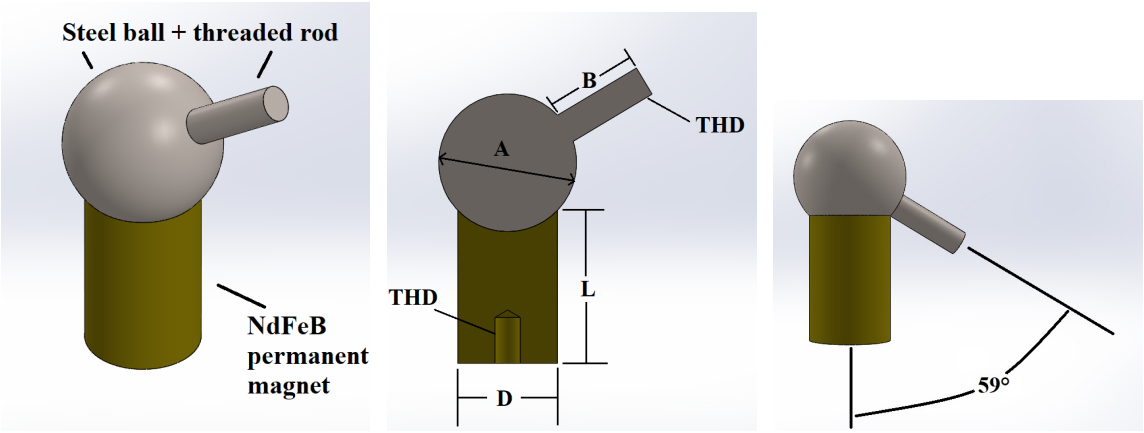
shows a range of common values for these magnets. The terminology in the table corresponds to the labels in Figure 2.3b, and all dimensions are in millimeters.

Table 2.1: Magnetic Spherical Joint Product Options [1]

Type	D	L	A	B	THD	Holding Force	Cost
KD310	10	20	10	12	M3	18 N	$3.65
KD312	10	20	12	12	M3	18 N	$3.95
KD413	13	20	13	12	M4	40 N	$4.84
KD418	13	20	18	12	M4	40 N	$5.11
KD625	20	25	25	16	M5	150 N	$6.55
KD725	25	35	25	16	M5	200 N	$8.85

Cost, size, and holding force were all considered when selecting the magnetic spherical joints. In general, all three of these characteristics increase together. A lower cost spherical joint is preferred, especially since 12 joints are needed. This must be balanced by the need for sufficient holding force. If the holding force is too low, then the magnetic joint may separate during normal operation which must be avoided. The holding force cannot be too high either; if a binding condition occurs, the joint needs to be able to fail and separate. This prevents the linear actuator from stalling and overheating while relieving the acrylic plate loadings. The maximum force the linear actuators can exert is 35 lb, equivalent to 156 N. The KD625 seems to perfectly meet this requirement but there is very little cushion between the holding force and actuator force. On the other end, the KD310 is the lowest cost joint, but is dwarfed by the linear actuators, and can hold little more than the weight of the linear actuators. The KD418 was chosen as a moderate cost, size, and holding force solution, and it has functioned as intended in operation of the Stewart Platform.

There are additional advantages in using the magnetic joints with this Stewart platform. The magnetic ball joints have a ball swivel angle of 242°, as shown in Figure 2.3c, around an order of magnitude greater than the ball joints previously mentioned.



(a) Spherical Components (b) Dimension Key (c) Angle Limit

Figure 2.3: Magnetic Spherical Joint Properties

This angle is well beyond the angles of the linear actuators relative to the base and removes the joints as a limiting factor of range of motion. Friction was a concern initially because of a lack of lubrication between the steel and magnet surfaces. Friction was very apparent when the steel ball was rotated against the magnet by hand, and smooth rotation was difficult to achieve. However, the rotation was much smoother once the linear actuator was attached to the joint. This is because the linear actuator provides several inches of leverage, whereas twisting the sphere or pushing the threaded rod only gives about an inch of leverage. When installed on the Stewart Platform, the actuators easily overcome friction and rotate smoothly.

2.3 Platform Material

Low weight, low cost, ease of manufacturing, and pleasing aesthetics were all considered when selecting the material for the top and base platform plates. Wood, acrylic, and aluminum were top material candidates. Weight is not critical for this application, but lower plate weight reduces the overall Stewart Platform weight. Additionally, the weight of the top plate takes away from the payload capability of the

platform. Wood and acrylic have lower densities than aluminum, but aluminum is stiffer, allowing for plate thickness and weight reduction; no material has a significant advantage for weight. Wood is around 2.5 times cheaper than acrylic and several times cheaper than aluminum for the same raw material volume. Reducing the plate thickness for aluminum still results in a much higher relative cost. All materials are able to be manufactured at Cal Poly; wood and acrylic can be cut using a laser cutter, and the aluminum can be cut with a water jet. While wood can be aesthetic, acrylic and aluminum are best suited to the Stewart Platform's more industrial and mechanical appearance.

Acrylic was chosen because it is the best overall fit due to its moderate weight, cost, and easy to laser-cut. The base and top plates are both 1/4 inch thick acrylic sheets and weigh 2.67 lb and 1.66 lb respectively. For reference, if made out of aluminum 6061-T6 of equivalent size, their weights would be 6.02 and 3.75 lb respectively. Laser cutting of the acrylic was straightforward and resulted in aesthetic Stewart Platform plates. Additionally, acrylic is transparent which is beneficial because the electronics under the base platform are visible, exposing students to the components that control and drive the system.

2.4 Shaft Couplers

In a Stewart Platform, all six linear actuators must actuate together in a coordinated manner. If they do not, then the position and orientation of the robot will be incorrect. At worst, a large deviation in one or more actuators can cause the robot to move outside of its envelope or achieve an unstable position, causing the magnetic joints to separate. It is evident that the Stewart Platform, and parallel robots in general, are very sensitive to actuator accuracy. This necessitates a mechanical solution called

compliance. Bram Vanderborght gives a helpful introduction to compliance, summarizing that a compliant member will allow deviations from its equilibrium when a force acts on it, whereas a stiff member will not, within the device limits [23].

The magnetic joints, linear actuators, and acrylic are considered stiff components. However, the acrylic is the least stiff of the three so if the robot encounters a binding position or an unintended loading condition, the acrylic plates will take the load, risking deformation and breakage. This should be avoided by placing a compliant member in series with the linear actuator. The compliant member must be in a compact and professional package, and removable if the Stewart Platform can be run without the compliant member.

Helical shaft couplings were chosen to be the compliant member because they satisfied the above requirements. Shown in Figure 2.4, they are primarily intended for torque transmission but offer the flexibility in axial and bending motions to provide the necessary compliance. The shaft couplings chosen are the WAC20-4mm-4mm Aluminum Alloy Couplings and the data sheet is located in Appendix B.

Figure 2.4: Helical WAC20mm-4mm-4mm Couplings

2.5 6-3 Configuration Geometry

In a 6-3 configuration, the six base joints are evenly spaced at 60° angles and the three top joints are evenly spaced at 120° angles. Two actuators have to share a joint in this configuration. When two magnets are attached to a single steel ball to accomplish this, there is interference shown in Figure 2.5, where the Stewart Platform is in a normal position and orientation.

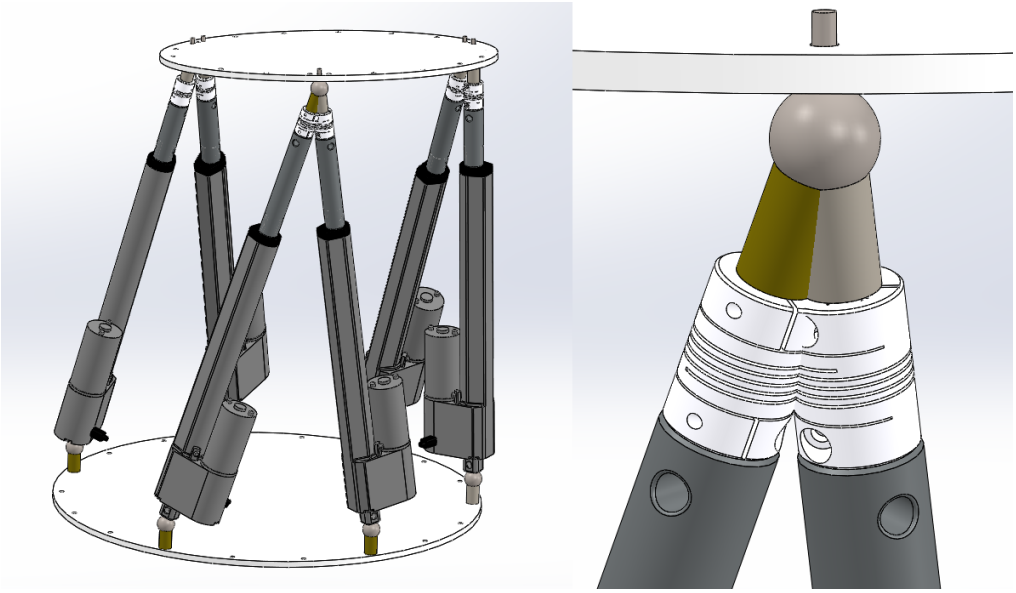


Figure 2.5: Range-of-Motion Limitation with Sharing Spherical Joint

The only way for a spherical joint to be shared by two magnetic joints is for their cylindrical axis to be greater than 92.9° apart as shown in Figure 2.6. This is only possible outside of the range-of-motion of the Stewart Platform. Therefore, the magnetic portions of the joints cannot share the same steel ball.

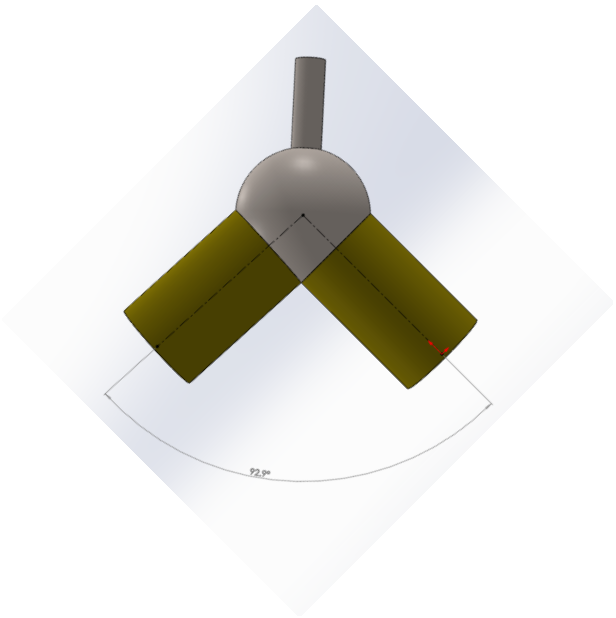


Figure 2.6: Range-of-Motion Limitation with Sharing Spherical Joint

To fix this, the spherical joints had to be separated. Separation means the robot will not truly be in a 6-3 configuration, only close to it. The geometry of the spheres drove the angle separation of 3.47° between each joint from the 120° centerline is shown in Figure 2.7.

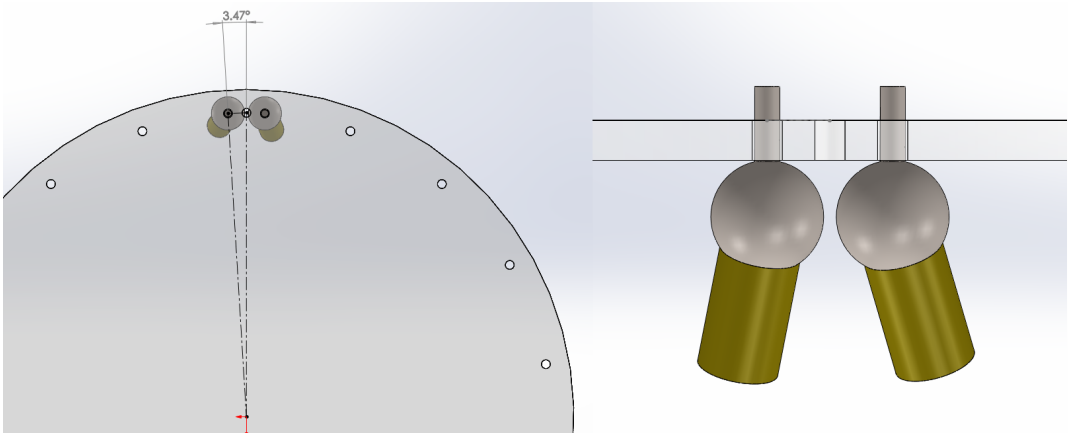


Figure 2.7: Joint Angle of Spherical Joint in 6-3 Configuration

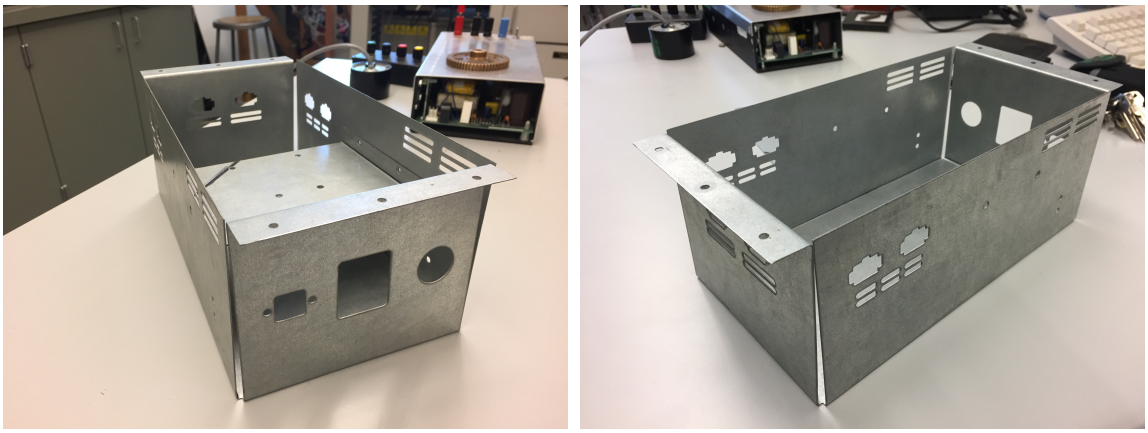
This solution creates a new problem; the joint angles for the 6-3 configuration are no longer at 120°because of this angle of separation. This joint angle must be included in the modeling and kinematic equations of the Stewart Platform. This implementation is discussed in Section 4.1.1.

2.6 Electronics Enclosure

There are a few key electronic components required for the operation of the Stewart Platform. These components include a power supply, microcontroller, motor driver, and connection ports, plugs, and switches. These components are further discussed and detailed in Chapter 3. The components must be housed in an electronics enclosure for safety and reliability, protecting users from contact with live power and internal components from careless or accidental handling. Optimally, the enclosure should be relatively lightweight and portable, provide sufficient airflow, and be easy to manufacture at a low material price. Furthermore, the enclosure is an appropriate means to connect the PA-14P motor and potentiometer through the included 6-pin connector.

Sheet metal was chosen to be the electronics enclosure material because of its easy manufacturability, high strength, and low cost. Alternatives considered were plastic 3D printed enclosures or acrylic. The volume was too large and geometry too simple to justify all the 3D printed material. Acrylic could have been cut on the a laser cutter, but each surface would have to be fastened together. The sheet metal enclosure was designed, cut with a water jet, and bent using Cal Poly resources. The enclosure is shown in Figure 2.8

The front of the enclosure has three cutouts for connecting USB and AC power along with a power switch. The rear has six cutouts for the female 6-pin Molex connectors



(a) Enclosure Front (b) Enclosure Rear

Figure 2.8: Electronic Enclosure

that attach to the linear actuator connectors. On the walls of the enclosure are vents that allow air circulation. The enclosure will be attached underneath the base of the Stewart Platform, making the Stewart Platform and its electronics a complete unit.

The power supply resides in the bottom of the electronics enclosure. Suspended above it is the Arduino due and motor driver, to be discussed in Chapter 3, which helps keep the enclosure compact. This uninstalled bracket is shown in Figure 2.9. The drawings for all manufactured mechanical parts are located in Appendix C.

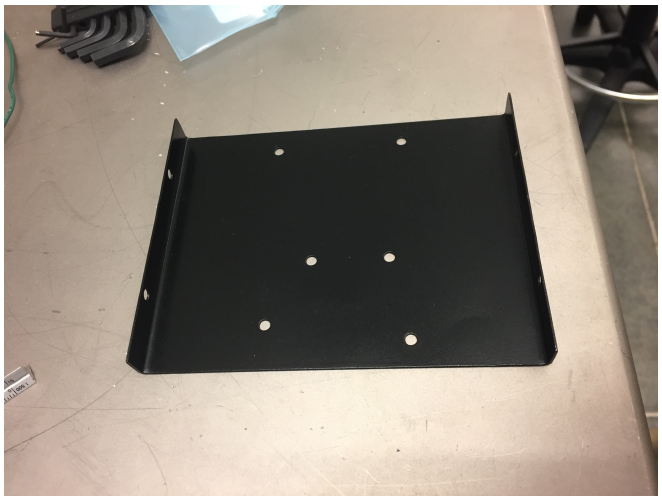


Figure 2.9: Arduino Due Mount

When the 6-pin female Molex connectors were inserted into the enclosure walls, the wall thickness was insufficient to retain the connectors. When the linear actuator was plugged in, the force would push the connector back into the enclosure, which would be impossible to bring back into place when the enclosure is installed under the Stewart Platform base. Retainers were printed to increase the wall thickness, allowing the connectors to stay in place when the linear actuators are plugged in. These retainers are shown in Figure 2.10. An additional benefit from these retainers is a permanent labeling solution, ensuring that the actuator number cannot be switched or removed.

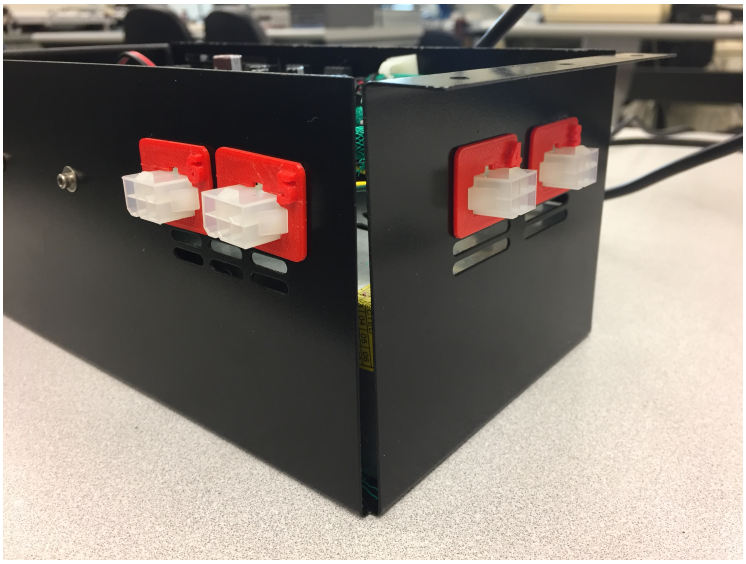


Figure 2.10: Molex Connector Retainer and Label

The bill of materials for all mechanical components of the Stewart Platform can be found in Appendix D. The total weight of the Stewart Platform is 23.7 pounds, which includes mechanical and electrical hardware. The electronics are discussed in Chapter 3.

Chapter 3

ELECTRONIC DESIGN

The electronic design of the Stewart Platform must meet the operational needs of the robot. The robot needs power to drive the actuators through its motions, which need to be accurately calculated and controlled. A controller can apply a control loop and send actuation signals based on calculated inputs and current positions of the linear actuators.

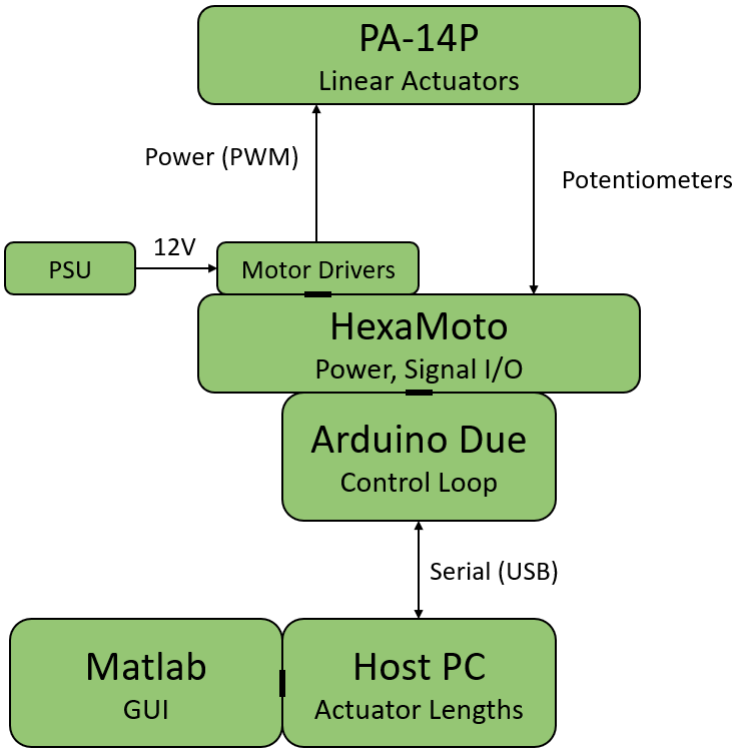


Figure 3.1: Stewart Platform System Diagram

A high level overview of the system is presented in Figure 3.1. At the top of the diagram are the PA-14P actuators discussed in Chapter 2. This chapter discusses the design and realization of the middle portion, the electronics. A controller, motor

drivers, and a power supply comprise the core electronic components. The bottom of the diagram incorporates the code inside the controller and the user interface with the Stewart Platform, discussed in Chapter 4.

3.1 Microcontroller - Brief Overview

A microcontroller uses a microprocessor, along with peripherals including read-only memory (ROM), random-access memory (RAM), and input and output capabilities (I/O) to make a computer on a chip [4]. They are typically used as part of an electronic or mechanical system with dedicated functionality. This is known as an embedded system, which are often subject to real-time constraints. There are countless microcontrollers available to control the Stewart Platform.

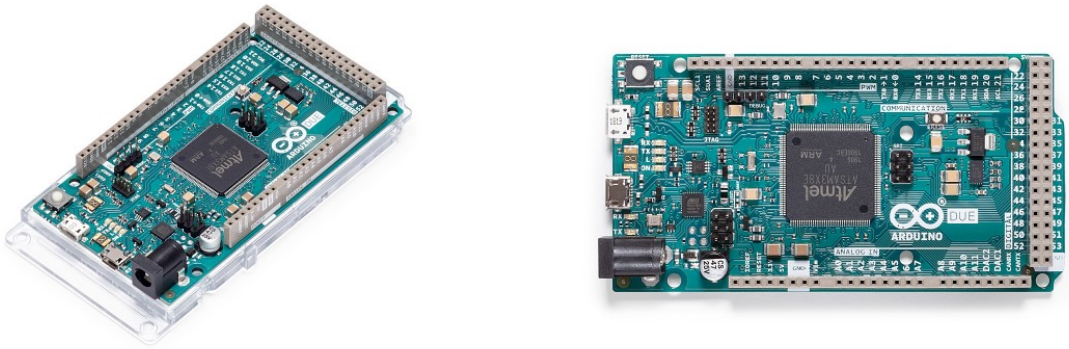
3.1.1 Microcontroller Selection

An Arduino board was chosen as the microcontroller platform to control the robot for several reasons. Arduinos are generally implemented in an easy-to-use prototyping environment comprised for makers, hobbyists, students, and more. The setup, usage, and intended audience of an Arduino platform make it a suitable choice for the Stewart robot. The board's design and software are open-source, providing a basis to make modifications as necessary. For board designs and hardware, modifications can be made to existing designs to implement additional features or tailor performance to a specific application. Hardware modifications were not necessary for the Stewart Platform, but having this option was helpful in case the necessity did arise further into the design. An additional benefit to using an Arduino is the simplicity of replacement. If the Arduino board becomes faulty, it can be replaced within a few days by ordering

a ready-made board, installing it, and downloading the project to the new board. This method is a near-turnkey solution to replacing the microcontroller.

Availability of usable software and resources was another driving factor in deciding to use the Arduino platform. This served to reduce development time, errors, and shift the development focus from setup to application and refinement. Progressive Automations has several examples of Arduino code that apply directly to their linear actuators. Examples of code applicable to this project include "Controlling Multiple Actuators with the Multimoto Arduino Shield" and "Controlling the Timing of a Single Linear Actuators Motion" [20]. The provided code helped with the setup and initialization required for working with the PA-14P actuators. Further details regarding the software used for operating the robot are discussed in Section 4.2.

Arduino offers several board choices, with varying levels of features and performance. Boards include the Uno, Nano, Mega, and Due. The selected board, at the minimum, has to have available a minimum number of necessary inputs and outputs. The PA-14P actuators each have a potentiometer, so there must be 6 analog inputs to read the position of the actuators. The motor driver, covered in further detail in Section 3.2, outputs voltage at a level and polarity based on a PWM pin and a direction pin. The motor drivers also share an Enable and Disable pin. This required 14 digital outputs, 6 of which must be capable of PWM output. The Arduino Mega 2560 and Due fulfill both requirements. They have some similarities in regards to I/O capabilities, but there are differences in their features. However, the Due offers a clock 5.25x faster than the Mega 2560, twice the flash memory size, and 12 times the SRAM at the same price point. The Due is selected as the microcontroller for controlling the Stewart Platform, and is shown in Figure 3.2.



(a) Isometric View (b) Top View

Figure 3.2: Arduino Due [3]

3.1.2 Arduino Due Specifications

The following section describes the selected Arduino Due in further detail and its application to the Stewart platform. Table 3.1 summarizes some of the technical specifications of the Arduino Due.

Table 3.1: Arduino Due Technical Specifications [3]

Feature	Value
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Digital I/O Pins	54
Analog Input Pins	12
Analog Output Pins	2
Flash Memory	512
SRAM	96 KB
Clock Speed	84 MHz

A distinct characteristic of the Arduino Due is the presence of two USB ports. According to Arduino's web page on the Due, one port is a Programming port and the other is a Native port. The Programming port is connected to an ATmega16U2, which connects to the SAM3X though hardware UART and also provides a virtual

COM port to a connected computer. The Native USB connects to the SAM3X, enabling serial communication over USB. The Native port can also emulate various USB devices, or act as a USB host. Both ports can be used for programming the Due, but there are some notable differences. When the Programming port is connected at 1200 baud and opened or closed, a SAM3X "hard-erase" procedure is performed, activating the Erase and Reset pins before UART communication. In comparison, the Native port performs a "soft-erase" procedure when opened and closed at 1200 baud. Flash memory is erased and the bootloader restarts the board. Programming via the Programming port is preferred because it is more reliable compared to the soft-erase procedure, which may not perform when the microcontroller has crashed. However, the Programming port is limited to a baud rate of 115200, which can be an issue if high-speed serial communication is required. The Native port is capable of much higher serial communication speeds because of the USB connection, and setting a baud rate in the Arduino sketch is ignored. This functionality and performance difference was notable in programming and operating the robot. Uploading code using the Native port was several times faster than uploading via the Programming port. However, the Programming port was used when uploaded sketches crashed because of its reliability. For normal operations, functionality, and data transmission, the Native port was used [3]. All further information on the Arduino Due, including overview, technical specifications, and documentation, are available on the cited Arduino Due web page.

3.2 HexaMoto Shield

The Arduino Due is capable of controlling motors through its PWM outputs but lacks the hardware to power them so an additional motor driver board is necessary

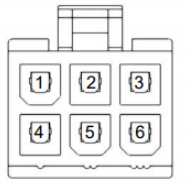
to complete the electronic hardware design. Two approaches include purchasing an aftermarket motor driver board or designing and assembling a custom board.

3.2.1 Design Requirements

The motor driver shield will need to have six channels to power each linear actuator. Each motor driver must provide a form of speed control. The robot operation would not be acceptable if the actuators could only operate at maximum speed. Additionally, it would be ideal to incorporate the potentiometer signals on the Arduino shield. This would provide an inclusive solution in which all inputs and outputs (except for 12V motor power) interface with the Arduino Due through the shield.

A potentiometer outputs a voltage as its analog signal. An 8-bit analog-to-digital converter (ADC) can output 256 discreet values. For a 4-inch stroke, each discreet change in the ADC reading of the potentiometer corresponds to a length of 0.0156 inches. For a 12-inch stroke, this value is 0.0469 inches. Similarly, a 10-bit ADC that outputs 1024 values means each increment is a change in length of 0.004 inches for a 4-inch stroke and 0.012 inches for a 12-inch stroke. It is important to note that ADCs have sources of measurement error, and they are not completely accurate to the above values.

In regards to electronics integration, Progressive Automation's DC linear actuator with a potentiometer will have five wires. Two wires for motor power will provide +12VDC and 0VDC (GND) and three for the potentiometer will provide +5VDC, signal and GND. The shield should accommodate these five lines for each of the six actuators. This is illustrated in the PA-14P datasheet shown in Figure 3.3.



Motor		Potentiometer			
4	5	2	3	1	6
-	GND (White wire)	Actuator Negative (Red Wire)	Potentiometer (Blue Wire)	Actuator Positive (Black Wire)	5V (Yellow Wire)

Resistance*	Number of Turns	Tolerance
0-10kΩ	10	+/- 5%

*Actual resistance value may vary within the 0-10kΩ range based on stroke length

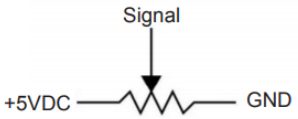
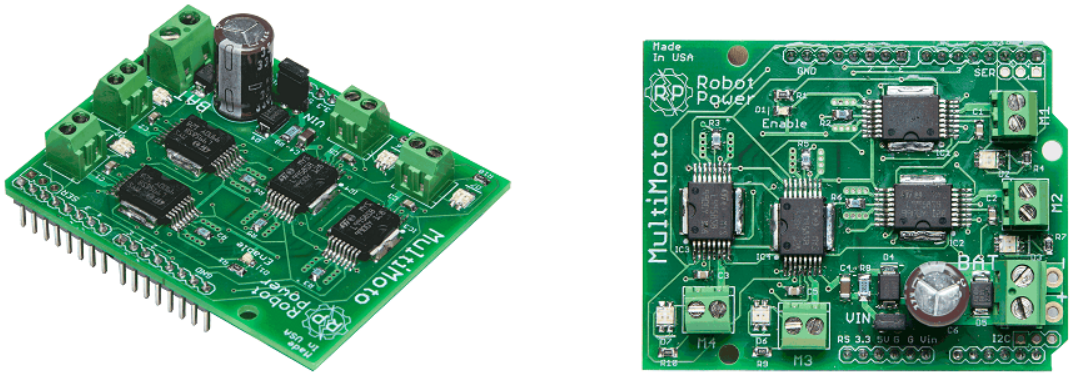


Figure 3.3: Progressive Automation PA-14P Pinout [19]

3.2.2 Robot Power's MultiMoto Arduino Shield

Progressive Automations has various products suggested for controlling and powering their linear actuators. One such product is a MultiMoto Arduino Shield, designed by Robot Power as shown in Figure 3.4.



(a) Isometric View (b) Top View

Figure 3.4: Robot Power MultiMoto Arduino Shield [18]

According to Robot Power's website, the MultiMoto features four independently controlled and fully-reversible channels due to the use of the H-bridge design. Each channel can handle 6.5A continuous on a 6-36V input voltage. With current and over-temp limiting safety features, the general design and scope of the board is appropriate for the environment and application for the Stewart Platform. However, this board is two channels short of controlling the 6 linear actuators required on the Stewart Platform. It will however provide to be a useful reference point in designing a purpose-built driver. The schematics for the Robot Power Multimoto Arduino Shield are included in Appendix E.

3.2.3 HexaMoto Design

The HexaMoto is designed to meet all the requirements set in Section 3.2.1. It uses six L9958SBTR motor drivers, pin headers to interface with the Arduino Due, and 13 screw terminals for I/O. The specifics of these components are detailed in the following subsection. The schematic for the HexaMoto may be found in Appendix F.

Investigating different PCB vendors, JLCPCB offered two-layer boards under 100 mm by 100 mm at $2 for quantity five. This drove the overall dimension of the HexaMoto. By its own virtue, the shield must have pin headers that match with the Arduino due, so these were placed about the center plane. The motor drivers and their screw terminals were arranged around the bottom half of the board, and the potentiometer screw terminals were placed around the top half perimeter.

Figure 3.5 shows the board layout with the ground planes deleted from the top and bottom layer to illustrate the traces and polygons. Signal traces were routed with 16 mil traces, and power traces were routed with 24 mil to 32 mil traces. Traces on the top of the board were primarily routed horizontally and traces on the bottom were

primarily routed vertically. This trace design helps keep traces organized and reduce overlapping and space issues, which are common occurrences when routing the last few traces of a board.

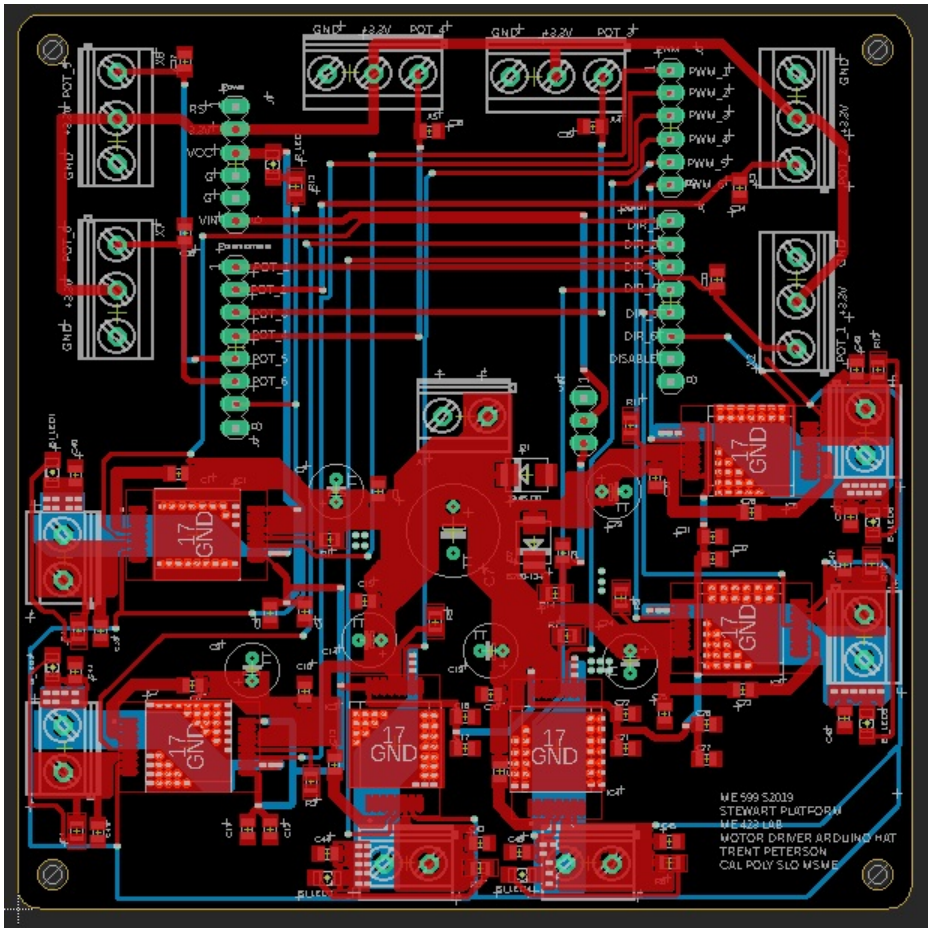


Figure 3.5: HexaMoto Board Layout, Top Layer

The board needs to support up to 5A per motor channel, which requires very large traces to reduce resistance. Polygons are used instead of traditional traces to carry this high current. The screw terminal at the center of the board takes the 12V input from the power supply and distributes it to the motor drivers through the polygons, which are between 50-100+ mils wide.

3.2.4 Component Details

3.2.4.1 STMicroelectronics L9958SBTR Motor Driver

The primary component of the HexaMoto Shield is the motor driver. The MultiMoto uses the L9958 motor driver [21] and it contains the functionality required for driving a Progressive Automation linear actuator. It is designed to drive brushed DC motors, and can output up to 8.6A at 4V to 28V. Protection features include undervoltage and overvoltage protection, and H-bridge over temperature and short circuits [22]. The block diagram and pinout of the selected PowerSO16 power package are shown in Figure 3.6.

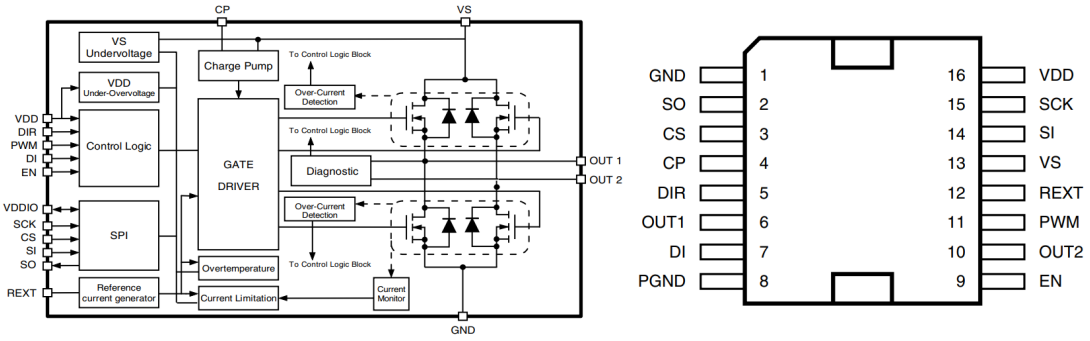


Figure 3.6: L9958 Block Diagram and PowerSO16 Pinout [22]

The L9958 motor driver is controlled by PWM, DIR, EN, and DI inputs. The EN (Enable) and DI (disable) pins determine whether the bridge is set to Tri-state or On-state. The DIR (Direction) and PWM (Pulse Width Modulation) pins control the direction and speed of the connected motor. Serial peripheral interface (SPI) is available for this motor driver, but its usage was not required and was not connected on the motor driver. The application circuit in the L9958 manual is shown in Figure 3.7.

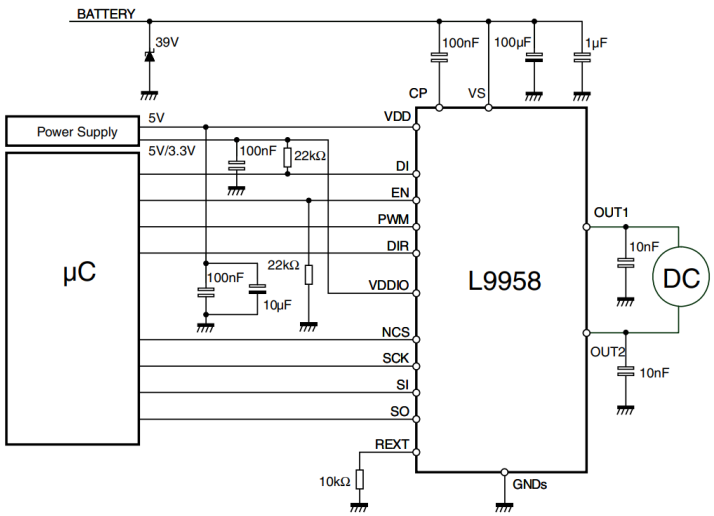


Figure 3.7: L9958 Application Circuit [22]

All associated resistors and capacitors were included on the HexaMoto Shield. The 10nF capacitors on the outputs are decoupling capacitors, to reduce high frequency noise to the outputs. The 100μF and 1μF capacitors connected to V_s are for decoupling to increase the robustness of the output short protection.

An LED was used to visually indicate the direction of each L9958 motor driver's output. The extension of the linear actuators is indicated by a green output. Retraction of the linear actuators are indicated by a red output. This was accomplished by connecting the LED between OUTA and OUTB of the L9958 motor driver. This LED is made by SunLED, and shown in Figure 3.8.

Figure 3.8: SunLED Bi-Directional LED [9]

3.2.4.2 I/O Hardware Components

Screw terminals were used for connecting the linear actuator wiring. 2-position screw terminals were used for the input voltage from the DC power supply unit, discussed in Section 3.4, and for the six outputs to the linear actuator motors. 3-position screw terminals, as shown in Figure 3.9b, connect the linear actuators' potentiometer, comprised of the sensor, 3.3V, and GND. Pin headers are used to easily connect the HexaMoto Shield to the Arduino Due. Standard 0.10 inch pin headers are shown in Figure 3.9a. Note that the Arduino Due operates on +3.3VDC logic instead of +5VDC, hence the difference in the voltage used for the potentiometer.

(a) Pin Header, 0.10 in Pitch [7]

(b) Screw Terminal, 3 Position[8]

Figure 3.9: HexaMoto Shield I/O Hardware

3.3 HexaMoto Shield Assembly

The HexaMoto Shield assembly required soldering both through-hole and surface-mount parts. Through-hole parts are typically easier to solder because they are larger, easier to handle, and can be soldered on the opposite side of the board. Surface mount parts can be more difficult, especially in smaller package sizes. For this application, surface-mount components were soldered using solder paste, a stainless steel stencil,

and an oven. The stencil is a sheet of metal that is laser cut to match the solder pads on the PCB. The solder paste is applied to the stencil and distributed, resulting in a small and even deposit of solder paste on each pad. The components are placed on the PCB and then the PCB is put into an oven, allowing the solder to reflow. Figure 3.10 shows the HexaMoto Shield after reflow and cooling. Please note that solder paste can be irritating to the eyes, skin, and respiratory system. Make sure to handle solder paste with care, avoiding contact with eyes, skin, and vapor inhalation. Use isopropyl alcohol for solder paste cleanup.

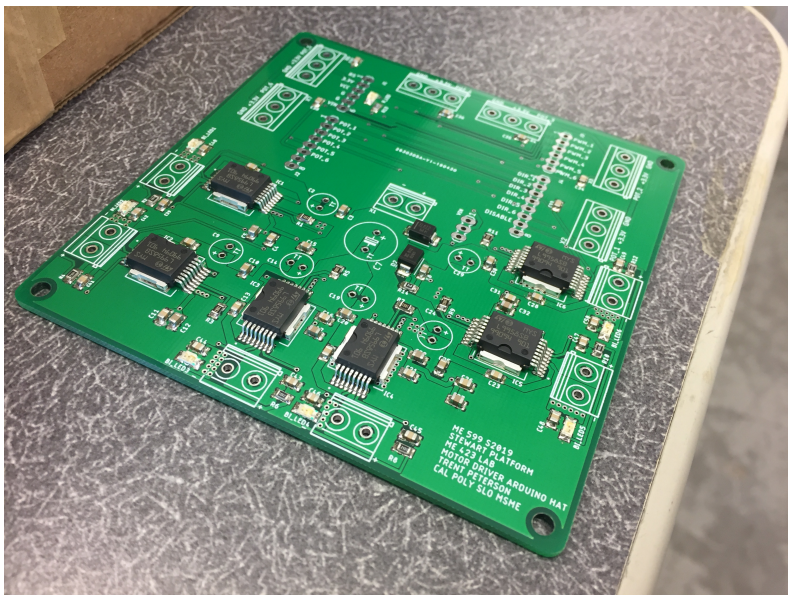


Figure 3.10: HexaMoto Shield, Post Solder Paste Application and Reflow

Next, the capacitors, pin headers, and screw terminals were hand-soldered to the HexaMoto Shield. These are through-hole parts that were not compatible with the stencil. After soldering, the HexaMoto Shield was fully assembled, as shown in Figure 3.11. All traces were checked for shorts and continuity before any further integration.

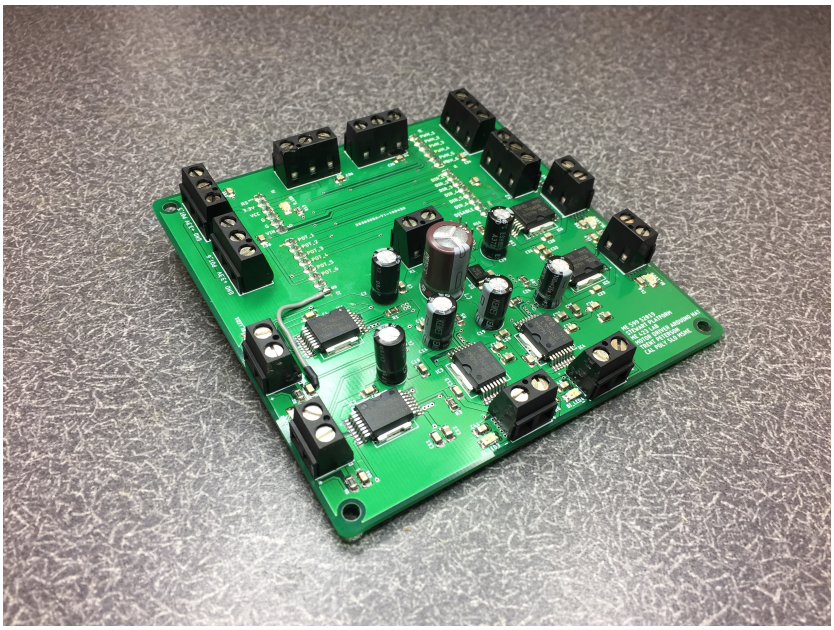


Figure 3.11: HexaMoto Shield, Completed

After the assembly of the HexaMoto Shield was complete, it was fit-tested on the Arduino Due. The pin headers aligned and the shield fit on the Due successfully. During the fit test, it became apparent that the DI pin header was connected to the Arduino TX pin. The TX pin is the transmit pin for serial communication and while it theoretically can be used as a general purpose digital I/O pin, it is not realistic in practice. Therefore, the trace to the TX pin was broken and a jumper wire connected it to an available pin header.

3.4 DC Power Supply

To function properly, each linear actuator must be properly powered. According to Progressive Automations, each linear actuator draws a minimum of 1A during a no-load condition, and can draw up to 5A of current at a maximum-load condition as illustrated in Figure 3.12.

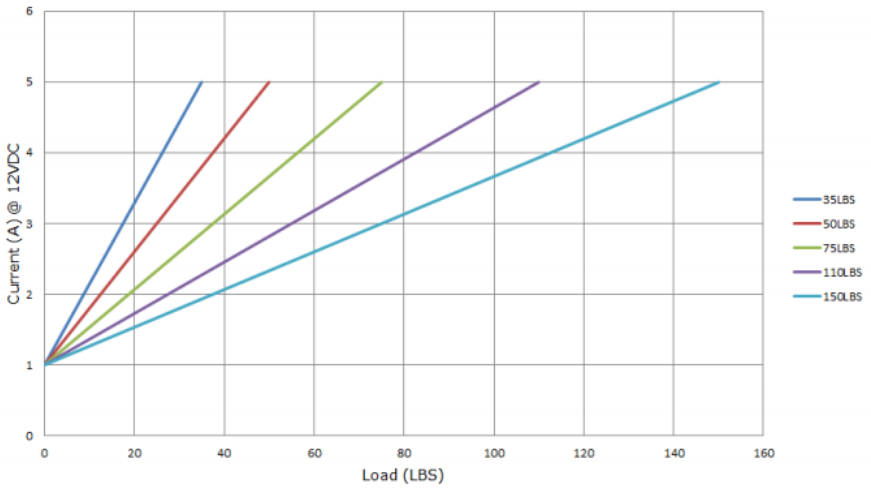


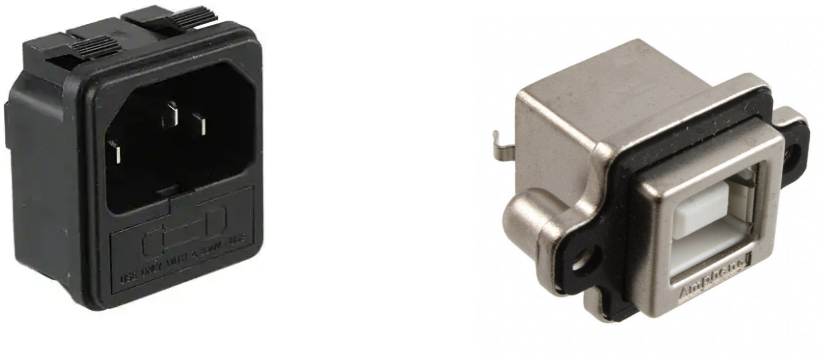
Figure 3.12: PA-14P Current Draw vs Load [19]

A power supply unit was selected to support the maximum current draw of the system. If all linear actuators encountered a simultaneous maximum-load condition, then 30A would be drawn. A 12VDC, 360W power supply was chosen to supply the necessary power. With a maximum output of 30A, the unit can sufficiently power the robot. Additionally, the PSU has a built-in fan that always operates when powered. This provides intake towards the rear of the unit, and exhausts heated air through the front of the unit. Elevated temperatures were not noticed during extended cycle times. The electrical bill of materials may be found in Appendix G.

3.5 Connectors

The power supply unit is connected to AC power through a fused power entry receptacle. The receptacle has a Digikey part number 723W-X2/02 and is shown in Figure 3.13a. The live AC wire is connected in series through a switch before reaching the power supply, allowing the Stewart Platform power to be toggled without having to unplug. Additionally, the power receptacle has a fuse, increasing safety to users and preventing damage to the robot if a short occurs.

The PC host connects to the Arduino with a USB-B cable that plugs into the USB-B receptacle in the electronics enclosure. The receptacle is Digikey part number MUSB-D511-00 and is shown in Figure 3.13b.



(a) Power Receptacle (b) USB-B Receptacle

Figure 3.13: Power and USB Receptacle Connectors Enclosure

The Arduino Due requires a micro-USB connection. A micro-USB cable was spliced into the USB-B receptacle, as shown in Figure 3.14. This provides the intermediate connection between the host PC and the Arduino Due.

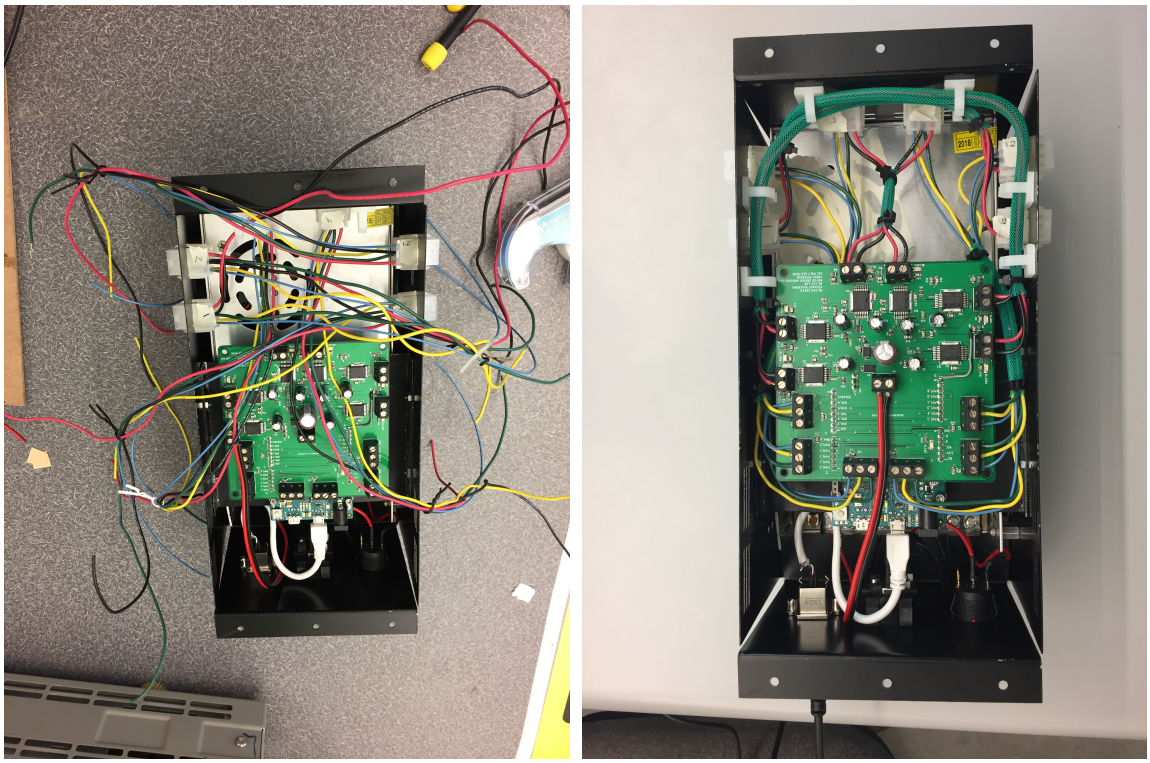
Figure 3.14: MicroUSB to USB-B Connector

3.6 Electrical Assembly and Integration

The following details the progression of the electrical assembly and wiring of the electronic enclosure. First, the PSU, switch, and power receptacle were inserted and wired into the enclosure as shown in Figure 3.15.

Figure 3.15: Painted Electronic Enclosure with PSU and Connectors

Then, the Arduino Due and Hexamoto Shield were installed on the bracket. Once the female Molex connectors were inserted into the enclosure, everything was ready for wiring. The initial "rat's nest" is shown in Figure 3.16a. Zip ties, wire routers and wraps, and heat shrink were used to organize the electronics enclosure as shown in Figure 3.16b. The acrylic base allows the internals of the electronics to be visible but protected. Typically, the electronics is enclosed and hidden away. For professional consumer products this is preferable but for an educational lab, students may enjoy seeing the electronics "under the hood" of the Stewart Platform. Organized wiring can make it easier to compare the system to the schematic for troubleshooting and replacement purposes.



(a) Beginning (b) Completion

Figure 3.16: Enclosure Wiring

Microcontrollers and PCBs can last for many years, especially if steps to properly handle and care are taken. However, replacement of such components is a possibility. A spare HexaMoto was assembled and tested and is ready for replacement. An Arduino Due is an off-the-shelf component that is readily available online. Appendix H details replacing these components of the Stewart Platform Electronics.

Chapter 4

SOFTWARE DESIGN

4.1 Matlab Platform Simulation

During the design stage, a kinematic simulation of the Stewart Platform was created using Matlab. This simulation uses the geometrical properties of the platform, and given a predetermined motion profile, calculates the joint lengths required to perform the move. Joint velocities are found by numerical differentiation of the joint lengths. Platform position, orientation, joint lengths and their respective velocities are plotted over time, with an accompanying 3D animation to visually verify the motion profile. Figure 4.1 shows the Stewart Platform oscillating by 15° about ψ, 2 inches above and below a Pz value of 20 inches, and increase linearly in θ from -15° to +15°.

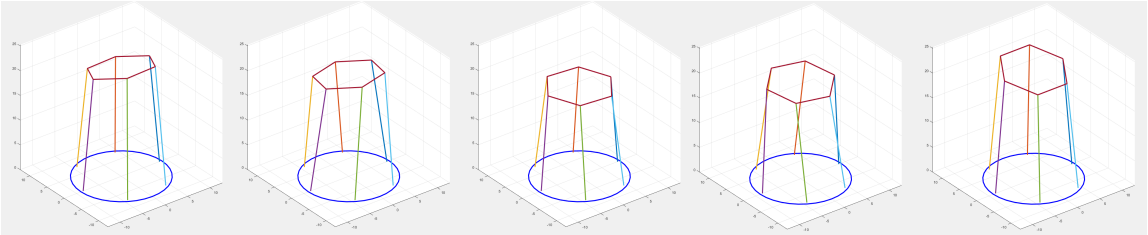


Figure 4.1: Animation Screenshots

4.1.1 Geometrical Properties

The simulation takes in the geometrical properties that pertain to the kinematic chains of the Stewart Platform to accurately calculate joint lengths. The geometry comprises two of the known vectors, defined in Section 1.3.3. The first is the base vector, which needs the distance and angle from the base center to each of the joint

connections. Similarly, the top vector is comprised of the distance and angle of the joint connections on the top plate. The top vectors must be rotated according to the orientation of the plate.

The simulation can be set for a true 6-6 configuration or 6-3 configuration. In a true 6-6 configuration, each joint on the base and top plate are spaced at 60°, which is set automatically in the simulation. In reality, this configuration is unstable, per the discussion in Section 1.3.1. Joints spaced other than 60° degrees apart will still be considered 6-6 configurations but the angles are set manually in the simulation setup. In the 6-3 configuration, there is an angle offset of the joints from the 120° centerline, as discussed in Section 2.5. This modifies Equation 1.3 to become

$$\overline{B_i} = \begin{bmatrix} B*cos((i-1)*60-angle) \\ B*sin((i-1)*60-angle) \\ 0 \\ 1 \end{bmatrix}, \overline{B_{i+1}} = \begin{bmatrix} B*cos((i-1)*60+angle) \\ B*sin((i-1)*60+angle) \\ 0 \\ 1 \end{bmatrix} \tag{4.1}$$

for i = 1, 3, and 5 where angle is 3.47° for this Stewart Platform.

This joint angle difference is accounted for in the Matlab simulation as part of the 6-3 configuration initialization. Ultimately, the angles could be input manually into the script with the angle offset already accounted for. However, the option of 6-3 configuration makes for an improved user experience because the angle offset is already accounted for. The Matlab simulation script can be found in Appendix I.

4.1.2 Motion Profile

The motion profile in the simulation is predetermined and set independently for each degree of freedom. The duration of the motion is set in seconds. The number of steps within that duration can be set depending on how coarse or fine the simulation needs to be. Each of the six degrees of freedom can be set to a constant value, a continuous function, or a piecewise-defined function to achieve the desired motion. The motion profile used to create the animation in Figure 4.1 is included in Figure 4.2 as an example.

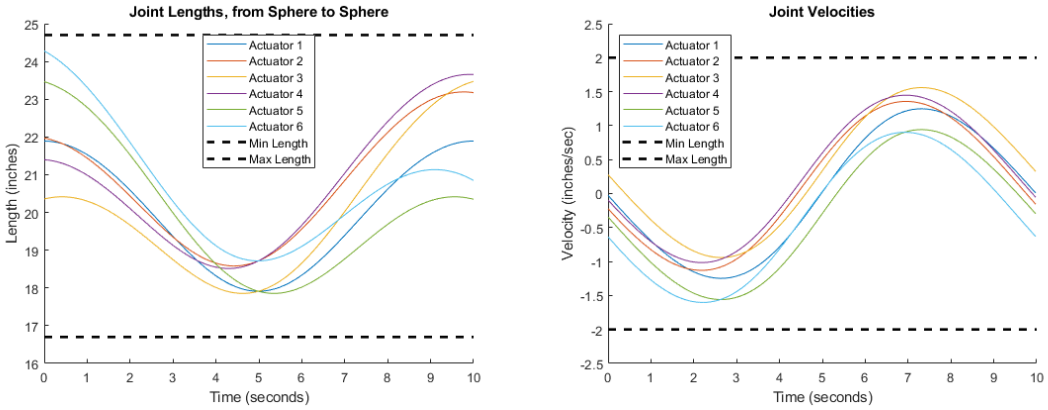
```
%% Motion Profile Creation
% Create function based motion per degree of freedom
% based on the time duration over n steps. All motions are allowed.
% Resulting plots will determine if move profile is valid or not

for m = 1:length(n)
    theta(m)   = -15 + 30 * m/max(n);        %[degrees], relative to x axis
    phi(m)     =  0;                         %[degrees], relative to y axis
    psi(m)     = 15*sind(m/max(n)*360);      %[degrees], relative to z axis
    Px(m)      = 0;                          %[inches], x position
    Py(m)      = 0;                          %[inches], y position
    Pz(m)      = 20 + 2*cosd(m/max(n)*360);  %[inches], z position
end
P = [Px;Py;Pz;zeros(1,length(theta))]; %combine into array
```

Figure 4.2: Motion Profile Creation

It is important to note that the motion profile should be continuous to avoid large spikes in joint lengths and velocities that are not possible to achieve on the physical Stewart Platform. For example, because the simulation uses discreet time steps, a piecewise-defined function may orient the moving platform at $\theta = 20^\circ$ at time $t = 5$ seconds and $\theta = -10^\circ$ at $t = 5.01$ seconds. This would result in a calculated average velocity of $3000^\circ/sec$ about θ across this time interval, which is beyond the capabilities of the Stewart Platform.

4.1.3 Calculation and Animation

The calculations for the joint lengths are completed in the Matlab simulation according to equations 1.11 and 1.12 in Section 1.3.3. In order to achieve the animation, the motion profile above (Figure 4.2) is linearly divided into hundreds of discreet steps over a specified time span. The inverse kinematics are calculated for each step giving a resolution equal to the motion profile time divided by the total number of steps. For example, the following plots show the length and velocities of linear actuators (Figure 4.3) for the given motion profile. This motion profile was specified to take 10 seconds and was calculated with 400 steps, for a resolution of 25 ms/step.



(a) Joint Lengths (b) Joint Velocitiess

Figure 4.3: Matlab Simulation Plots for Joint Values

The script also outputs plots for the platform position and orientation and the velocity of each axis with respect to time. A plot of the angles of rotation about each axis is shown in Figure 4.4. The plot matches the values of θ, ϕ, and ψ specified in the motion profile.

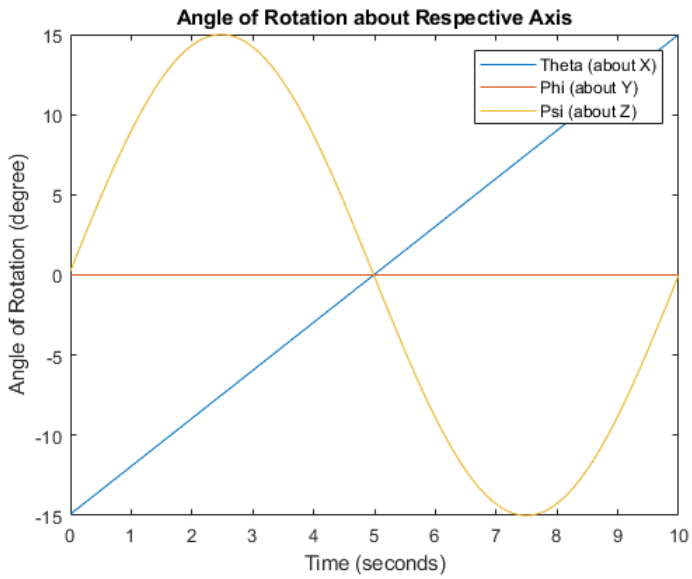


Figure 4.4: Matlab Simulation Plots for Orientation Values

Table 4.1 is provided as an example of these calculations for a Stewart Platform in a 6-6 configuration with the following values: $a = 9$ in, $b = 7$ in, $\theta = 0°$, $\phi = -30°$, $\psi = 45°$, $P_x = 0$, $P_y = 0$, $P_z = 22$ in. A 3D representation of these values is shown in Figure 4.5. These calculations are done for every step of the animation.

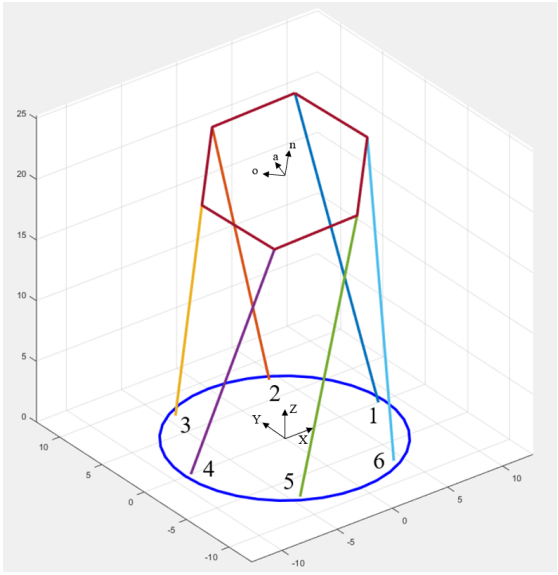


Figure 4.5: 6-6 Configuration Model for Inverse Kinematics Solution

Table 4.1: Inverse Kinematics Solution for 6-6 Configuration

Loop	1	2	3	4	5	6
a_x	9.00	4.50	-4.50	-9.00	-4.50	4.50
a_y	0	7.79	7.79	0	-7.79	-7.79
a_z	0	0	0	0	0	0
b_x	7.00	3.50	-3.50	-7.00	-3.50	3.50
b_y	0	6.06	6.06	0	-6.06	-6.06
b_z	0	0	0	0	0	0
b_x rotated	4.29	-2.14	-6.43	-4.29	2.14	6.43
b_y rotated	4.29	6.43	2.14	-4.29	-6.43	-2.14
b_z rotated	3.50	1.75	-1.75	-3.50	-1.75	1.75
d_x	-4.71	-6.64	-1.93	4.71	6.64	1.93
d_y	4.29	-1.36	-5.65	-4.29	1.36	5.65
d_z	25.50	23.75	20.25	18.50	20.25	23.75
d	26.28	24.70	21.11	19.57	21.35	24.49

4.2 Arduino Sketch

The Arduino code must fulfill 3 primary requirements: Manage I/O, communicate over a serial connection, and run a control loop for the actuators. A state transition diagram, shown in Figure 4.6 was created to lay out the design of the code to achieve these design requirements.

First, the robot needs to establish serial communication with the host PC. Once this connection is made, then the robot will perform an initial calibration and homing procedure. Once successfully calibrated and homed, the code will wait for a commanded position for each actuator. When received, the Arduino Due will run the control loop in the sketch and move the linear actuators.

For terminology reference, a sketch is the name for a program written for an Arduino. The sketch is comprised of two special functions, setup() and loop(). Upon the start

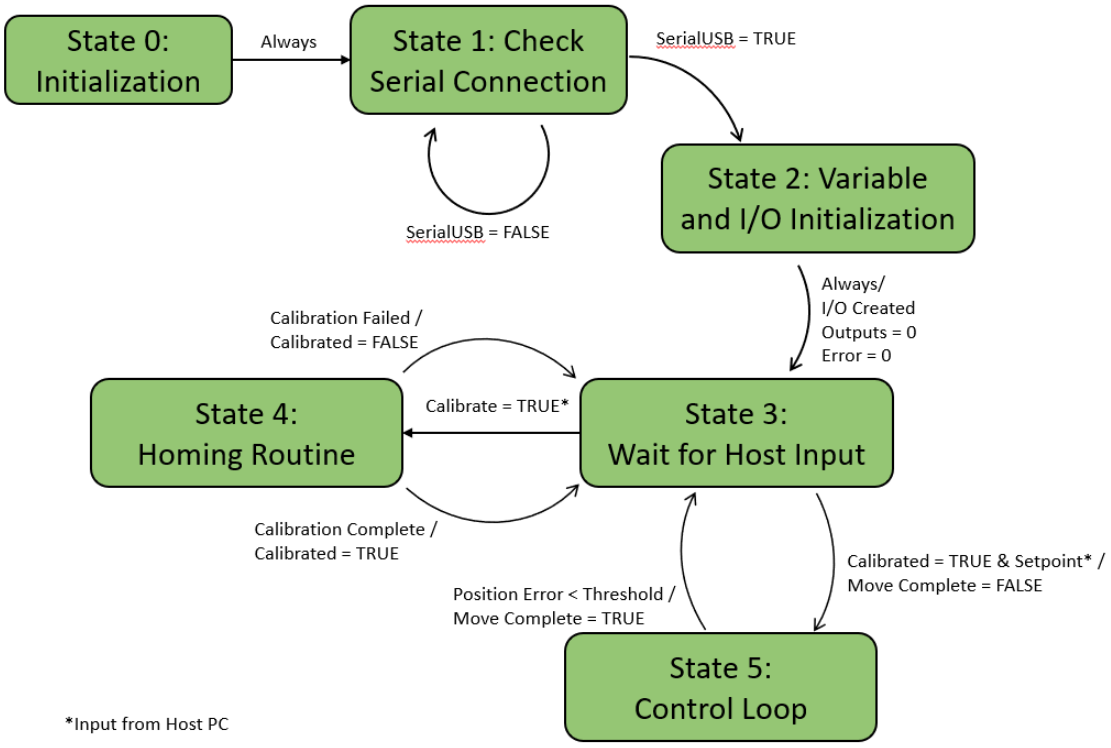


Figure 4.6: Arduino State Transition Diagram

of the sketch, setup() runs once and after completion, loop() is repeatedly called. Custom functions can be declared in the sketch and called within loop().

4.2.1 Arduino Sketch Source Credit

When researching Progressive Automations and their linear actuators, it was discovered that they sponsored a similar Stewart Platform project. The sponsored team is from the University of British Columbia and worked on the project as part of an Engineering Physics course. The team documented the project, including their code for the Stewart Platform. Finding this code, which was released as open-source, presented two options; one option is to utilize the code, given proper credit, and adapt it to the project; the alternative is to develop the code from scratch.

The code satisfied the design requirement set above. Rather than developing brand-new code that would just result in near-identical function, the code was taken, with proper credit, verified, and adapted to this particular Stewart Platform, and may be found an Appendix J.

4.2.2 Setup and Initialization

The setup waits for the serial connection between the host PC and the NativeUSB port before starting. Once connected, the direction, PWM, and potentiometer pins are declared, and all outputs are set low. The control loop error is cleared to prevent premature or unexpected actuation. This setup is followed by a calibration and homing routine.

4.2.3 Calibration and Homing

After initialization, the robot first performs a homing and calibration procedure. The procedure records the maximum and minimum potentiometer readings by fully extending then retracting the platform. The Arduino Due ADC is 10 bits, so the range of potentiometer values in software is 0-1023. When fully retracted, the potentiometer readings on the actuators are between 80 and 90 counts; the readings are between 950 and 960 counts when fully extended. The calibration routine scales these raw readings to the full 10-bit range of 0-1023. This eliminates potentiometer variations between each actuator, enabling them to have an identical control loop. The calibration routine also detects if the motor power is off and will return an error. When the calibration routine is successful and the robot is fully retracted, it waits at its home position, ready to receive position commands.

4.3 Matlab GUI

A graphical user interface (GUI) was created using Matlab App Designer to allow users to easily connect to the robot and operate it. The user interface is shown in Figure 4.7, and the source code may be found in Appendix K. This section will cover the design choices and software flow of the app and its intended usage.

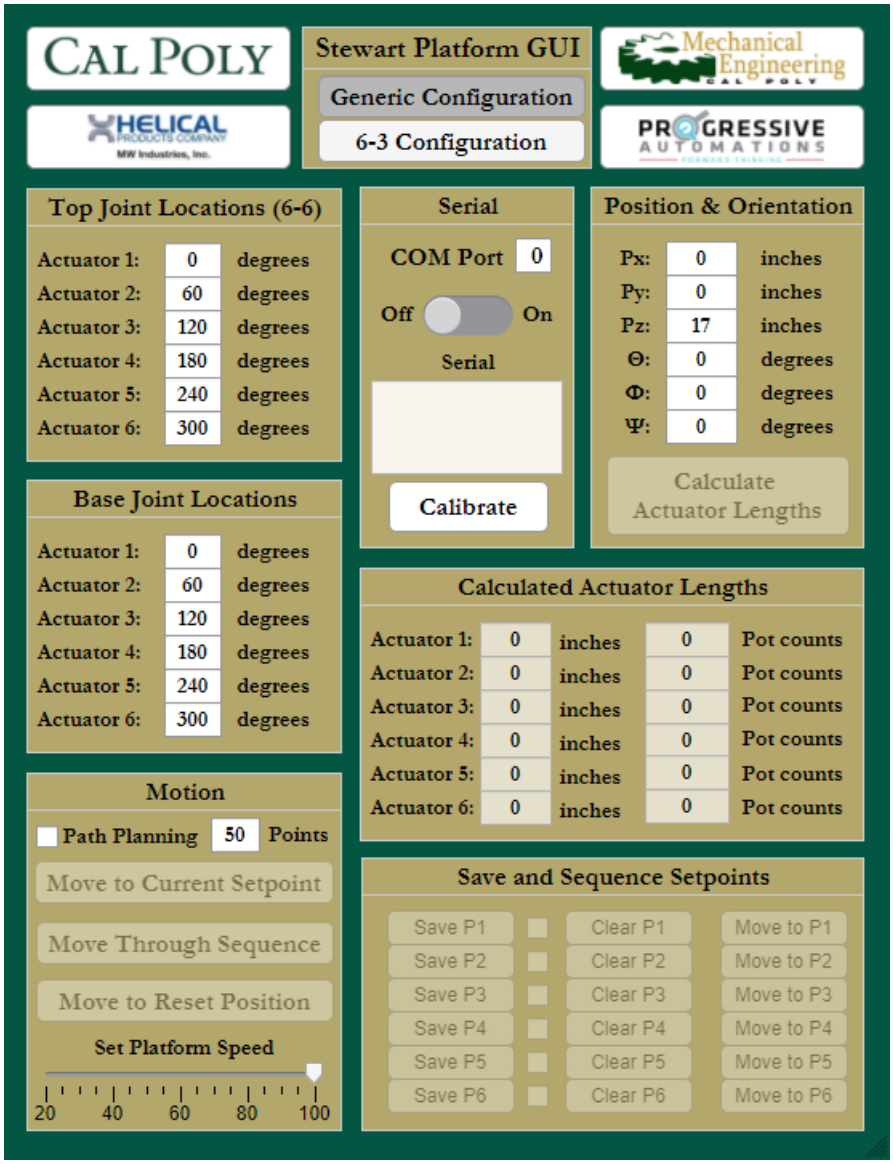


Figure 4.7: Matlab GUI for Stewart Platform

4.3.1 Serial Connection

Matlab first connects to the Arduino using serial communication over the COM port specified by the GUI. Determining the correct COM port is detailed in the Lab Manual. Once the correct COM port is entered, and the 'Calibrate' button is clicked, the Stewart Platform begins its calibration procedure. When the calibration completes, the 'Calibrate' button changes to 'Calibrated'. Only when the serial connection is successful and calibration is complete can the joint lengths be calculated. This portion of the GUI is shown in Figure 4.8.

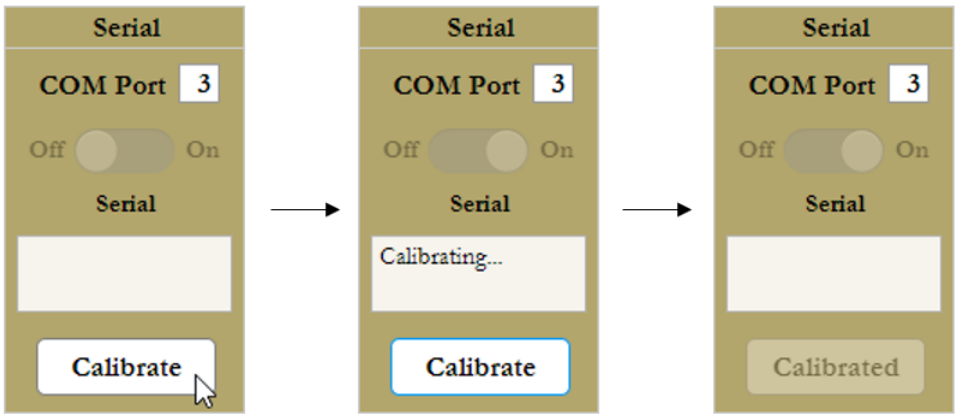


Figure 4.8: Serial Connection and Calibration

4.3.2 Platform Configuration

The next step is to set up the geometrical configuration of the Stewart Platform. The first option to select is whether the Stewart Platform is currently in a generic configuration or a 6-3 configuration. If the 6-3 selection is configured, the location of the top joints is automatically set to 0°, 120°, and 240°. This cannot be set by the user because the top acrylic plate only allows for the steel balls to be placed together at these locations. The generic configuration allows the user to set the locations of

the joints on the top plate. Both options allow the user to input the joint positions on the base plate. This portion of the GUI is shown in Figure 4.9.

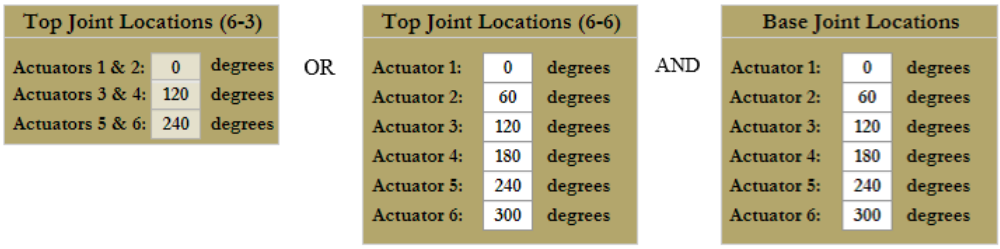


Figure 4.9: Top and Bottom Joint Locations

4.3.3 Linear Actuator Position Calculation

The actuator lengths can now be calculated with the correct joint locations. The position and orientation of the platform is set by the user as desired. There are no software limits on the position and orientation values. There are dozens of possible configurations of the Stewart Platform, and each one has a different range of motion. Students will investigate the range of motion for two of these configurations. This portion of the GUI is shown in Figure 4.10.

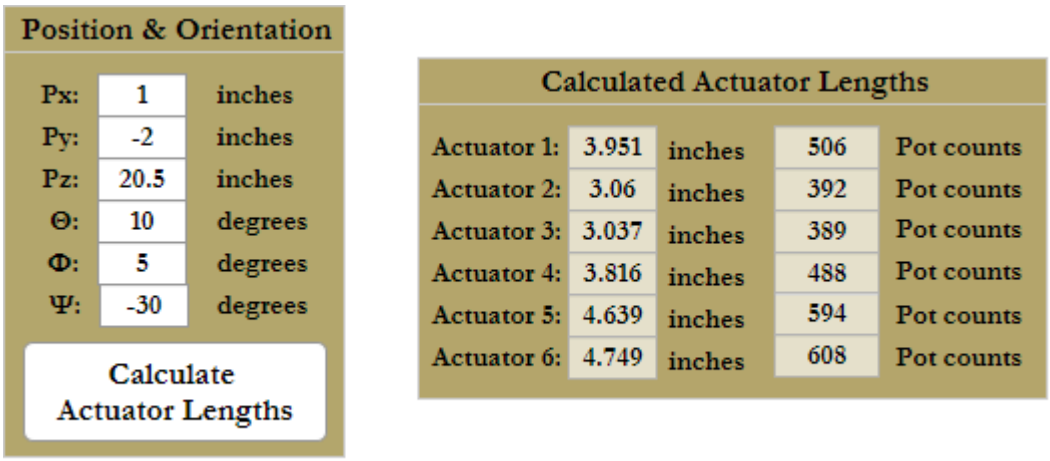


Figure 4.10: Position and Orientation Input

Once the position and orientation of the Stewart Platform are given, the joint lengths can be calculated. It is important to note that if the joint positions are not set correctly, the length calculation will not be correct. At best, the robot will actuate to an incorrect position. Worse cases include the robot actuating to an unstable condition and collapsing. The results of the length calculation will be presented in two forms. The first form is in units of inches. If the length of any actuator is calculated to exceed its stroke to attain a certain position, an error will result. The robot will not be allowed to move to this location or be saved.

4.3.4 Saving and Sequencing Points

The user has the option to save the current calculated position under the 'Save and Sequence Setpoints' panel shown in Figure 4.11. It can be saved in any of the six options. If a saved setpoint is no longer desired, it can be cleared, and saved setpoints can be overwritten. An option is provided to move to a setpoint after it has been saved.

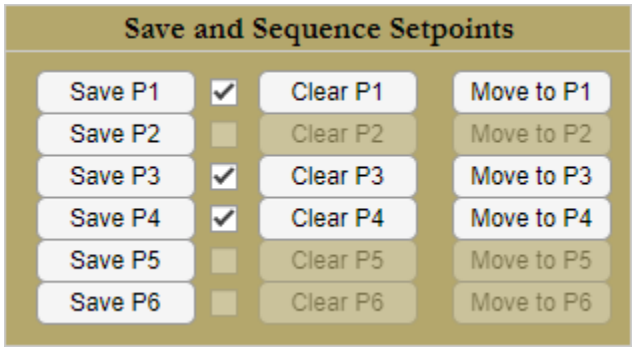


Figure 4.11: Save and Sequence Panel

4.3.5 Motion

This GUI panel, shown in Figure 4.12 contains the settings that alter the robot motion and the buttons that command robot motion. By checking the 'Enable Path Planning' button, the controller will divide the move into smaller moves specified in the 'Points' text box. This path is normalized in each degree of freedom via interpolation.

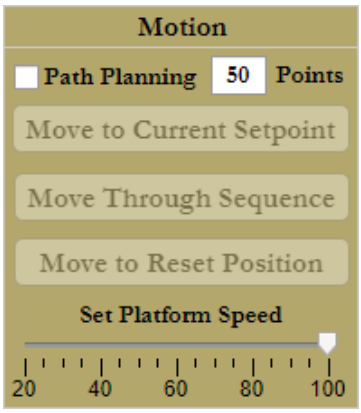


Figure 4.12: Motion Control Panel

The 'Set Platform Speed' slider changes the Stewart Platform's speed by scaling the PWM signal to the motor drivers. When at 100%, no scaling occurs and up to the maximum voltage (12VDC) can be sent to the actuators, as calculated by the control loop. If set to 50%, all PWM signals calculated by the control loop are scaled by 50% before being output. The 'Move to Current Setpoint' button moves the Stewart Platform to the position and orientation specified in the 'Position and Orientation' panel by sending the actuator lengths from the 'Calculated Actuator Lengths' panel to the controller. The 'Move Through Sequence' button will command the Stewart Platform through up to six consecutive moves, from P1 to P6. The sequence will skip any empty inputs, which may occur if a saved setpoint was cleared after being saved. In other words, the only setpoints that will be in the sequence will have a check next to them.

Chapter 5

TESTING AND FUTURE WORK

5.1 Testing

Testing was performed on the linear actuators to ensure that they would achieve the length commanded. Table 5.1 contains the data taken for each actuator as they were commanded from 0 to 8 inches in 1 inch increments. The measurements were taken with a tape measure with a resolution of 1/16" (0.063"), and measured values interpolated to the nearest 1/32" (0.031"). The data shows that the actuators were accurate to within 1/16 of an inch across the entire stroke.

Table 5.1: Commanded and Actual Actuator Lengths

Input	Actuator 1	Actuator 2	Actuator 3	Actuator 4	Actuator 5	Actuator 6
0	0.00	0.00	0.00	0.00	0.00	0.00
1	1.00	1.00	1.00	1.00	1.00	1.00
2	2.00	2.00	2.00	2.00	2.00	2.00
3	3.00	3.00	3.00	3.00	3.00	3.00
4	4.00	4.06	4.03	4.03	4.03	4.03
5	5.00	5.06	5.06	5.03	5.03	5.03
6	6.03	6.06	6.06	6.03	6.03	6.03
7	7.06	7.06	7.06	7.03	7.03	7.03
8	8.06	8.06	8.06	8.03	8.03	8.03

The simulation script and its animation confirmed that the inverse kinematics equations are correct and working properly. Tests were performed on the Stewart Platform to ensure that the inverse kinematics work on the actual system, and that values specified in the GUI were achieved within tolerance.

The position of the Stewart Platform was measured first. The height (Z) was simply measured from the top surface of the base platform to the top surface of the moving platform. This measurement was taken with a tape measure with markings every 1/16 inches. The moving platform position in the x-axis and y-axis was measured by attaching a weighted pendulum to the top platform, which would make contact with a ruled grid placed on the base platform below. The grid had markings every tenth of an inch. Table 5.2 contains the nominal values input to the GUI and the actual position achieved by the Stewart Platform.

Table 5.2: Stewart Platform Positional Test Measurements

X $[in]$		Y $[in]$		Z $[in]$	
Input	Measured	Input	Measured	Input	Measured
-4	-4.25	-4	-4.20	18	17.94
-3	-3.20	-3	-3.10	19	18.97
-2	-2.10	-2	-2.05	20	20.00
-1	-1.05	-1	-1.00	21	21.00
0	0.00	0	0.00	22	22.00
1	1.05	1	1.00	23	23.00
2	2.10	2	2.00	24	24.03
3	3.15	3	3.10		
4	4.20	4	4.20		

The platform was most accurate at the center of its positional range of motion. The height of the platform was very accurate and had a maximum error of 1/16" at the minimum height and 1/32" at the maximum height. The error along the x-axis and y-axis increased fairly linearly, by approximately 0.05 inches per inch.

The angles of the Stewart Platform were also measured using an inclinometer app on a smartphone. A 30-60-90 degree drafting triangle and a 45-90 degree drafting triangle were used to verify the app readings at 30°, 45°, and 60°. The moving platform was leveled with respect to gravity and then rotated by θ about the x-axis and by ϕ about

the y-axis; each axis was measured separately. The measurement resolution of the inclinometer was 1°. Examples of these measurements are shown in Figure 5.1.

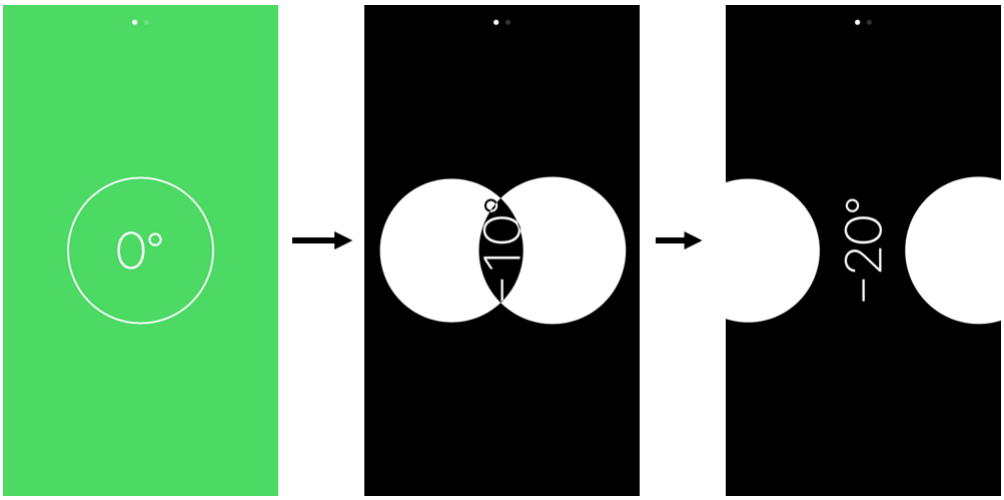


Figure 5.1: Inclinometer App Example Readings

A protractor with a resolution of 1° was used to measure ψ. The protractor was zeroed to the moving platform, and then held in place independently as the platform was rotated about the z-axis. Table 5.3 documents the input and measured values for θ, ϕ, and ψ.

Table 5.3: Stewart Platform Orientation Test Measurements

θ $[degrees]$		ϕ $[degrees]$		ψ $[degrees]$	
Input	Measured	Input	Measured	Input	Measured
-30	-30	-30	-31	-60	-61
-25	-24	-25	-26	-50	-51
-20	-19	-20	-20	-40	-41
-15	-14	-15	-15	-30	-30
-10	-9	-10	-10	-20	-20
5	-4	-5	-5	-10	-10
0	0	0	0	0	0
5	4	5	4	10	10
10	10	10	9	20	19
15	15	15	14		
20	20	20	19		
25	25	25	24		
30	30	30	29		

The orientation error of the platform was constant, and did not significantly increase across the range of motion, unlike the positional error. The platform was generally 1° less in magnitude in the $-\theta$ domain and $+\phi$ domain. The platform also slightly over-rotated at the edge of the $-\phi$ and $-\psi$ domains. The primary causes to these sources of error can be attributed to geometry inaccuracies and residual control-loop error. The nominal design values for joint-to-joint lengths, which include the spherical joints and actuator geometry, and platform bolt-circle diameters were used in the equations. These nominal values were checked on the actual platform for accuracy, but there was variation up to 1/16 of an inch from kinematic chain to kinematic chain. The control-loop error originates from the controls design. When the actuators are within a few potentiometer counts, the effective voltage sent to the motors is very low, resulting in no movement but an audible whine. If the actuator is within this threshold, the PWM output from the Arduino is set to zero, eliminating the whine. This solution introduces an error in the actuator lengths, up to 1/16" as demonstrated in Table 5.1, which will result in the small error in platform position and orientation.

5.2 Conclusion

A Stewart Platform was successfully designed, assembled, and implemented as part of a lab experiment. It covers many facets of robots at an introductory level, including inverse kinematics, simulation, assembly and disassembly, and motion implementation. Using the platform does not require a steep learning curve and performs satisfactorily for student experimentation. The mechanical design allows for easy reconfiguration and uses the magnetic joints as the mode of failure, protecting the rest of the system components. While the Stewart Platform is an overall success, it is open to further improvements and future work.

5.3 Improvements

5.3.1 HexaMoto Shield

Overall, the Hexamoto Shield design proved successful but it can be improved. As discussed in 3.3, the disable pin was connected to the TX pin, which is used in serial communication. This was fixed on the current PCB revision by cutting the trace and rerouting to another pin. Revision 2 of the Hexamoto Shield schematics F fixes this design flaw. It is labeled 'A' in Figure 5.2.

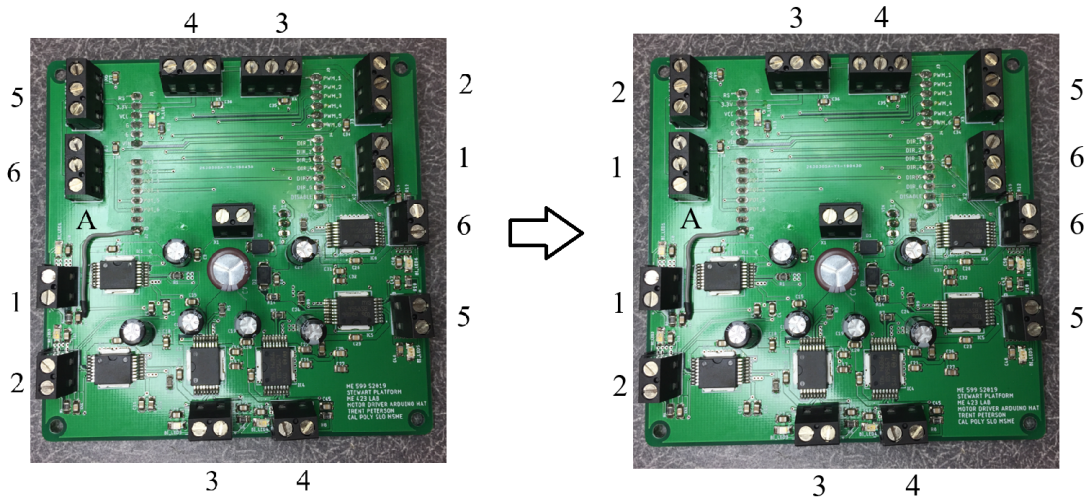


Figure 5.2: HexaMoto Shield Improvements

The placement of screw terminals could also be changed to improve ease of assembly. The labels 1-6 correspond to the linear actuator. The potentiometer and power terminals are separated across the board. Pairing these terminals by mirroring across the horizontal would make more sense, as only actuators 3 and 4 would be separated across the board. During board design, pairing the power and potentiometer terminals was attempted, but it was ultimately decided to keep the motor drivers together.

5.3.2 Acrylic Plates

Using acrylic plates for the Stewart Platform was appropriate when considering cost and available manufacturing resources. In retrospect, it may have been justifiable to use aluminum for the base and platform instead. The aluminum, while more expensive, would be stronger and stiffer than the acrylic. Thicknesses between 1/8" to 1/4" are recommended. Threaded inserts should be used in the top plate because the steel spheres will be assembled and disassembled frequently when the Stewart

Platform is reconfigured per the lab manual. Aluminum threads can wear down under these conditions.

The current angle spacing on the base and moving platforms is 20°. A spacing of 30° may be more appropriate; 20° spacing offered a few too many configuration options, and 30° spacing is more intuitive when using sine, cosine, and the x and y-axes. Additionally, the angles increased in the clockwise direction from a top view. This should be changed to counter-clockwise in order to follow the right-hand rule for Cartesian coordinates. Currently, CW results in +Z in the downwards direction.

To achieve this, the material would have to be sourced and purchased. If using only Cal Poly resources is desired then the aluminum should be cut with a water jet. Once cut, the threads in the top platform would need to be tapped, ideally for threaded inserts.

5.4 Future Work

A method of jogging the robot would be a helpful addition to the Stewart Platform. This could be accomplished using buttons mapped to increasing or decreasing the value of a degree of freedom. A joystick, a phone, or a 3D mouse (Figure 5.3) would be particularly suited to controlling a Stewart Platform.

This device is able to sense in all 6 degrees of freedom, which matches the Stewart Platform. Any user input to the 3D mouse would be matched by the moving platform. The mouse can also sense magnitude. A light push or twist of the mouse could jog the robot slowly, and larger twists etc. would result in faster jogging.

Incorporation of this device can be accomplished through either the Arduino Due or the host PC. The 3D mouse can be plugged into the Native USB port on the Arduino

Figure 5.3: 3Dconnexion 3D Mouse [2]

Due, which can act as a USB host for USB peripherals. Software would have to be developed in order to parse the data from the mouse. The host PC could utilize Matlab's Space Mouse functionality in Simulink as another means of communicating with the mouse. Either device would have to receive and interpret the 3D mouse data and use it in a velocity-control loop. This device would be especially useful to decrease teaching time of a Stewart Platform. For example, if a laser was attached to the moving platform and had to be positioned and aligned with a receiver, the 3D mouse could quickly jog the robot to achieve this. Currently, the alternative is to guess a numerical position and orientation in the GUI, move the platform, and increment the numerical input until satisfied.

Chapter 6

LAB MANUAL

6.1 Lab Manual

A lab manual was developed in conjunction with the Stewart Platform. The experiment investigates inverse kinematics, reconfiguration, range-of-motion, and use of the Stewart Platform as a flight simulator. The simulation script from Section 4.1 will be used by students to verify their inverse kinematics. Next, the students will reassemble the robot into two configurations, and determine the range-of-motion of each. Finally, students will plan and run a movement sequence to simulate basic flight motion. The full lab manual can be found in Appendix L.

6.2 Learning Objectives

The usage of the Stewart Platform in the lab will complement the parallel robots material covered in lecture. The simulation provides students with the opportunity to apply their understanding of inverse kinematics with low-risk. The accompanying animation gives a glimpse into a Stewart Platform's motion, and can show how catastrophe can result if the inverse kinematics are incorrect. Re-configuring the robot and testing its range-of-motion highlights the various geometries for an SPS robot and the resulting impact on its capabilities and performance. In addition, this allows students to become familiar with the robot and the GUI. The flight simulation helps answer the often asked question, "What is the point in learning this?" by using the Stewart Platform in this common commercial application.

BIBLIOGRAPHY

[1] AliExpress. *KD418 Universal ball and socket joint*, 2019. Accessed 8 October 2019.

[2] AmericanXplorer13. Space navigator. Web, March 2007. Accessed 5 February 2020.

[3] Arduino. *ARDUINO DUE*, 2019. Accessed 22 June 2019.

[4] K. J. Ayala. *The 8051 microcontroller*. Cengage Learning, 2004.

[5] P. Cruz, R. Ferreira, and J. Silva Sequeira. Kinematic modeling of stewart-gough platforms. *ICINCO*, 2005.

[6] B. Dasgupta and T. Mruthyunjaya. The stewart platform manipulator: a review. *Mechanism and Machine Theory*, 35(1):15–40, December 1998.

[7] Digi-Key. Molex 0022284060. Web, August 2019. Accessed 18 August 2019.

[8] Digi-Key. Molex 0398800303. Web, August 2019. Accessed 18 August 2019.

[9] Digi-Key. SunLED XZMDKVG55W-4. Web, August 2019. Accessed 18 August 2019.

[10] Firgelli Automations Team. Linear Actuators 101 - Everything you need to know about Linear Actuators. Web, November 2018. Accessed 13 June 2019.

[11] IBS Magnet. *Magnetic ball joints*, 2019. Accessed 13 June 2019.

[12] K. Lancaster. AMiBA Radio Telescope. Web, June 2006. Accessed 10 January 2020.

[13] Matthewbarry. Eric gough's tire testing machine. Web, March 2016. Accessed 10 January 2020.

[14] McMaster-Carr. *Ball Joint Rod Ends Datasheet*, 2019. Accessed 13 June 2019.

[15] J. Merlet. *Parallel Robots*. Springer Netherlands, P.O. Box 17, 3300 AA Dordrecht, The Netherlands, 2006.

[16] S. B. Niku. *Introduction to Robotics: Analysis, Control, Applications*. John Wiley and Sons, 2nd edition, 2011.

[17] S. B. Niku. *Introduction to Robotics: Analysis, Control, Applications*. John Wiley and Sons, 3rd edition, 2020.

[18] Progressive Automations. Multimoto arduino shield. Web, August 2019. Accessed 18 August 2019.

[19] Progressive Automations. *PA-14P Datasheet*, 2019. Accessed 13 June 2019.

[20] Progressive Automations. Resources. Web, August 2019. Accessed 19 August 2019.

[21] Robot Power. Multimoto product information. Web, August 2019. Accessed 18 August 2019.

[22] STMicroelectronics. *L9958*, December 2013. Accessed 20 August 2019.

[23] B. Vanderborght. Compliant robots. Web, January 2012. Accessed 10 October 2019.

[24] S. Vincent and J. Bridges. Positioning with air electropneumatic-positioning systems bring extreme accuracy to high-speed automation. Web, April 2015. Accessed 13 June 2019.

APPENDICES

Appendix A

PA-14P DATASHEET

PROGRESSIVE AUTOMATIONS

FORWARD THINKING

PA-14P
Data Sheet

Contents

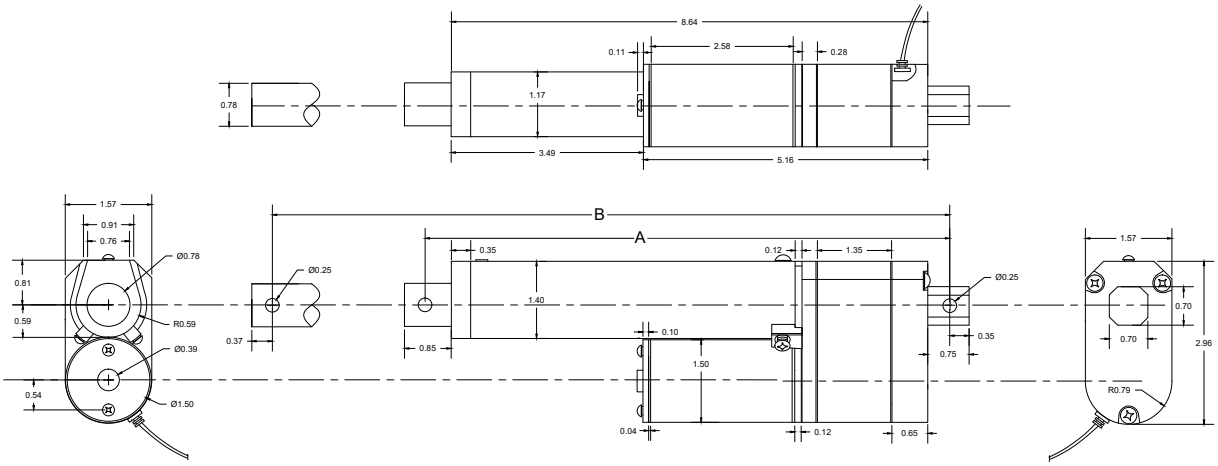
Specifications

Load (LBS)		No Load Current (A)				Full Load Current (A)				Speed (inch/sec)	
Dynamic	Static	12VDC	24VDC	36VDC	48VDC	12VDC	24VDC	36VDC	48VDC	No Load	Full Load
35	75	1.0	0.5	0.3	0.3	5.0	2.5	1.7	1.3	2.00	1.38
50	100	1.0	0.5	0.3	0.3	5.0	2.5	1.7	1.3	1.14	0.83
75	150	1.0	0.5	0.3	0.3	5.0	2.5	1.7	1.3	0.95	0.70
110	220	1.0	0.5	0.3	0.3	5.0	2.5	1.7	1.3	0.79	0.59
150	300	1.0	0.5	0.3	0.3	5.0	2.5	1.7	1.3	0.37	0.28

Stroke	1" to 40"
Limit Switch	Internal - Non-Adjustable
Limit Switch Feedback	Customizable
Screw Type	ACME Screw
Motor Type	Brushed or Brushless DC Motor
Connector Type	See Page 5
Wire Length	40" (customizable)
Housing Material	6062 Aluminum Alloy
Rod Material	Aluminum Alloy/Stainless Steel (customizable)
Gear Material	Polyformaldehyde (35 lbs only)/Powder Metallurgy Steel Alloy
Color (Shaft)	Silver
Color (Motor End)	Silver
Noise	<45dB
Duty Cycle	25% (5 minutes on, 15 minutes off)
Operational Temperature	-25ºC to 65ºC (-13ºF to 149ºF)
Protection Class	IP54 (IP65 customizable)
Feedback Options	Potentiometer (see page 5)
Certifications	CE/RoHS
Mounting Brackets	See Page 6
Mounting Ends	Customizable

Dimensions

(Dimensions in inches)



Hole to Hole

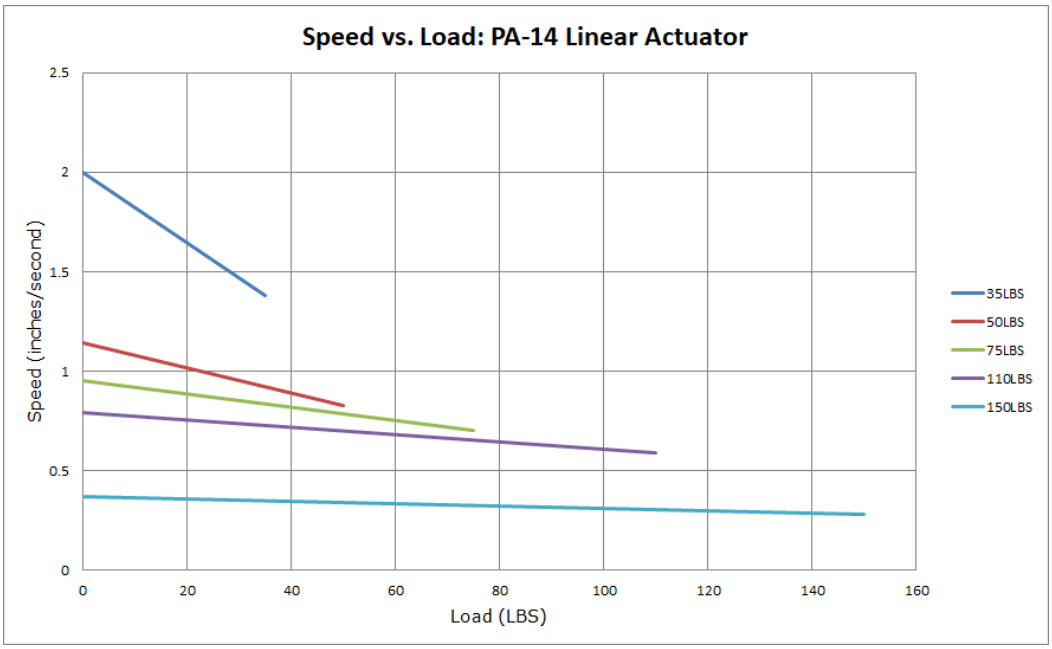
	Stroke	1	2	3	4	6	8	9	10	12	14	16	18	20	22	24	30	40
PA-14P	**A**	6.51	7.51	8.51	9.51	11.51	13.51	14.51	15.51	17.51	19.51	21.51	23.51	25.51	27.51	29.51	35.51	45.51
	B	7.51	9.51	11.51	13.51	17.51	21.51	23.51	25.51	29.51	33.51	37.51	41.51	45.51	49.51	53.51	66.61	85.51

For Stroke Length

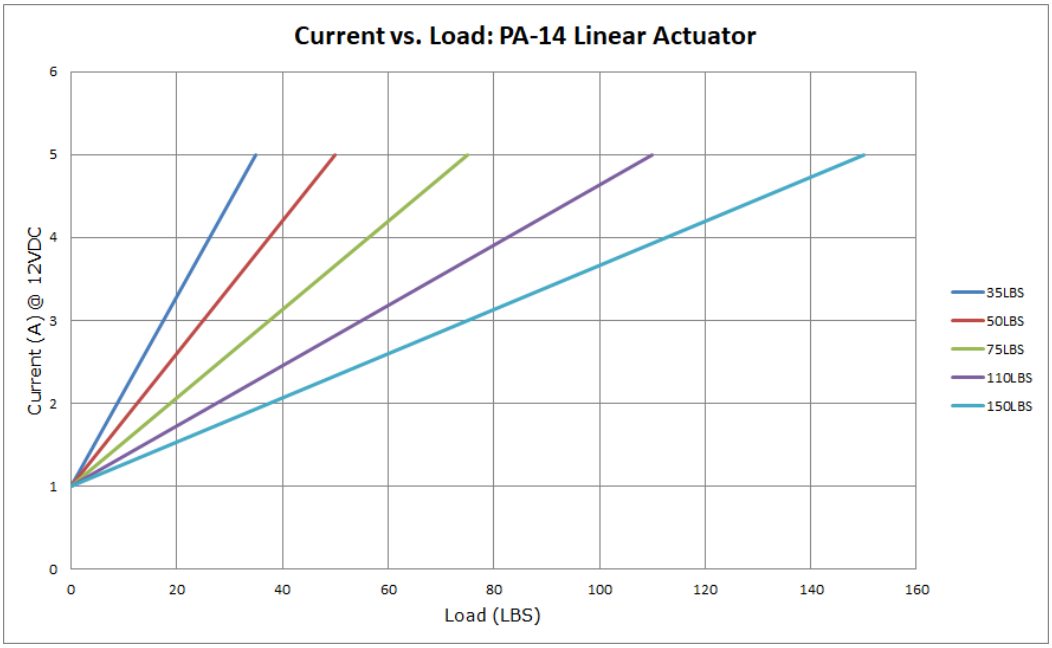
A = Stroke Length + 5.51"

B = Stroke Length x 2 + 5.51"

Speed vs Load

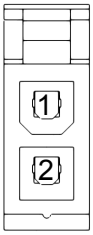


Current vs Load



Connectors & Feedback

2-Pin Connector (Standard)

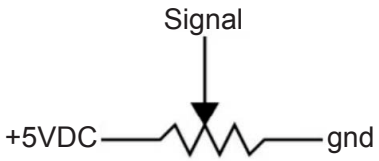


Motor	
1	2
M-	M+

Component	Part Name	Part Number	Mating Part Number
Housing	Molex Mini Fit Jr 2-Pin Receptacle	39-01-2025	39-01-3029/ 39-01-2026

Potentiometer Specifications

*For Stroke Length up to 40"



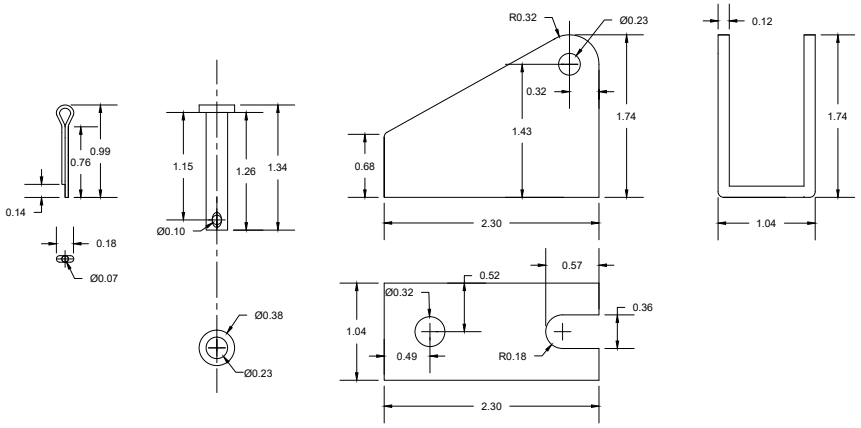
Resistance*	Number of Turns	Tolerance
0-10kΩ	10	+/- 5%

*Actual resistance value may vary within the 0-10kΩ range based on stroke length

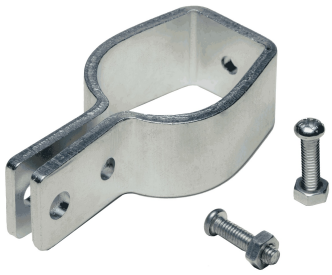
Mounting Brackets

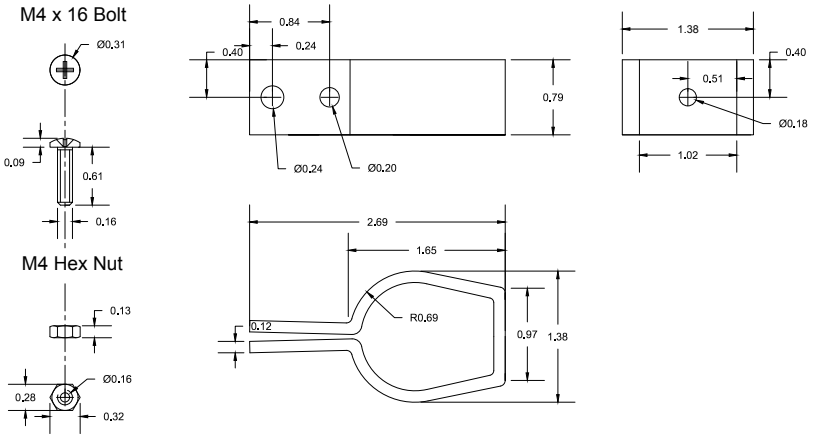
(Dimensions in inches)

BRK-14

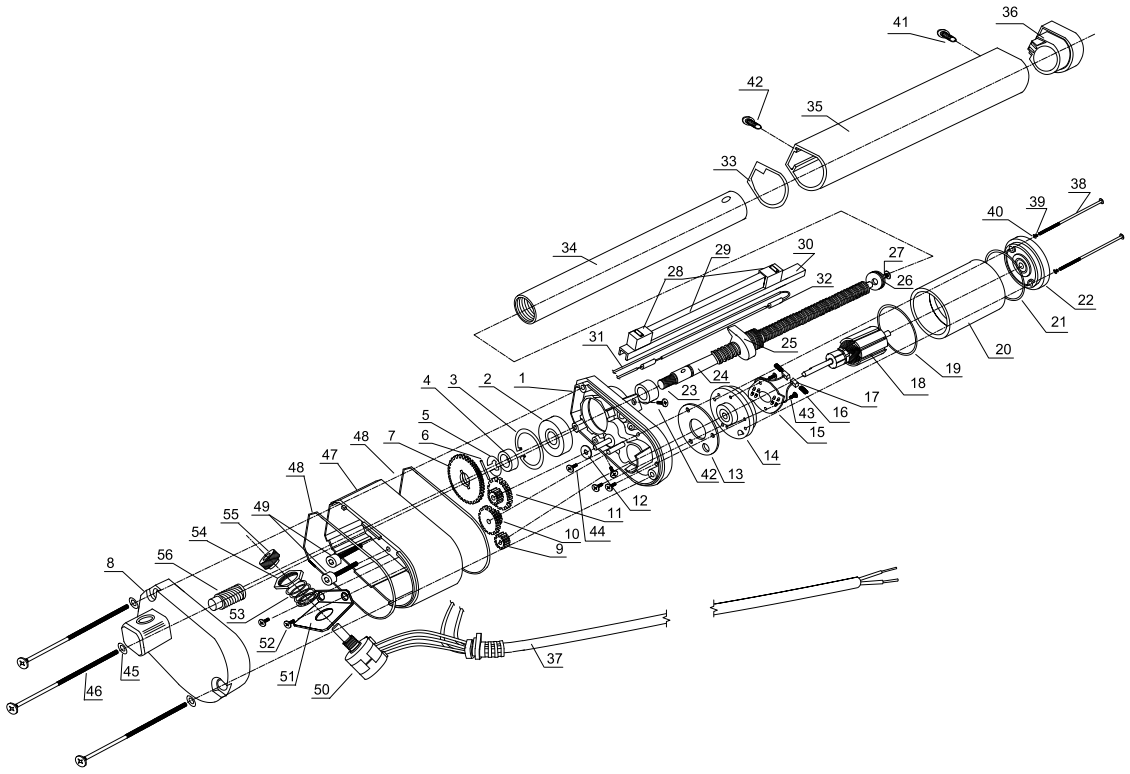


BRK-03





Internal Components



Internal Descriptions

Item	Description	Qty
1	Actuator base	1
2	Shaft Bearing	1
3	Shaft Bearing Lock	1
4	Shaft Base Spacer	1
5	Shaft Base Spacer Lock	1
6	Shaft Gear Wheel Holder	1
7	Shaft Gear Wheel	1
8	Base Cover with Mounting Support	1
9	Electric Motor Gear Wheel	1
10	Small Intermediate Gear Wheel	1
11	Medium Intermediate Gear Wheel	1
12	Teflon Washer	1
13	Electric Motor Base Washer	1
14	Electric Motor Base	1
15	Brush Holder PCB	1
16	Electric Motor Brush	2
17	Electric Motor Brush Spring	2
18	Electric Motor Rotor	1
19	Motor Enclosure Bottom Washer	1
20	Electric Motor Encloser with Stator	1
21	Motor Enclosure Top Washer	1
22	Electric Motor Cap with Rotor Bearing	1
23	Shaft Spacer	1
24	Treaded Shaft Drive / Lead Screw	1
25	Shaft Base with Limit Switches Arm	1
26	Shaft Drive End Support	1
27	Shaft Drive End Support Screw	1
28	Limit Switch	2

Item	Description	Qty
29	Limit Switches Spacer	1
30	Limit Switches Base	1
31	Limit Switches Wiring	1
32	Diode	2
33	Shaft Encloser Bottom Washer	1
34	Shaft with Mounting Hole	1
35	Shaft Encloser	1
36	Shaft Enclosure Top Cap	1
37	Power Cable	1
38	Motor Enclosure Screw	2
39	Motor Screw Spring Washer	2
40	Motor Screw Washer	2
41	Shaft Eclosure Top Cap Screw	1
42	Shaft Enclosure Base Screw	3
43	Brush Holder PCB Screw	2
44	Motor Base Screw	3
45	Base Cover Washer	3
46	Base Cover Screw	3
47	Base Extension	1
48	Base Extension gasket	2
49	Base Extension Screw	2
50	Potentiometer	1
51	Potentiometer Bracket	1
52	Potentiometer Bracket Screw	2
53	Potentiometer Washer	3
54	Potentiometer Nut	1
55	Potentiometer Gear	1
56	Shaft Gear	1

Appendix B

WAC20-4MM-4MM DATASHEET

HELICAL PRODUCTS COMPANY
MW Industries, Inc.

Search:

Navigation

PRODUCT INFORMATION

Part Description:	**WAC20-4mm-4mm**
Attachment Method: Clamp	Integral

VIEW CAD MODEL

Material Information

Finish: Anodize	Clear
Material: Aluminum Alloy	7075-T6

REQUEST QUOTE

Dimensional Information

Outside Diameter:	20mm
Length:	28mm

Standard Bore Diameters

Major Bore Diameter:	4mm
Minor Bore Diameter:	4mm

Attachment Data

A1 Type:	Cap Screw
A1 Screw Material:	Alloy Steel

Torque

Momentary Dynamic:	1.3 Nm
Non-Reversing:	0.65Nm
Reversing:	0.33Nm
Rated Speed:	10000 RPM

Misalignment

Angular Misalignment:	5 deg
Parallel Misalignment:	.25mm
Axial Motion:	0.25mm

A1 Screw Size:	M3x.5
A1 Seating Torque (numeric):	2.0Nm
A1 Hex Wrench Size:	2.5mm
A1 Center Line (Metric):	3.8mm
A2 Type:	Cap Screw
A2 Screw Material:	Alloy Steel
A2 Screw Size:	M3x.5
A2 Seating Torque (numeric):	2.0Nm
A2 Hex Wrench:	2.5mm
A2 Center Line (Metric):	3.8mm

Additional Specifications

Torsional Rate (deg/Nm): degree/Nm	2.7
Maximum Operating Temperature:	93C

1. Dynamic torque ratings are momentary values. For non-reversing applications, divide by 2. Divide by 4 for reversing applications. Should the torque ratings be marginal for your application, contact us for analysis.
2. Manufacturing dimensional tolerances unless otherwise specified are:

Inch:	Metric:
fraction +1/64	x + .5 mm
x.xx +.010	x.x + .25 mm

3. Inertia is based on smallest standard bore diameter.
4. Keyways available in MC Series and W Series with 40 & 50mm OD

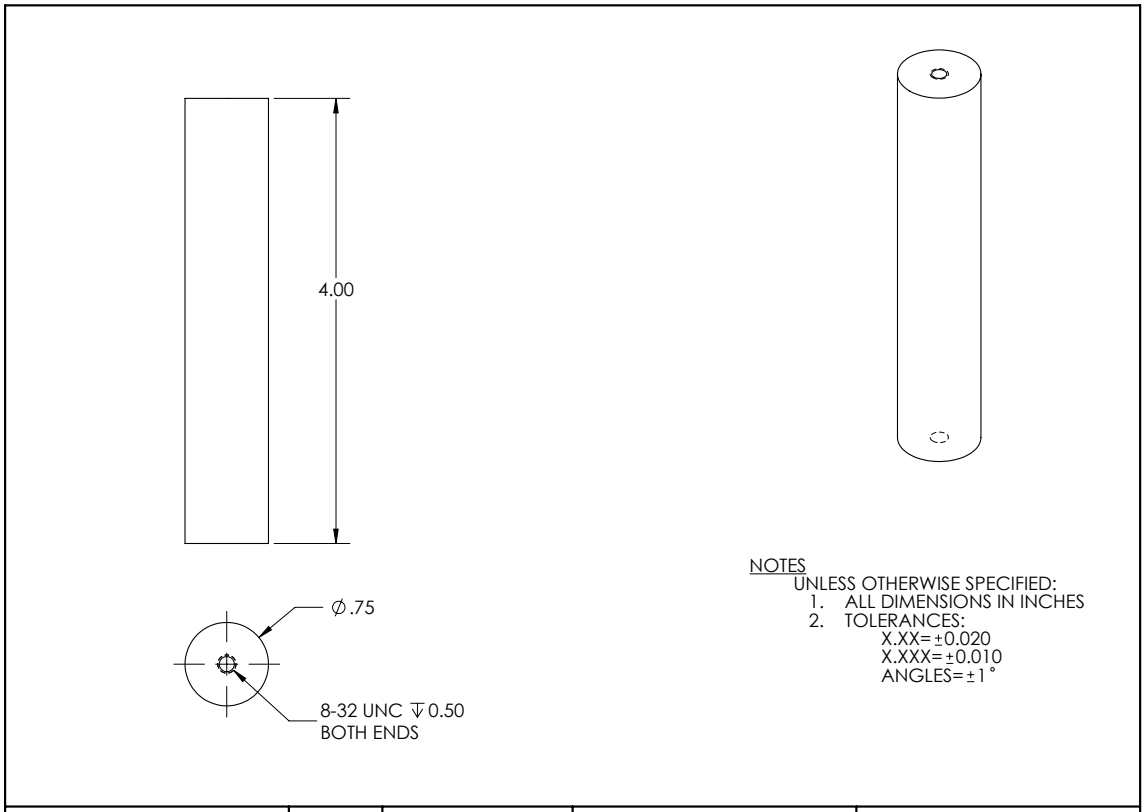
PRINT

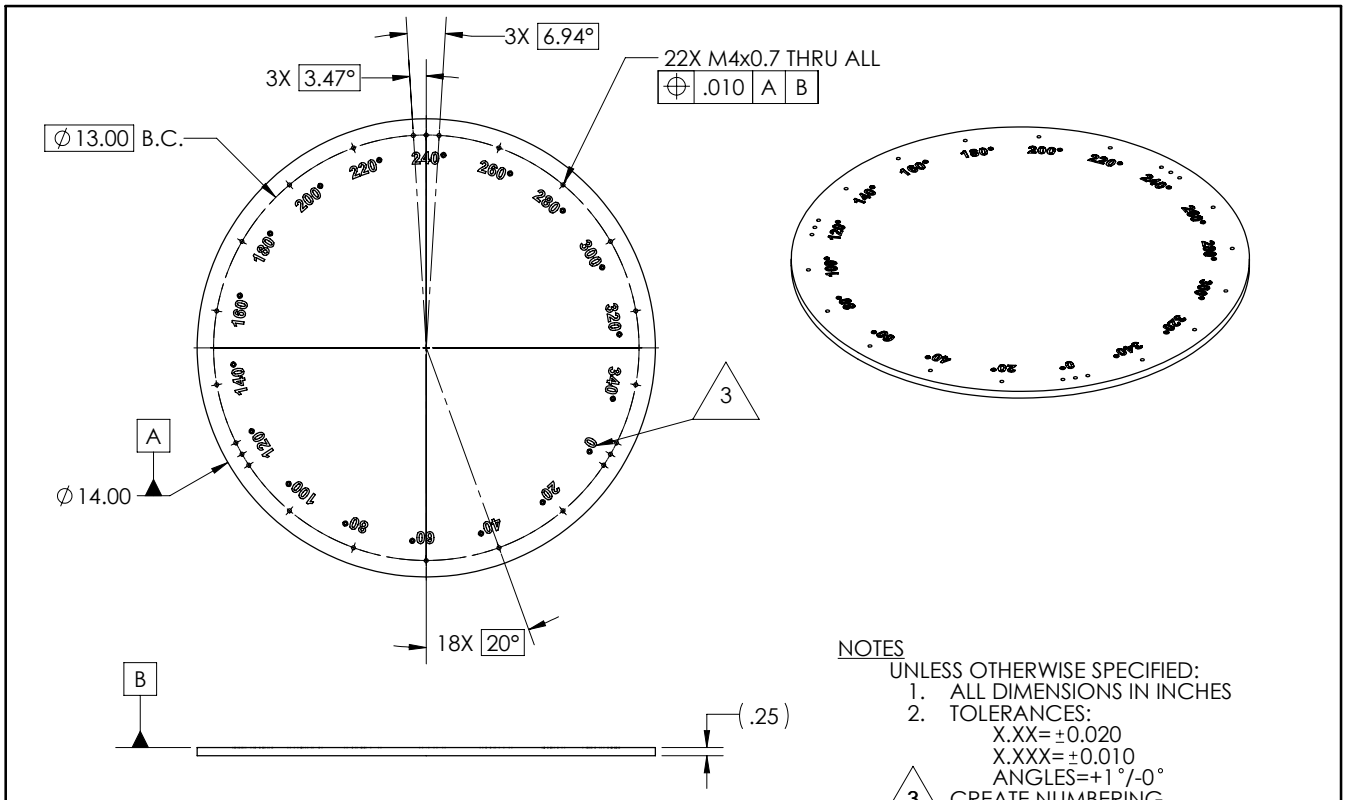
Products
Engineering
About Us
Request A Sample
Sales
News
Applications

Flexible Couplings
Beam Couplings

Appendix C

MECHANICAL DRAWINGS



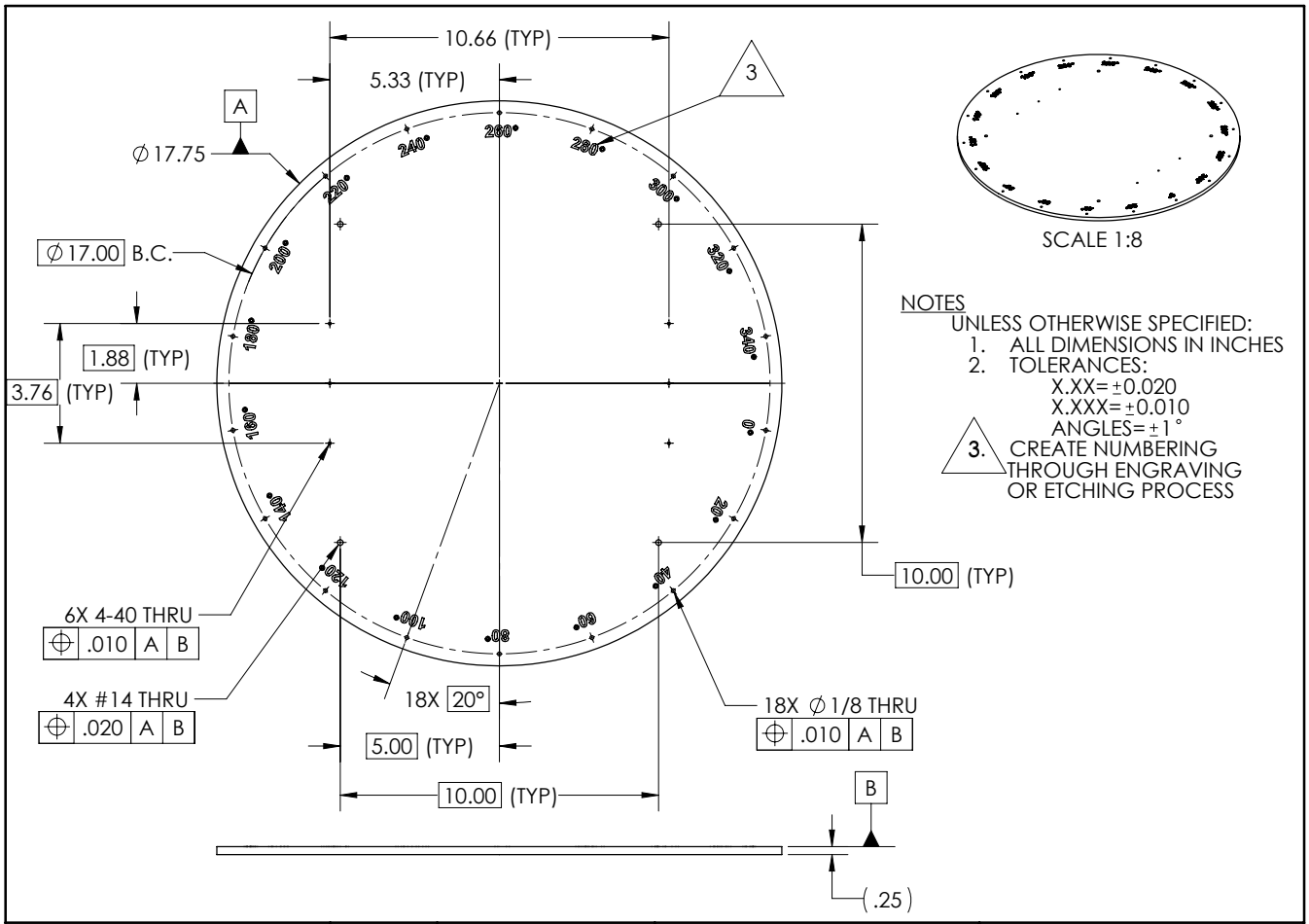


NOTES

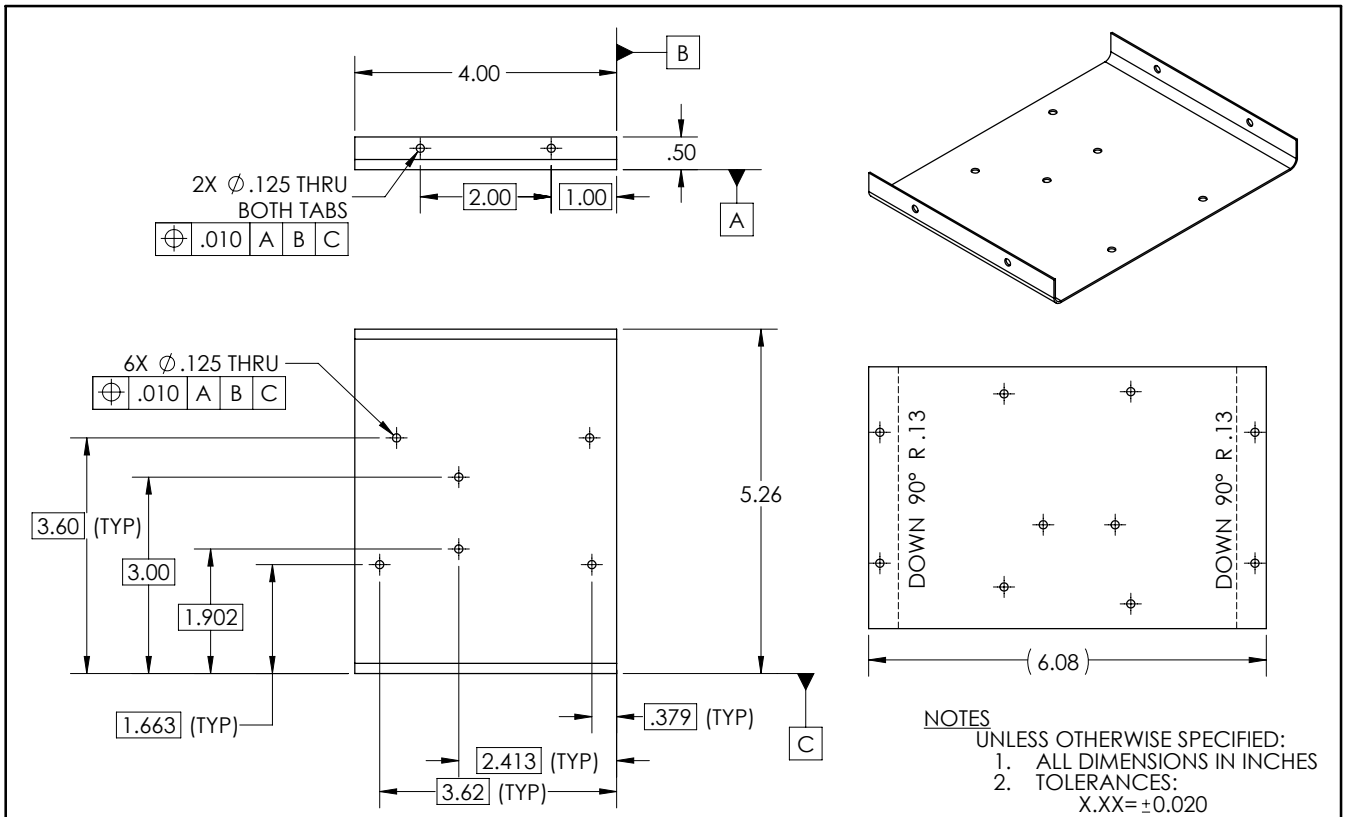
UNLESS OTHERWISE SPECIFIED:

1. ALL DIMENSIONS IN INCHES
2. TOLERANCES:
   X.XX= ±0.020
   X.XXX= ±0.010
   ANGLES=+1°/-0°
3. CREATE NUMBERING THROUGH ENGRAVING OR ETCHING PROCESS

Cal Poly Mechanical Engineering	Section:01	Mat'l: Acrylic, 0.25 THK	Title: TOP PLATE		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #: 02	Nxt Asb:SGP-001	Date:6/28/2019	Scale: 1:4	Chkd. By: ME STAFF



Cal Poly Mechanical Engineering	Section:01	Mat'l: Acrylic, 0.25 THK	Title: BASE PLATE		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #: 01	Nxt Asb:SGP-001	Date:6/28/2019	Scale: 1:4	Chkd. By: ME STAFF

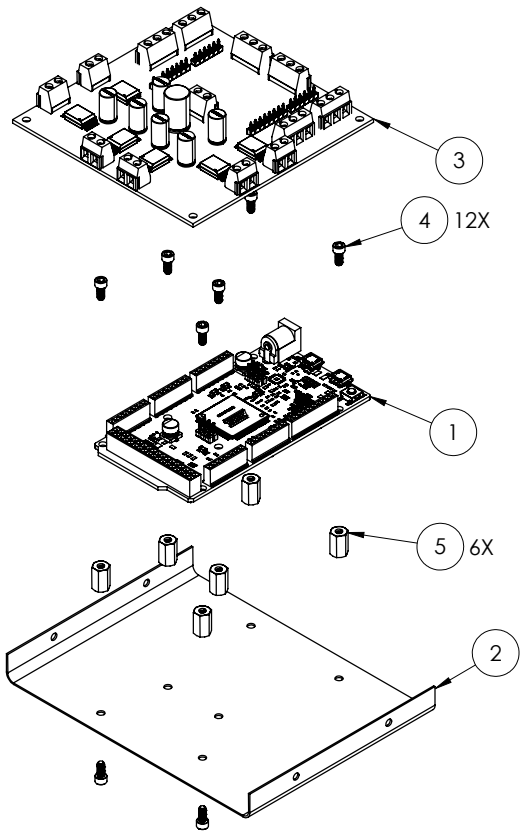


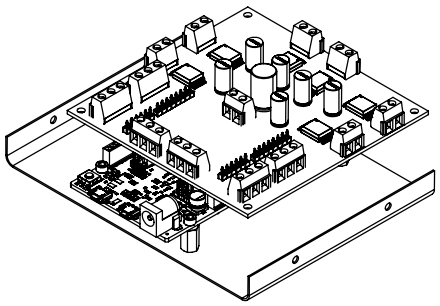
NOTES

UNLESS OTHERWISE SPECIFIED:

1. ALL DIMENSIONS IN INCHES
2. TOLERANCES:
   X.XX= ±0.020
   X.XXX= ±0.010
   ANGLES= ±1°
3. BEND RADIUS TO NOT EXCEED R=0.15

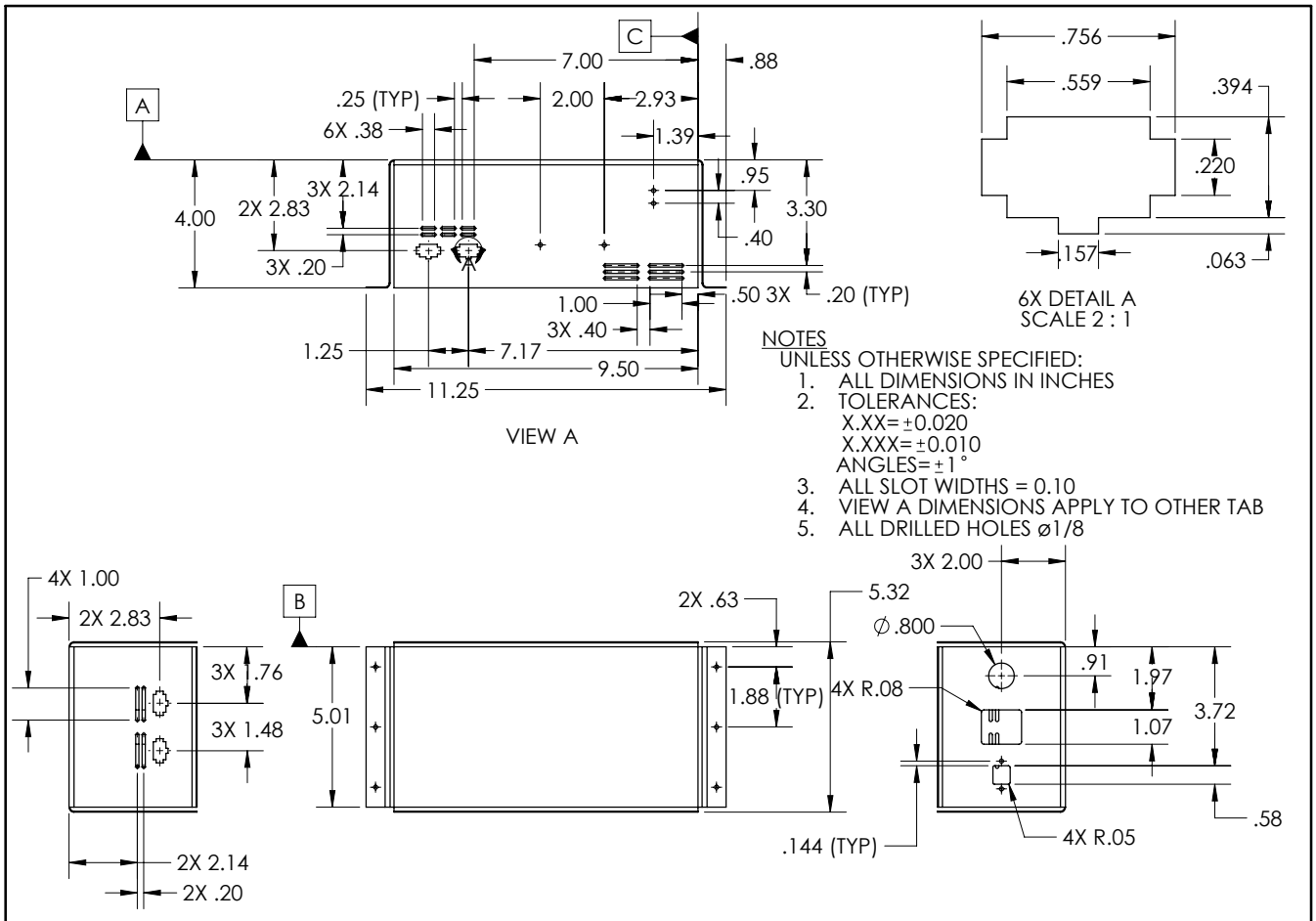
Cal Poly Mechanical Engineering	Section:01	Mat'l: AL, #14-#18 GA	Title: MOUNT, ARDUINO DUE		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #: 04	Nxt Asb:ELCE-001	Date:6/28/2019	Scale: 1:2	Chkd. By: ME STAFF



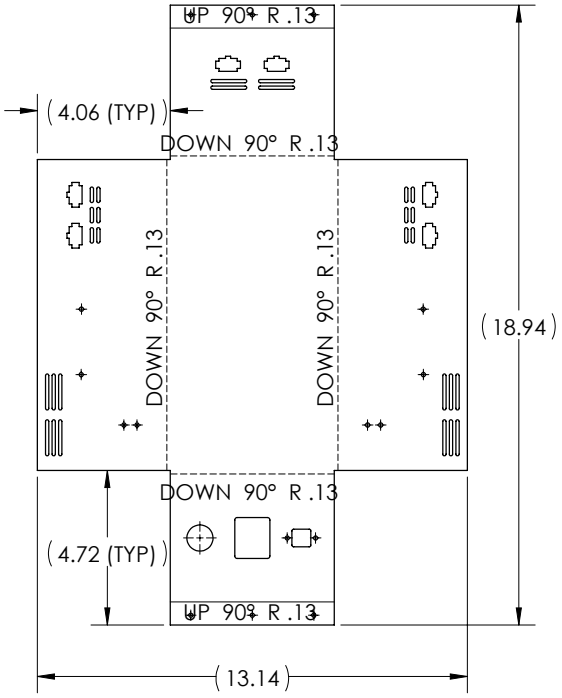


ITEM NO.	PART NUMBER	Default/ QTY.
1	ARDUINO DUE	1
2	MOUNT, ARDUINO	1
3	HexaMoto Shield	1
4	92196A106 (McMaster)	12
5	91780A427 (McMaster)	6

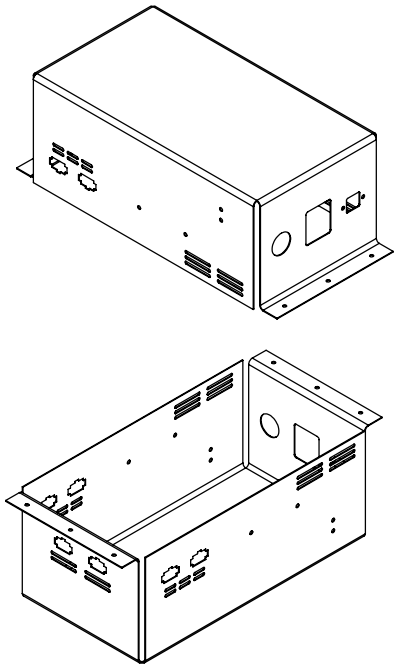
Cal Poly Mechanical Engineering	Section:01	Mat'l: N/A	Title: Arduino, Subassy		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #:01	Nxt Asb: ELEC,ENCL	Date:6/29/2019	Scale: 1:2	Chkd. By: ME STAFF



Cal Poly Mechanical Engineering	Section:01	Mat'l: AL, #14-#18 GA	Title: ENCLOSURE, ELECTRONICS		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #: 04	Nxt Asb:ELCE-001	Date:6/28/2019	Scale: 1:4	Chkd. By: ME STAFF



FLAT PATTERN (REFERENCE)

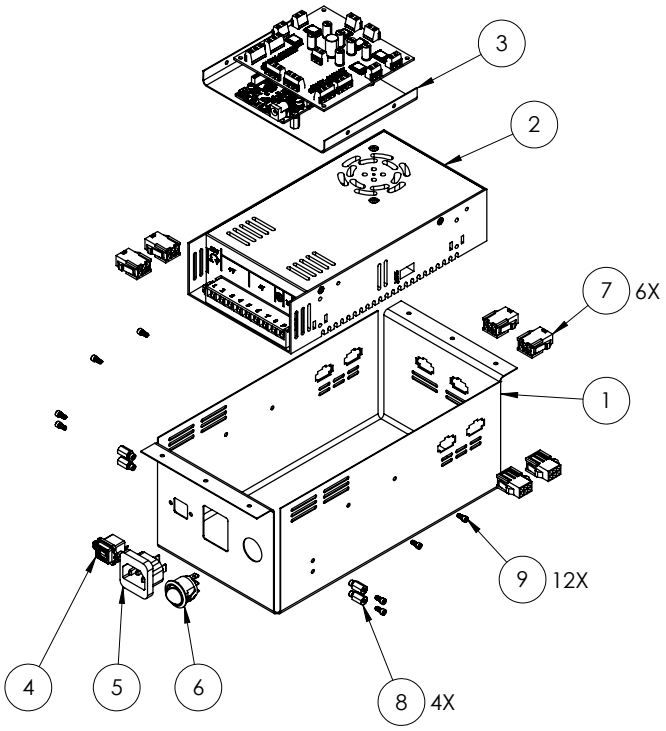


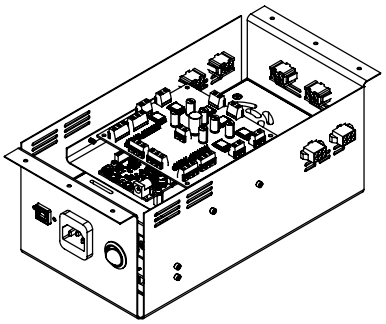
NOTES

UNLESS OTHERWISE SPECIFIED:

1. ALL DIMENSIONS IN INCHES
2. TOLERANCES:
   X.XX= ±0.020
   X.XXX= ±0.010
   ANGLES= ±1°
3. BEND RADIUS TO NOT EXCEED R=0.15

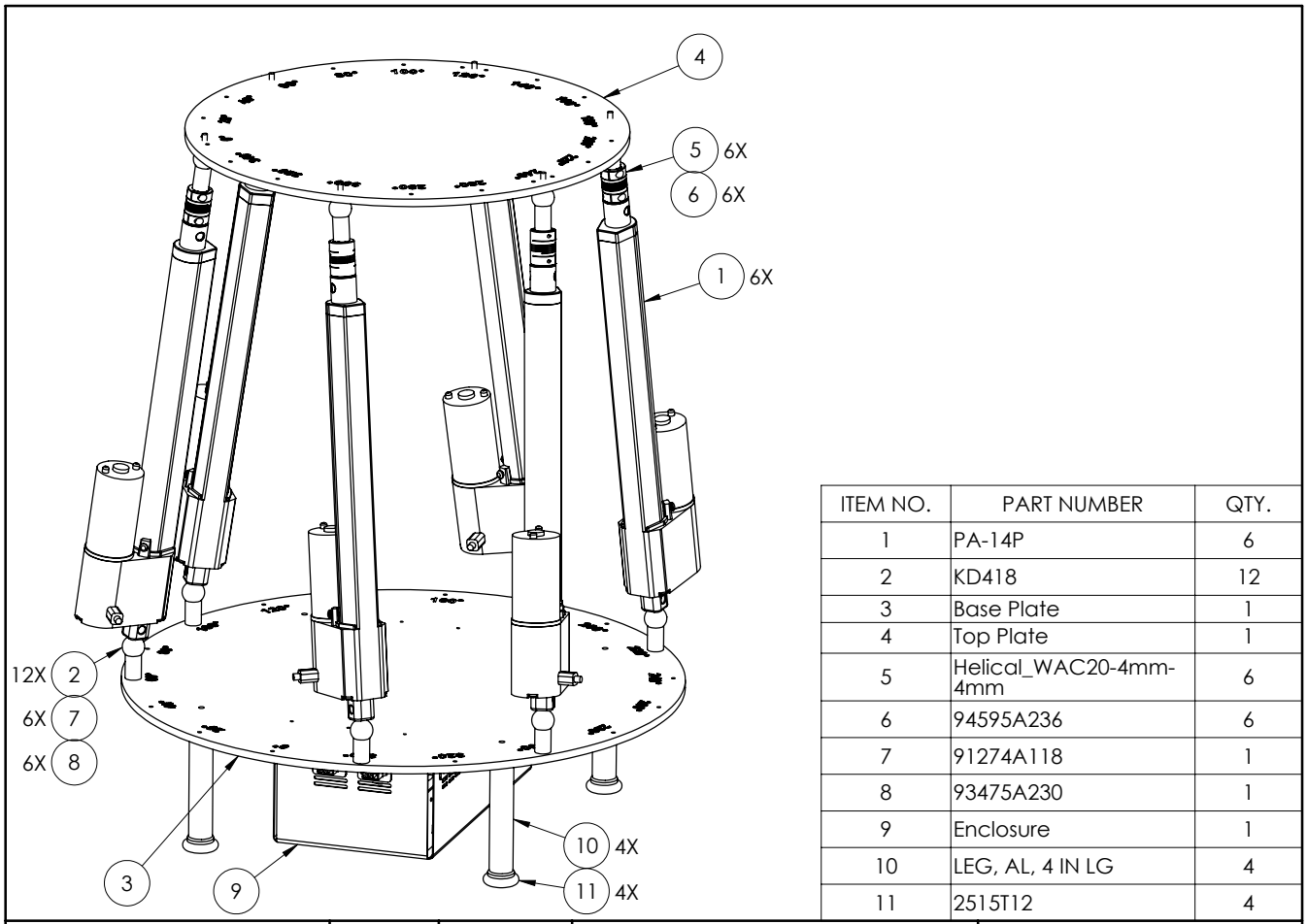
Cal Poly Mechanical Engineering	Section:01	Mat'l: AL, #14-#18 GA	Title: MOUNT, ARDUINO DUE		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #: 04	Nxt Asb:SGP-001	Date:6/28/2019	Scale: 1:2	Chkd. By: ME STAFF





ITEM NO.	PART NUMBER	QTY.
1	COVER, ELECTRONICS	1
2	12V 30A REGULATED PSU	1
3	ArduinoSubAssy	1
4	MUSB-D511-00-ND	1
5	723WX202	1
6	CR102J3RS215QF7	1
7	76650-0076	6
8	91780A427	4
9	92196A106	12

Cal Poly Mechanical Engineering	Section:01	Mat'l: N/A	Title: ENCLOSURE, ELECTRONICS		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #:01	Nxt Asb: SGP-001	Date:6/30/2019	Scale: 1:2	Chkd. By: ME STAFF



ITEM NO.	PART NUMBER	QTY.
1	PA-14P	6
2	KD418	12
3	Base Plate	1
4	Top Plate	1
5	Helical_WAC20-4mm-4mm	6
6	94595A236	6
7	91274A118	1
8	93475A230	1
9	Enclosure	1
10	LEG, AL, 4 IN LG	4
11	2515T12	4

Cal Poly Mechanical Engineering	Section:01	Mat'l: N/A	Title: STEWART GOUGH PLATFORM		Drwn. By:TRENT PETERSON
ME 599 SPRING 2019	Dwg. #:01	Nxt Asb:	Date:6/30/2019	Scale: 1:2	Chkd. By: ME STAFF

Appendix D

MECHANICAL BILL OF MATERIALS

Bill of Materials — Stewart-Gough Platform

Component - Stewart Platform	Price	Quantity	Shipping/Taxes	Total Cost	Vendor	Notes
KD418 Magnetic Ball Joint	$ 4.75	12	$ 12.83	$ 69.83	https://www.aliexpress.	Long lead
PA-14P-8-35 Feedback Actuator	$ 138.99	6	$ 12.98	$ 846.92	https://www.progressiv	50% Off with sponsorship
1/4 x 24 x 24 Acrylic Sheet	$ 20.75	1		$ 20.75	https://freckleface.com/	Base Plate
1/4 x 16 x 24 Acrylic Sheet	$ 13.83	1	$ 13.90	$ 27.73	https://freckleface.com/	Top Plate
WAC20-4mm-4mm	$ -	6	$ -	$ -	http://heli-cal.com/prod	Sponsored
94595A236 Threaded Rod M4 x 0.7 mm, 12 mm Long	$ 3.48	2	$ -	$ 6.96	https://www.mcmaster.	Pack of 10, need 12
93475A230 M4 SST Washer	$ 1.86	1	$ -	$ 1.86	https://www.mcmaster.	Pack of 100
90591A141 M4 Nut	$ 1.49	1	$ -	$ 1.49	https://www.mcmaster.	Pack of 100
91274A118 M4x12mm SHCS	$ 4.53	1	$ -	$ 4.53	https://www.mcmaster.	Pack of 50
McMaster Taxes and Shipping	$ -	0	$ 7.93	$ 7.93		

$ 988.00	Total - Main Platform
$ 571.03	Total After Rebate

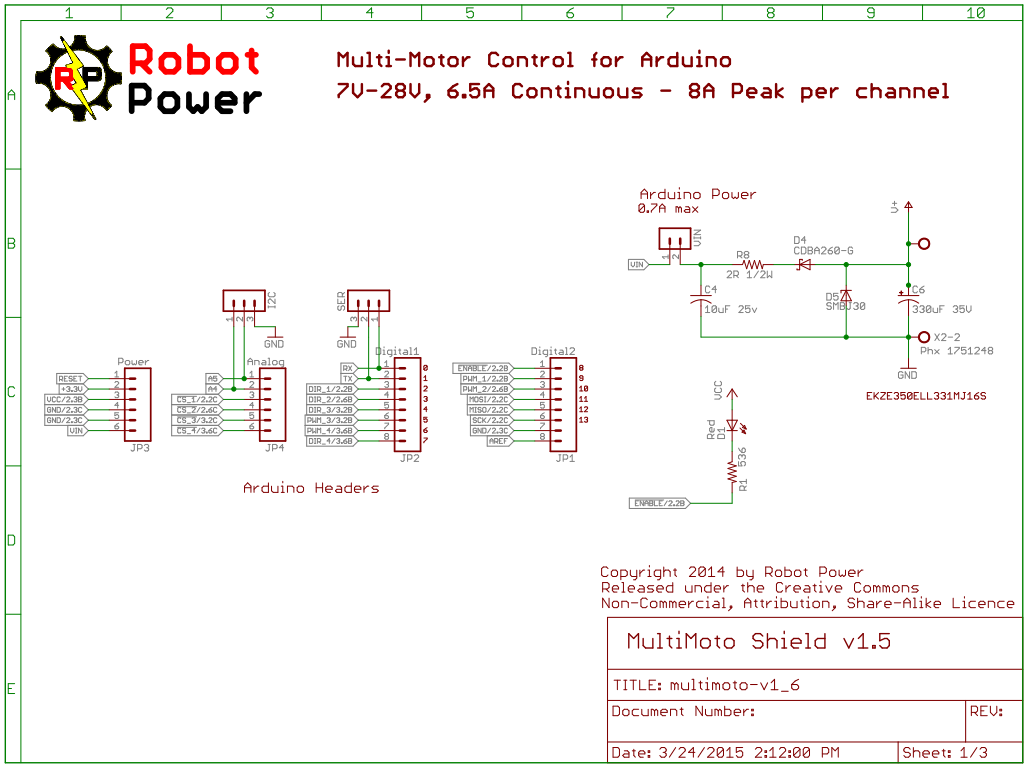
Component - Electronic Enclosure Hardware	Price	Quantity	Shipping/Taxes	Total Cost	Vendor	Notes
92196A106 4-40 x 0.25 SST SHCS	$ 3.92	1	$ -	$ 3.92	https://www.mcmaster.	Pack of 100
91780A427 4-40 x 13/32 Standoff	$ 0.38	10	$ -	$ 3.80	https://www.mcmaster.	Each
7565K65 Cable Holder	$ 5.11	1	$ -	$ 5.11	https://www.mcmaster.	Pack of 25
71535K51 Power Cord, 6 ft.	$ 5.44	1	$ -	$ 5.44	https://www.mcmaster.	Each
92141A005 4-40 Washer	$ 1.40	1	$ -	$ 1.40	https://www.mcmaster.	Pack of 100
8974K11 6061 0.75" Dia x 12" Lg	$ 4.24	2	$ -	$ 8.48	https://www.mcmaster.	1 ft length, Robot Feet (Qty 4, 5" Length each)
9284K313 1/8 ID Polyester Sleeving	$ 2.08	1	$ -	$ 2.08	https://www.mcmaster.	10 ft
7130K52 Zip Tie, 4"	$ 2.99	1	$ -	$ 2.99	https://www.mcmaster.	Pack of 100
McMaster Taxes and Shipping	$ -	0	$ 9.74	$ 9.74		

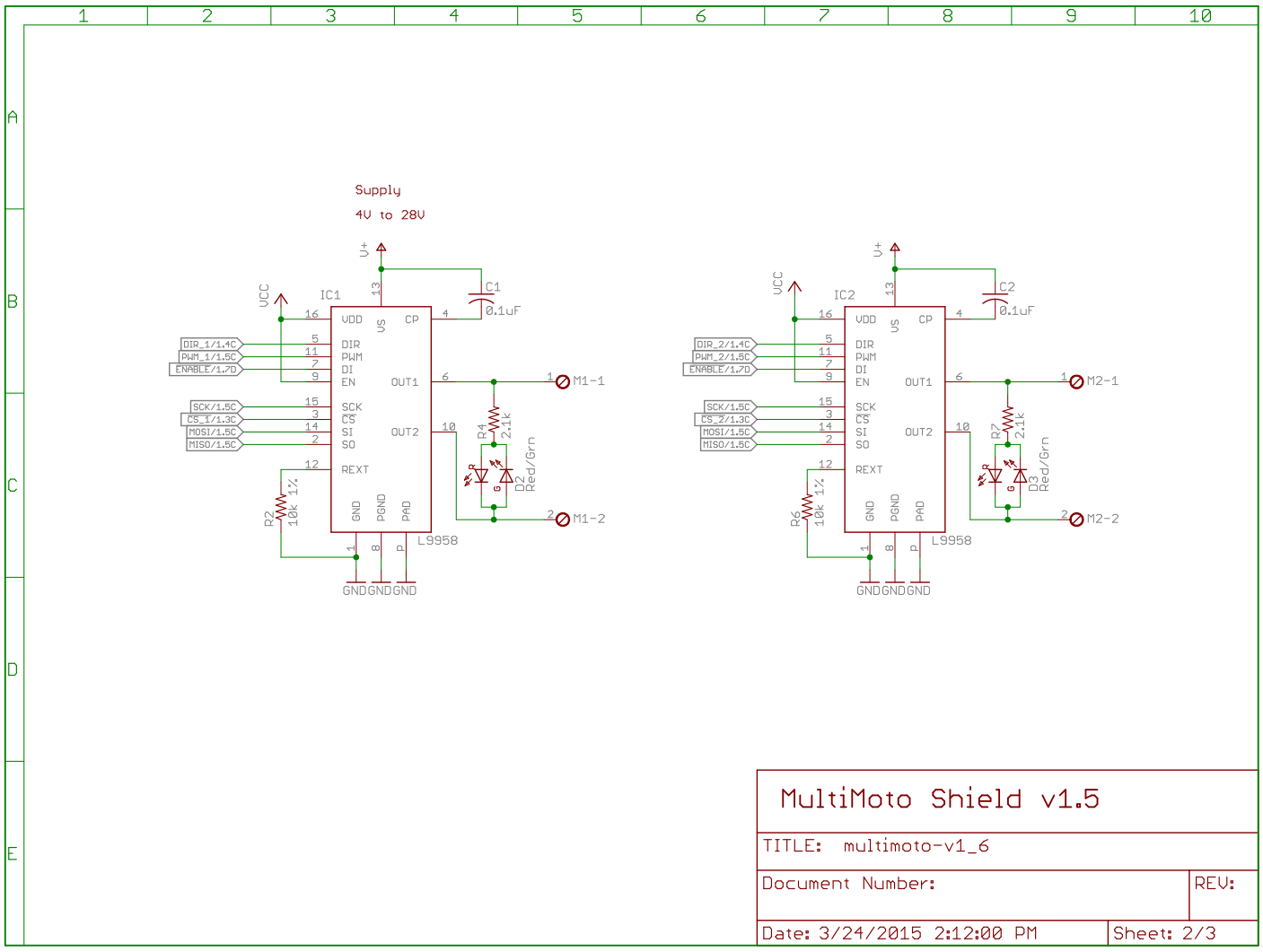
$ 42.96	Total - Electronics Enclosure (Hardware)

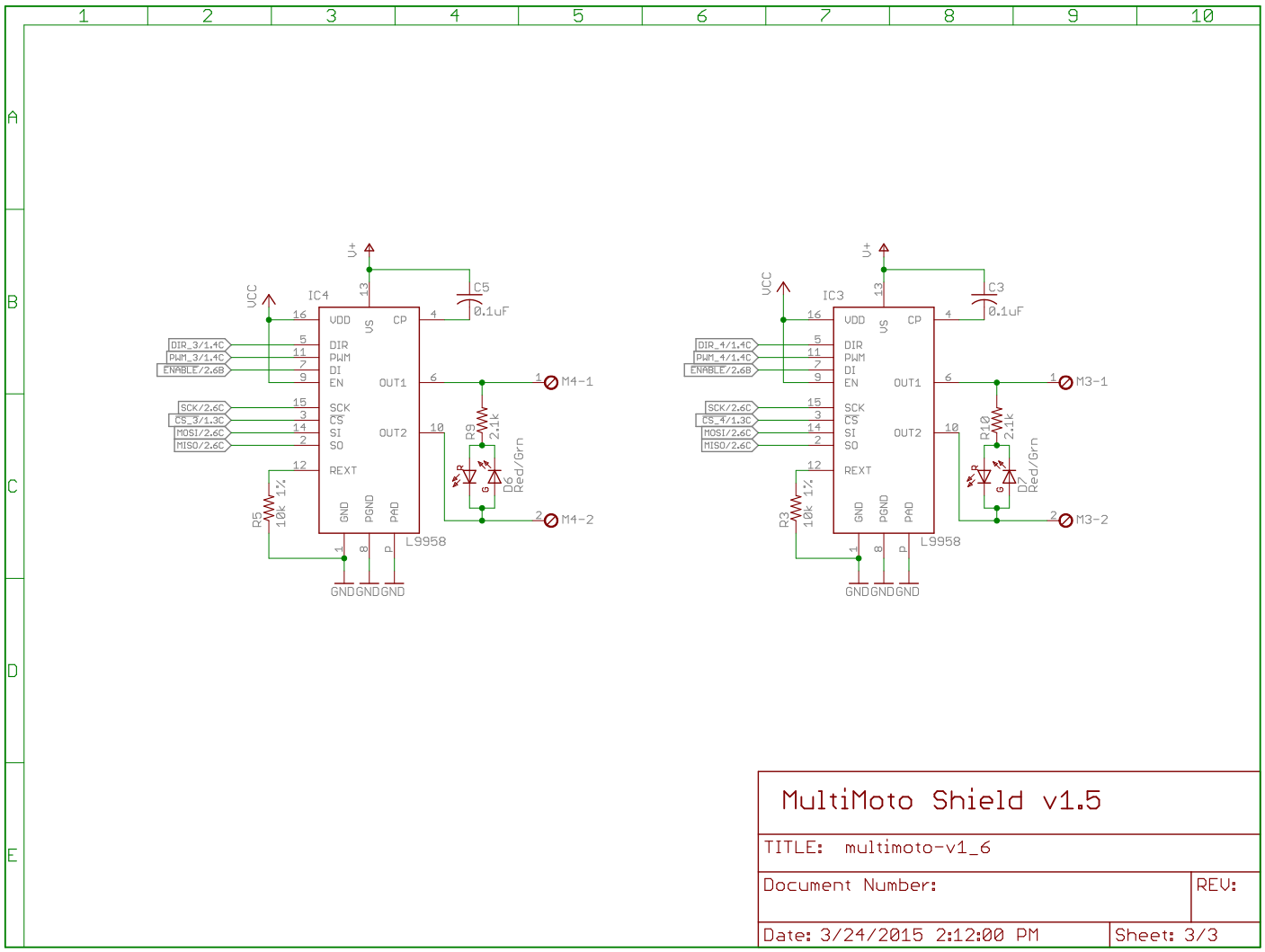
GRAND TOTAL (BEFORE SPONSORSHIP)	$ 1,223.61
GRAND TOTAL (AFTER PROG. AUTO. 50% RETROACTIVE REBATE)	$ 782.89

Appendix E

MULTIMOTO SCHEMATIC

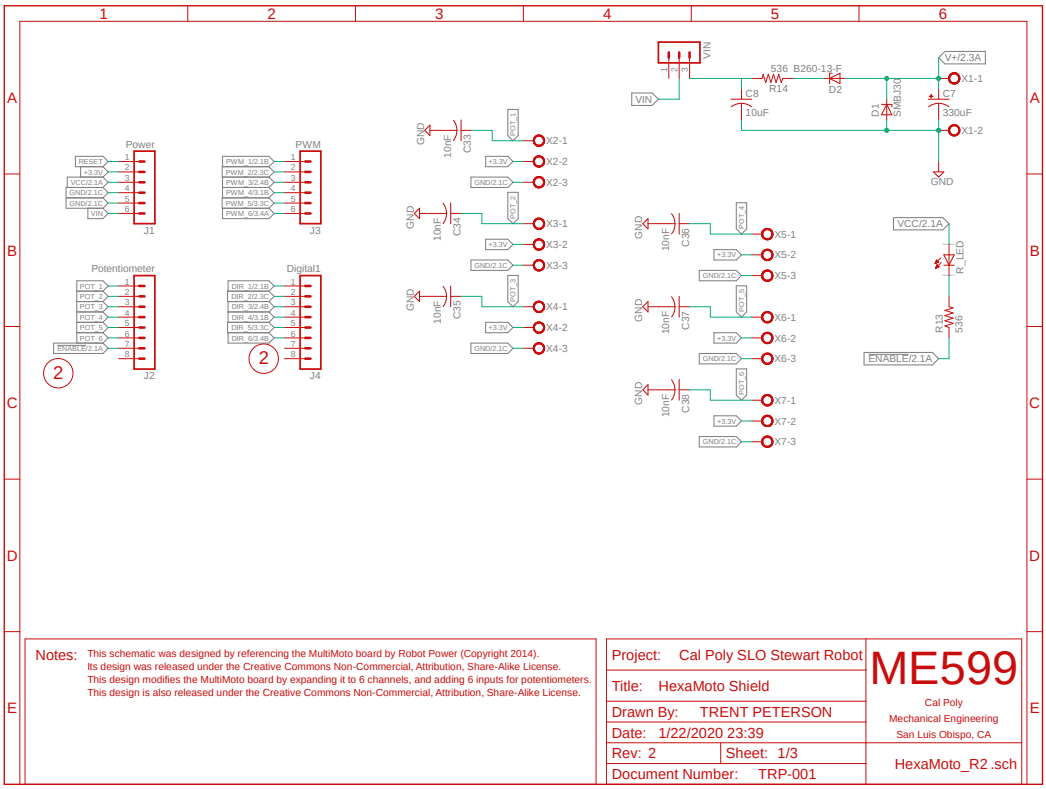


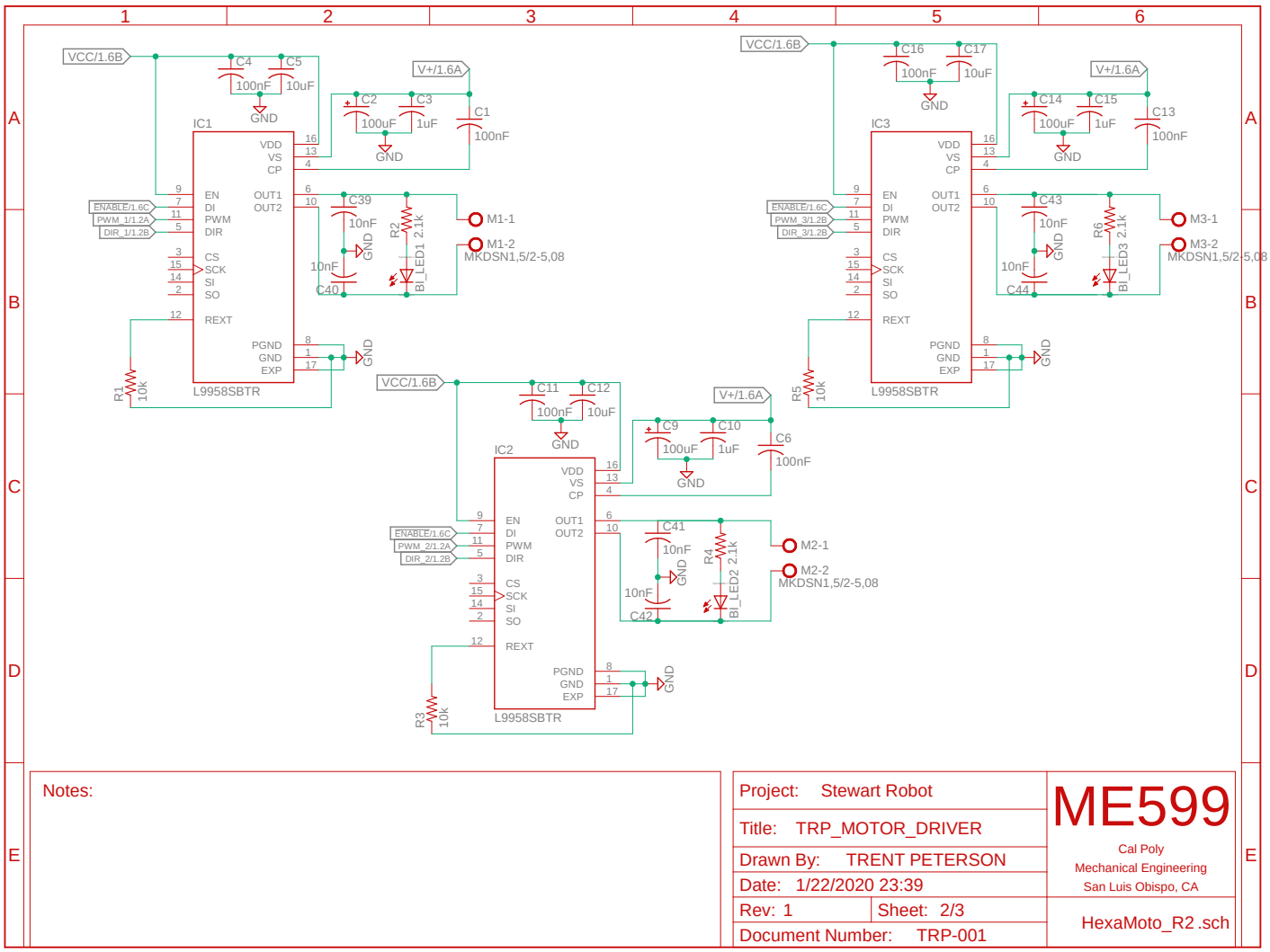

1 2 3 4 5 6 7 8 9 10
A B C D E
Supply
4V to 28V
U+
VCC
IC1
C1
0.1uF
VDD
VS
CP
DIR
PWM
DI
EN
OUT1
SCK
CS
SI
SO
OUT2
REXT
GND
PGND
PAD
L9958
DIR_1/1.4C
PWM_1/1.5C
ENABLE/1.7D
SCK/1.5C
CS_1/1.3C
MOSI/1.5C
MISO/1.5C
M1-1
M1-2
R2
10k 1%
R4
2.1k
D2
Red/Grn
IC2
C2
0.1uF
DIR_2/1.4C
PWM_2/1.5C
ENABLE/1.7D
CS_2/1.3C
M2-1
M2-2
R6
10k 1%
R7
2.1k
D3
Red/Grn
MultiMoto Shield v1.5
TITLE: multimoto-v1_6
Document Number:
REV:
Date: 3/24/2015 2:12:00 PM
Sheet: 2/3


1
2
3
4
5
6
7
8
9
10
A
B
C
D
E
U+
VCC
IC4
C5
0.1uF
VDD
VS
CP
DIR
PWM
DI
EN
OUT1
SCK
CS
SI
SO
OUT2
REXT
GND
PGND
PAD
L9958
DIR_3/1.4C
PWM_3/1.4C
ENABLE/2.6B
SCK/2.6C
CS_3/1.3C
MOSI/2.6C
MISO/2.6C
R5
10k 1%
R9
2.1k
D6
Red/Grn
M4-1
M4-2
GND
IC3
C3
0.1uF
DIR_4/1.4C
PWM_4/1.4C
ENABLE/2.6B
CS_4/1.3C
R3
10k 1%
R10
2.1k
D7
Red/Grn
M3-1
M3-2
MultiMoto Shield v1.5
TITLE: multimoto-v1_6
Document Number:
REV:
Date: 3/24/2015 2:12:00 PM
Sheet: 3/3

Appendix F

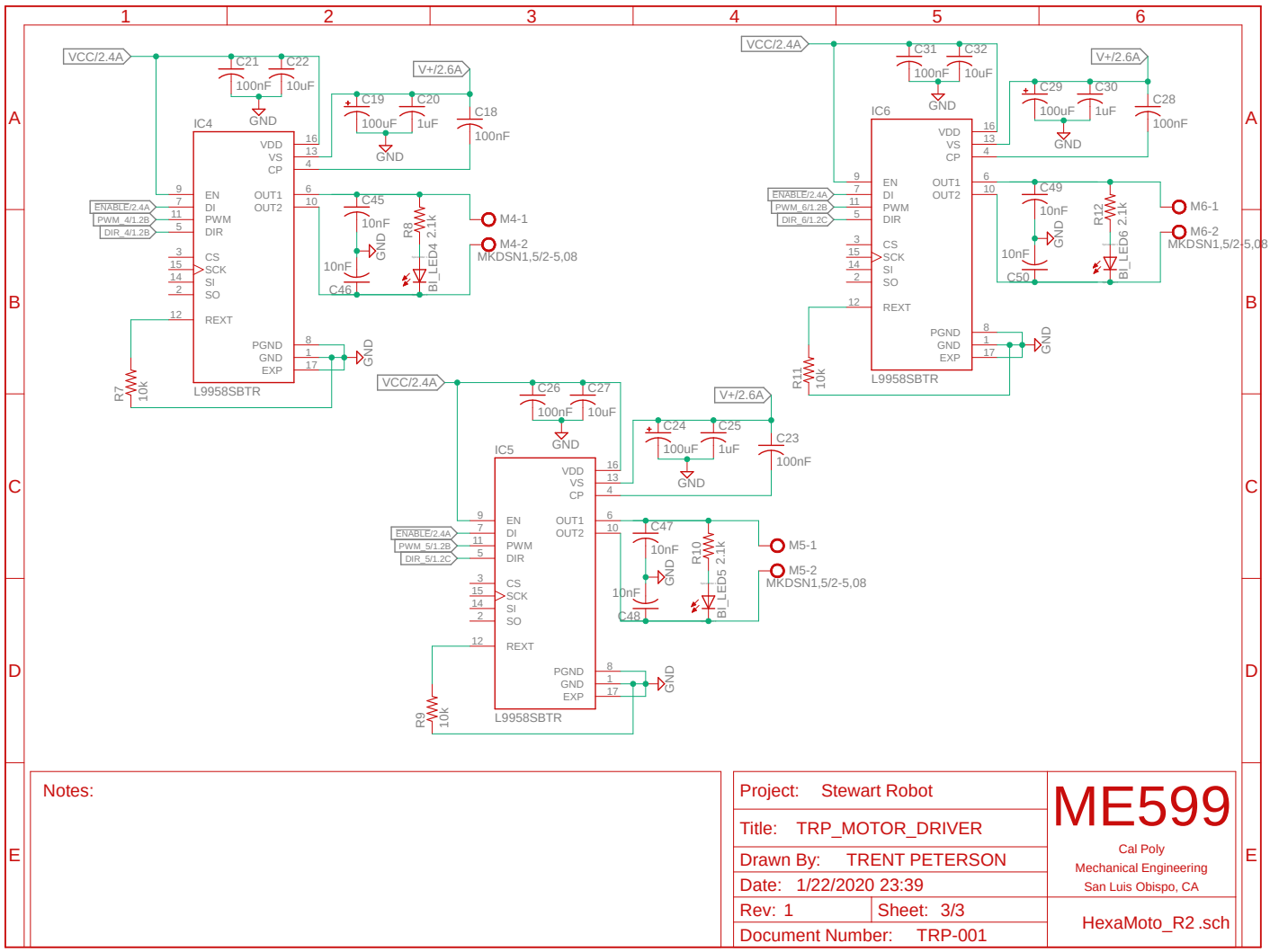
HEXAMOTO SCHEMATIC





Notes:

Project: Stewart Robot	ME599
Title: TRP_MOTOR_DRIVER	
Drawn By: TRENT PETERSON	Cal Poly Mechanical Engineering San Luis Obispo, CA
Date: 1/22/2020 23:39	
Rev: 1 / Sheet: 2/3	HexaMoto_R2 .sch
Document Number: TRP-001	



Notes:

Project:	Stewart Robot
Title:	TRP_MOTOR_DRIVER
Drawn By:	TRENT PETERSON
Date:	1/22/2020 23:39
Rev:	1
Sheet:	3/3
Document Number:	TRP-001

ME599

Cal Poly
Mechanical Engineering
San Luis Obispo, CA

HexaMoto_R2 .sch

Appendix G

ELECTRICAL BILL OF MATERIALS

Digikey Components

Qty Ordered	Cost (ea)	Total	Digikey PN	MFGR PN	Qty	Value	Device	Package	Parts	Description
10	$ 0.40	$ 3.95	1497-1314-1-ND	XZMDKVGS5W-4	7		LED_E	LED_1206	BI_LED1, BI_LED2, BI_LED3, BI_LED4, BI_LED5, BI_LED6, R_LED	LED
1	$ 0.16	$ 0.16	WM20124-ND	423751856	1		MA03-1	MA03-1	VIN	PIN HEADER
20	$ 0.13	$ 2.56	478-1395-1-ND	08055C104KAT2A	12	100nF	C-USC0805K	C0805K	C1, C4, C6, C11, C13, C16, C18, C21, C23, C26, C28, C31	CAPACITOR, American symbol
10	$ 0.15	$ 1.53	399-6546-ND	ESH107M025AE3AA	6	100uF	C-USC0805K	C0805K	C2, C9, C14, C19, C24, C29	CAPACITOR, American symbol
10	$ 0.14	$ 1.40	RNCF0805DTE10K0CT-ND	RNCF0805DTE10K0	7	10k	R-US_M0805	M0805	R1, R3, R5, R7, R9, R11, R13	RESISTOR, American symbol
25	$ 0.09	$ 2.20	478-10823-1-ND	08051C103K4T4A	18	10nF (10k pF)	C-USC0805K	C0805K	C33, C34, C35, C36, C37, C38, C39, C40, C41, C42, C43, C44, C45, C46, C47, C48, C49, C50	CAPACITOR, American symbol
14	$ 0.12	$ 1.67	1276-1096-1-ND	CL21A106KOQNNNE	7	10uF	C-USC0805K	C0805K	C5, C8, C12, C17, C22, C27, C32	CAPACITOR, American symbol
10	$ 0.06	$ 0.60	1276-1026-1-ND	CL21B105KOFNNNE	6	1uF	C-USC0805K	C0805K	C3, C10, C15, C20, C25, C30	CAPACITOR, American symbol
10	$ 0.08	$ 0.82	P2.10KCCT-ND	ERJ-6ENF2101V	6	2.1k	R-US_M0805	M0805	R2, R4, R6, R8, R10, R12	RESISTOR, American symbol
3	$ 0.73	$ 2.19	565-1693-ND	EKZE350ELL331MJ16S	1	330uF	CPOL-USTT5D10	TT5D10	C7	POLARIZED CAPACITOR, American symbol
2	$ 0.38	$ 0.76	P19275CT-ND	ERJ-6DQF2R0V	1	2	R-US_M0805	M0805	R14	
2	$ 0.36	$ 0.72	P2056JCT-ND	ERJ-PB6B5360V	1	536	R-US_M0805	M0805	R13	RESISTOR, American symbol
2	$ 0.42	$ 0.84	B260-FDICT-ND	B260-13-F	1	B260-13-F	ZENER-DIODESMB	SMB	D2	Z-Diode
3	$ 0.35	$ 1.05	WM50014-08-ND	22284080	2	Digital1, Pot	MA08-1	MA08-1	J4, J2	PIN HEADER
7	$ 5.46	$ 38.22	497-11667-1-ND	L9958SBTR	6	L9958SBTR	L9958SBTR	SOP80P1410X360-17N	IC1, IC2, IC3, IC4, IC5, IC6	L9958 Series H-Bridge - PowerSO16
10	$ 0.59	$ 5.93	WM4393-ND	398800302	7	MKDSN1,5/2-5,08	MKDSN1,5/2-5,08	MKDSN1,5/2-5,08	X1, M1, M2, M3, M4, M5, M6	MKDSN 1,5/ 2-5,08 Printklemme
10	$ 0.75	$ 7.48	WM6289-ND	398800303	6	MKDSN1,5/3-5,08	MKDSN1,5/3-5,08	MKDSN1,5/3-5,08	X2, X3, X4, X5, X6, X7	MKDSN 1,5/ 3-5,08 Printklemme
3	$ 0.26	$ 0.78	WM50014-06-ND	22284060	2	PWM, Power	MA06-1	MA06-1	J1, J3	PIN HEADER
2	$ 0.54	$ 1.08	SMBJ30ABCT-ND	SMBJ30A	1	SMBJ30	ZENER-DIODESMB	SMB	D1	Z-Diode
6	$ 1.26	$ 7.56	WM8369-ND	76650-0076	The right hand side above this note is the minimum BOM required to make one Motor Driver PCB. The quantity ordered (top left) includes a reasonable spare parts, a best design practice for PCB creation. Spare part total (~$10)					KIT CONN MINI-FIT JR 6 CIRCUITS
1	$ 2.53	$ 2.53	Q853-ND	3025010-03						USB A MALE TO MICRO B MALE 3'
1	$ 2.33	$ 2.33	Q361-ND	3021001-03						CBL USB A-B CON 3' 28/28 AWG
1	$ 1.52	$ 1.52	Q204-ND	723W-X2/02	<----Additional Digikey Parts (Enclosure Electronic Hardware) ---->					PWR ENT MOD RCPT IEC320-C14 PNL
1	$ 11.28	$ 11.28	MUSB-D511-00-ND	MUSB-D511-00						CONN RCPT USB2.0 TYPEB 4POS PCB
5	$ 0.23	$ 1.15	507-1254-ND	SST 6.3-R						FUSE GLASS 6.3A 250VAC 5X20MM

Merchandies Total	$ 100.31
Shipping	$ 4.99
Taxes	$ 7.77
Digikey Total	$ 113.07

Components - Electron	Price	Quantity	Shipping	Total Cost	Vendor	Notes
Arduino Due	$ 47.50	1	$ -	$ 47.50	https://www.progres	Dept Order, 50% Off with sponsorship
12V DC 30A Regulated Power Supply	$ 19.50	1	$ -	$ 19.50	https://www.ebay.com/itm/231849834132	
Motor Driver - PCB	$ 0.40	5	$ -	$ 2.00		
Motor Driver - Solder Stencil	$ 6.00	1	$ 4.58	$ 10.58	https://jlcpcb.com/	

Total - Electronics (Non-Digikey)	$ 79.58
Total After Rebate	$ 55.83

Appendix H

REPLACEMENT MANUAL

STEWART-GOUGH PLATFORM

REPLACEMENT GUIDE- ELECTRONICS

INTRODUCTION:

This is an instruction manual that will detail how to replace the HexaMoto Arduino Shield and the Arduino Due.

SOFTWARE STEPS (Arduino Due):

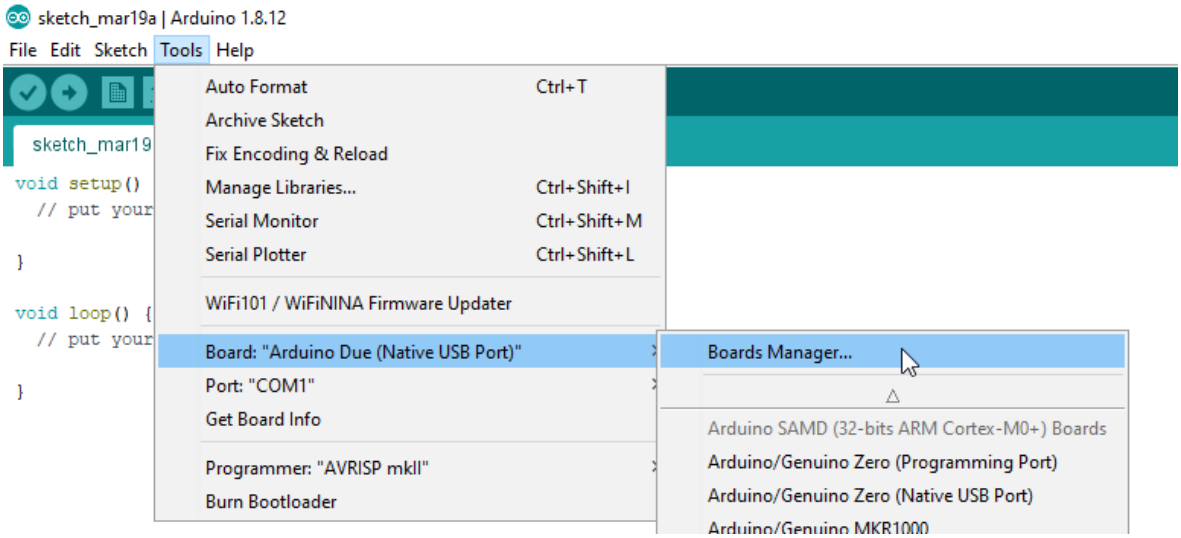
1. Download the Arduino IDE from https://www.arduino.cc/en/main/software by using the "Windows Installer, for Windows XP and up" link on the page. Run the executable and complete the installation.

2. Connect to the Arduino Due with a PC with the Arduino IDE installed.

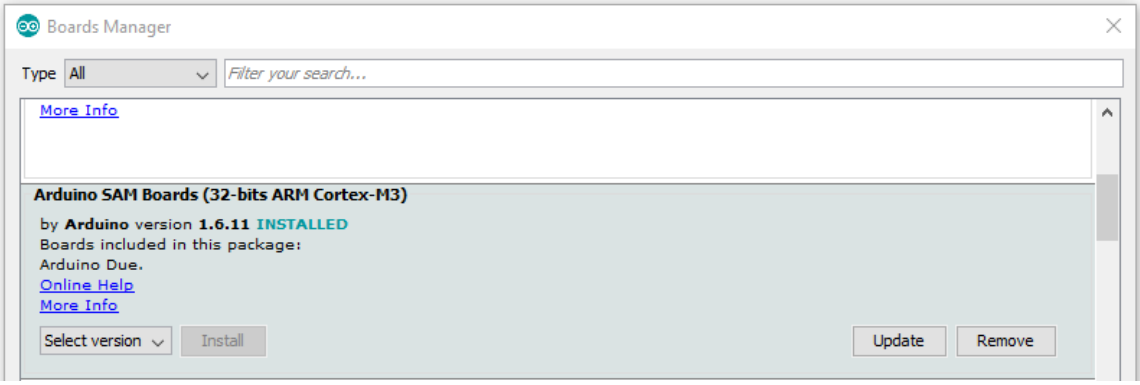
3. The program will launch, and the loading screen should appear.

4. Once the program is launched, hover over the Include Library selection under the Sketch tab. Select the Add .ZIP library… option that appears and select the provided StewartPlatformLibrary.zip file.

5. Ensure that the Arduino Due boards are loaded in the IDE. If they are, they will be located under the Board: … option under the Tools tab at the bottom of the list. If they are not present, they can be added with the Boards Manager.



Install the Arduino SAM Boards (32-bits ARM Cortex-M3) which will allow the Arduino Due board to be selected once installed.



6. Load the Stewart Platform sketch into the Arduino IDE. The program should look similar to the following.

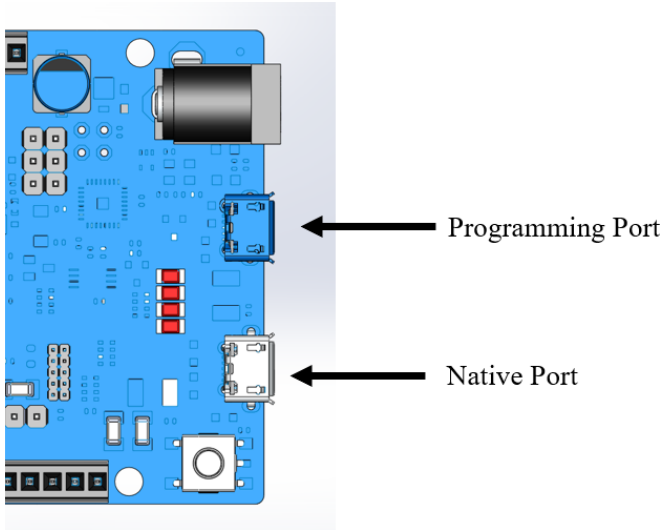
platform_TRP | Arduino 1.8.9

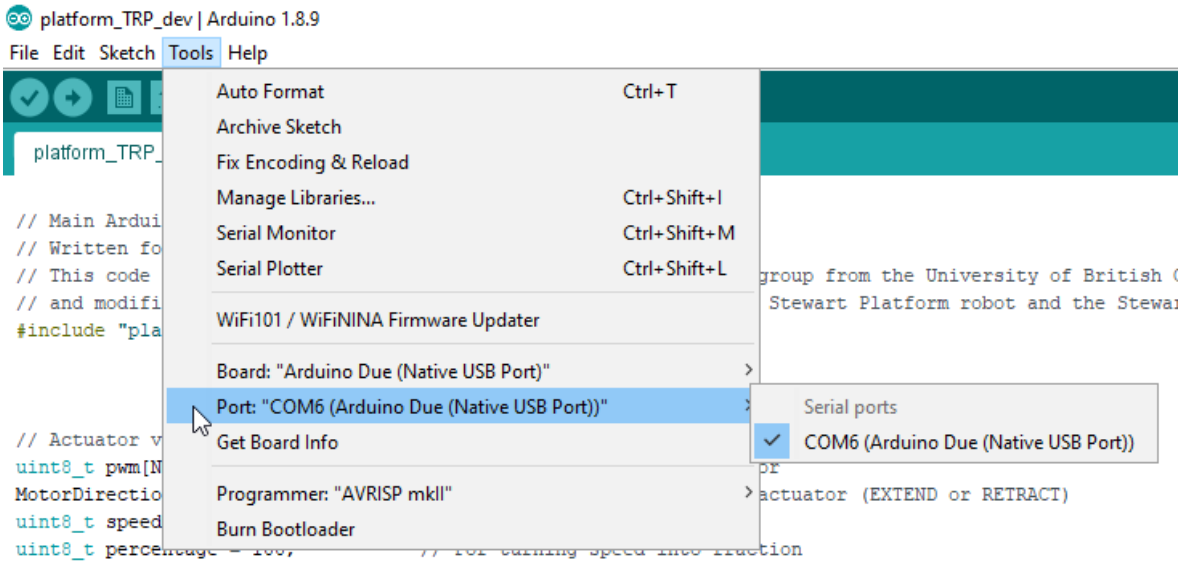
File Edit Sketch Tools Help

platform_TRP

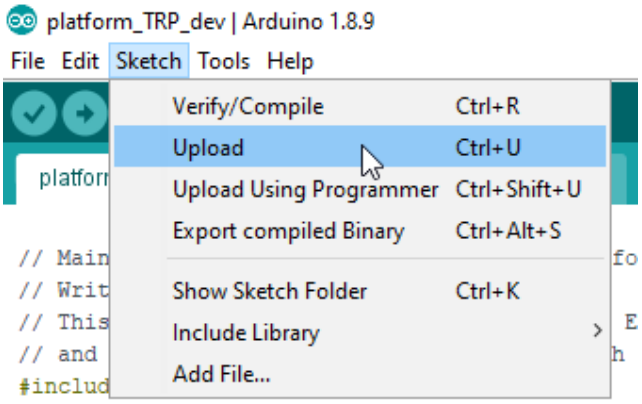
```
// Main Arduino code for a 6-dof Stewart platform.
// Written for the Arduino Due.
// This code has been taken from the ENPH 459 Engineering Physics group from the University of British Columbia
// and modified by Trent Peterson to work with the Cal Poly ME 423 Stewart Platform robot and the Stewart Platform Motor Driver PCB hat.
#include "platform.h"

// Actuator variables
uint8_t pwm[NUM_MOTORS];            // current PWM for each actuator
MotorDirection dir[NUM_MOTORS];     // current direction for each actuator (EXTEND or RETRACT)

// Position variables
int16_t pos[NUM_MOTORS];            // current position (measured by analog read) of each actuator
int16_t input[NUM_MOTORS];          // intermediate input retrieved from the SerialUSB buffer
uint16_t desired_pos[NUM_MOTORS];   // desired (user-inputted and validated) position of each actuator

// Feedback variables
int16_t pos_diff;                   // difference between current and desired position
int16_t prop_diff;                  // position difference used for proportional gain
int16_t previous_diff[NUM_MOTORS];  // last position difference for each actuator; for derivative gain
int16_t total_diff[NUM_MOTORS];     // cumulative position difference for each actuator; for integral gain
uint8_t previous_inst[NUM_MOTORS];  // starting/past sample instance for each actuator; for derivative gain
uint8_t current_inst[NUM_MOTORS];   // new/incrementing sample instance for each actuator; for derivative gain
float p_corr;                       // proportional correction for PID
float i_corr;                       // integral correction for PID
float d_corr;                       // differential correction for PID
float corr;                         // final feedback PWM value from PID correction
```

7. Ensure the device selected is an Arduino Due and that the COM Port is correct. The COM Port may be found in Device Manager in Windows. The Arduino Due has two USB ports: Native and Programming. For this manual's purpose, the Native port has a much faster serial connection allowing for quick uploads, but the Programming port is more robust when uploading sketches. Either should work, as long as the Arduino IDE selection of Native or Programming port matches the connection made with the Arduino Due.





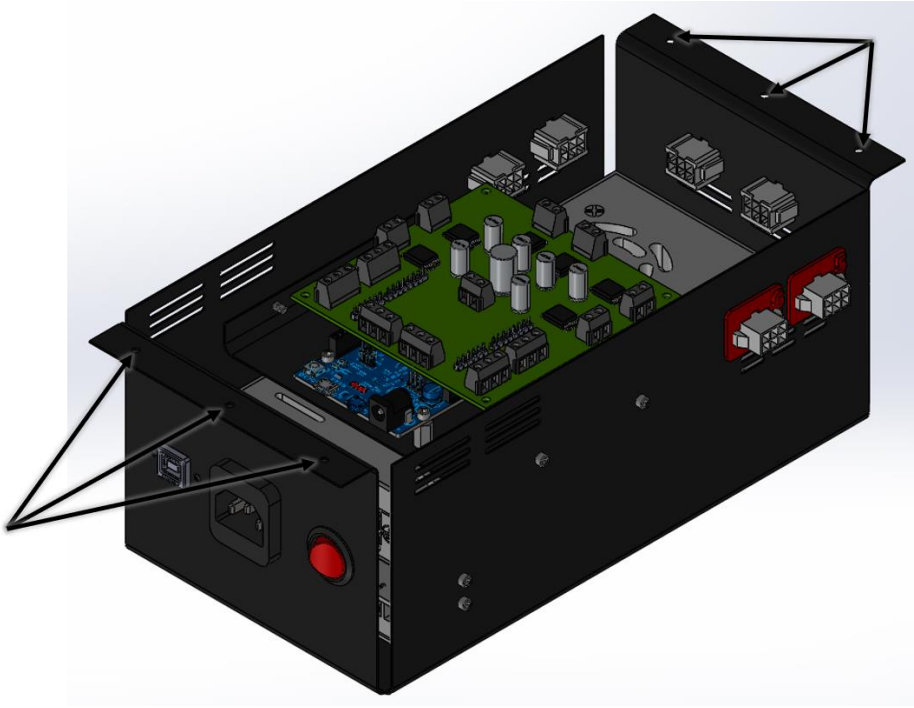
8. Once the sketch is loaded and the Arduino IDE is properly configured to upload to an Arduino Due, use the upload tool to put the sketch on the Arduino Due. The IDE should show the sketch compiling, then reading/writing to the Due, followed by a completion message.



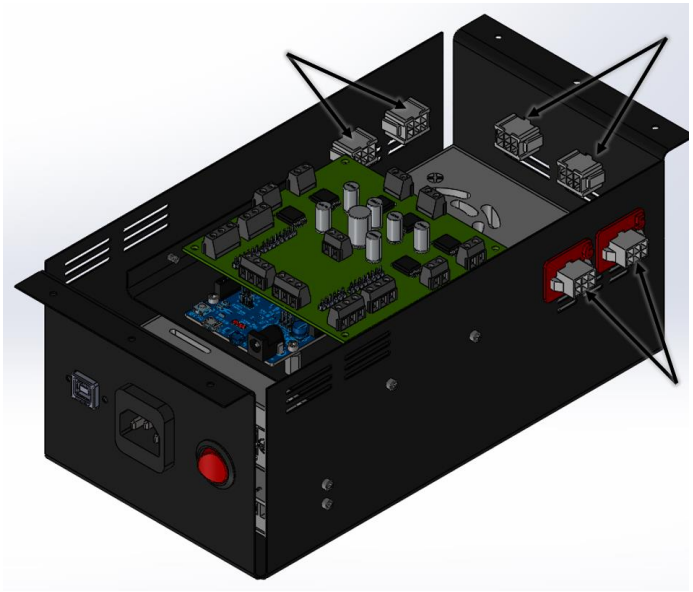
9. The Arduino Due may be simply unplugged from the PC.

HARDWARE STEPS:

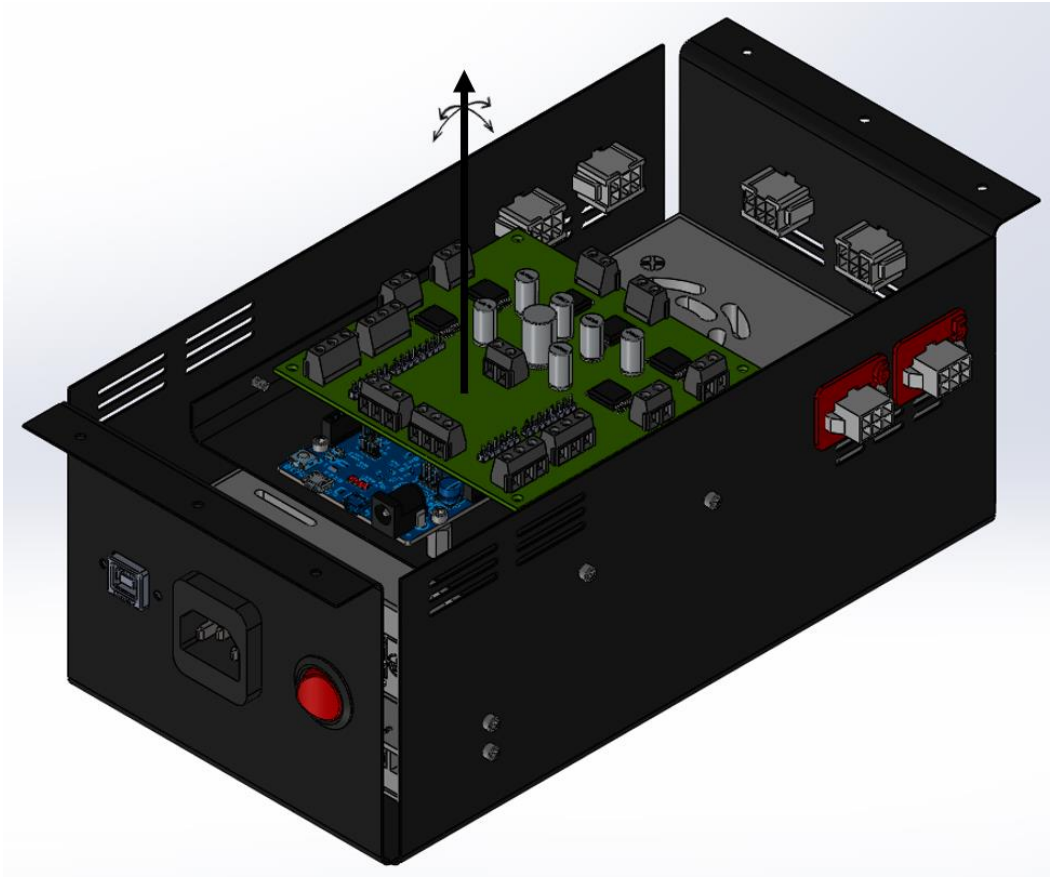
1. Unplug the USB and power cables from the ports in the front of the electronics platform. Unscrew the six (6) fasteners that attach the electronics enclosure to the bottom platform and lower it onto the working surface.



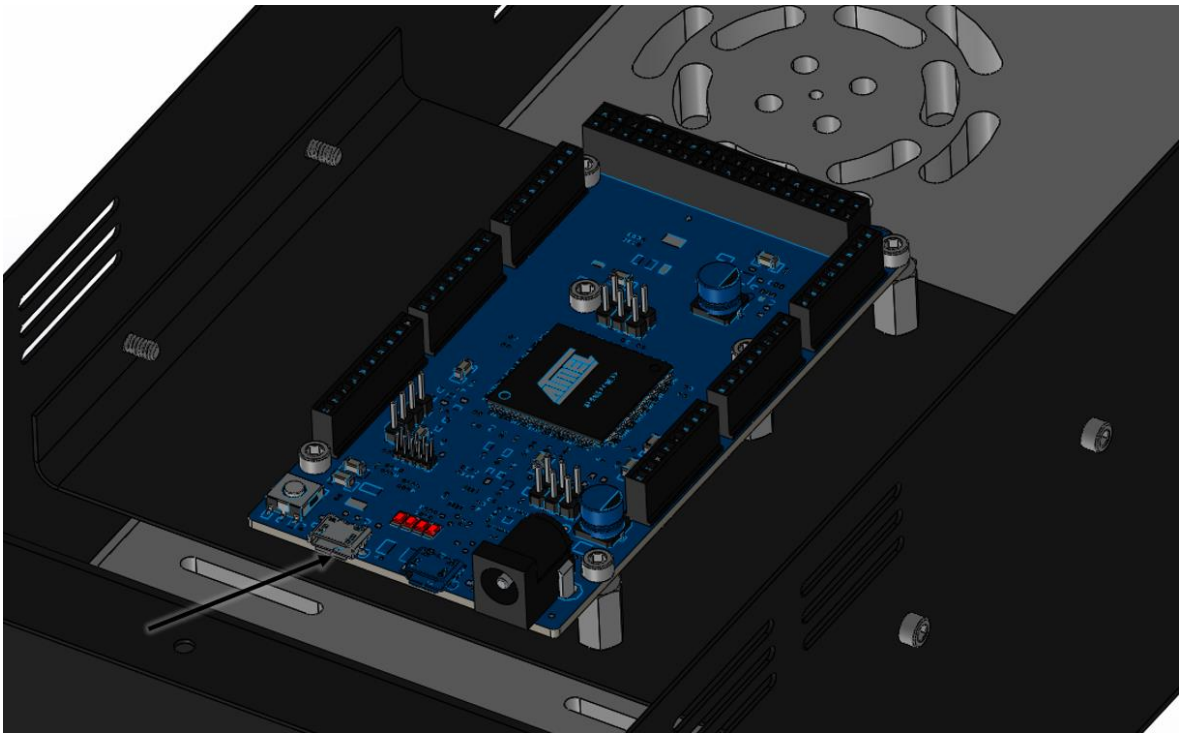
2. Unplug the six (6) actuator connectors from the electronics enclosure.



3. Remove the HexaMoto Arduino shield from the Arduino Due. The shield is connected by pins. Great care should be taken to not bend the pins upon removal. This is best accomplished by grabbing the sides of the shield, and slightly rocking it back and forth and left and right, as depicted by the arrows, to slowly lift the shield. To install a new HexaMoto, align the shield's pins with the Due's female pin headers and insert, with care not to bend the pins. Complete these steps in reverse order to conclude the Hexamoto replacement. If replacing the Arduino Due, continue to Step 4.



4. Remove the screws that attach the Arduino Due to the sheet metal mount. Note, due to hole alignment, not all 6 screws may be present. Unplug the USB cable from the Native USB Port, pointed to by the arrow.



5. Replace the old Arduino Due with a new Arduino Due that has the Stewart Platform code loaded. Ensure that the USB cable is plugged back into the Native USB Port. Follow all steps in reverse order to reassemble the electronics enclosure.

Appendix I

MATLAB SIMULATION SCRIPT

Table of Contents

Stewart Gough Platform Equations of Motion

This script will take in, in variable form, the dimensions of the bottom and top of the robot, a and b respectively, the values of the theta, phi, and psi, respective to the x, y, and z axis respectively, and the position vector P, and output the link lengths of each linear actuator. The outputs are then simulated for a predetermined motion profile and various metrics are

```
%By Trent Peterson
```

Clear Previous

```
close; clc; clear;
```

Set Constants

```
base_diameter       = 18;                       %[inches], acrylic
 platform bolt circle
top_diameter        = 14;                       %[inches], acrylic
 platform bolt circle
a                   = (base_diameter)/2;        %[inches], distance
 from center of base to actuator
b                   = (top_diameter)/2;         %[inches], distance
 from center of top to actuator

%Set base and top angles
base_angle = [ 0 60 120 180 240 300 ];          %[degrees], Spacing of
 actuator pairs, base
base_angle = sort(base_angle);                  %sorted for
 calculation integrity

top_angle = [ 0 40 120 160 240 280 ];           %[degrees], Spacing of
 actuator pairs, top
top_angle = sort(top_angle);                    %sorted for
 calculation integrity

% Set configuration
% 6-3 Configuration when the base joints are spaced 60° and the top
 joints
```

```
% pairs are spaced ~120° (Physical joint angle accounted for)
% 6-6 Configuration when all joints are spaced 60° (unstable
 configuration)
configuration_6_3   = false;                    % Run preset 6-3
 configuration simulation
configuration_6_6   = false;                     % Run preset 6-6
 configuration simulation
animation           = false;                    % Display Animation
plots               = true;                    % Display Plots
%Set physical parameters
stroke_length       = 8;                        %[inches]
joint_angle         = 3.47;                     %[degrees], for 6_3
 configuration
min_joint_length    = 16.7;                     %[inches], from CAD
max_joint_length    = 24.7;                     %[inches], from CAD
min_joint_velocity  = -2;                       %[inches/sec], from
 Progressive Automations
max_joint_velocity  = 2;                        %[inches/sec], from
 Progressive Automations
% Set up motion variables
n= 1:400;                                           % Array dimension
[theta,phi,psi,Px,Py,Pz]=deal(zeros(1,length(n)));  % Create position
 array
time_duration= 10;                                  % Seconds
time = 0:(time_duration/(max(n)-1)):time_duration;  % Create time
 array, position
time_dot = time;                                    % Create time
 array, velocity
time_dot(end)=[];                                   % Delete array
 end, discrete differentiation
motion_profile = 2;                                 % Choose motion
 profile

% Preallocation, create joint locations and distance arrays
a_i         = zeros(4,6,length(theta));
b_i         = zeros(4,6,length(theta));
b_rot       = zeros(4,6,length(theta));
L_vector    = zeros(4,6,length(theta));
L_length    = zeros(length(theta),6);
L_length_dot = L_length;
L_length_dot(length(n),:)=[];
```

Motion Profile Creation

Create function based motion per degree of freedom (independently) based on the time duration over n steps. All motions are allowed. Resulting plots will determine if move profile is valid or not

```
if motion_profile == 1
    for m = 1:length(n)
       theta(m)    = -30 + 30 * m/max(n);       %[degrees], relative
 to x axis
       phi(m)      =  0;                        %[degrees], relative
 to y axis
```

```
        psi(m)      = -30 + 60 * m/max(n);       %[degrees], relative
 to z axis
        Px(m)       = 0;                         %[inches], x position
        Py(m)       = 0;                         %[inches], y position
        Pz(m)       = 20 + 2*cosd(m/max(n)*360); %[inches], z position
    end
end

if motion_profile == 2
    for m = 1:length(n)
        theta(m)    = 10;                        %[degrees], relative
 to x axis
        phi(m)      = 10*cosd(m/max(n)*360);     %[degrees], relative
 to y axis
        psi(m)      = 0;                         %[degrees], relative
 to z axis
        Px(m)       = 3 * m/max(n);              %[inches], x position
        Py(m)       = 0;                         %[inches], y position
        Pz(m)       = 22 - 3*m/max(n) ;          %[inches], z position
    end
end

if motion_profile == 3
    for m = 1:length(n)
        theta(m)    = 0;                         %[degrees], relative
 to x axis
        phi(m)      =  10;                       %[degrees], relative
 to y axis
        psi(m)      = -15 + 25 * m/max(n);       %[degrees], relative
 to z axis
        Px(m)       = 3;                         %[inches], x position
        Py(m)       = -2 + 3 * m/max(n);         %[inches], y position
        Pz(m)       = 20; %[inches], z position
    end
end

if motion_profile == 4
    for m = 1:length(n)
      theta(m)    = -15 + 30 * m/max(n);         %[degrees], relative
 to x axis
      phi(m)      =  10 - 5 * m/max(n);          %[degrees], relative
 to y axis
      psi(m)      = 15*sind(m/max(n)*360);       %[degrees], relative
 to z axis
      Px(m)       = 1.5;                         %[inches], x position
      Py(m)       = -3 + 4*m/max(n);             %[inches], y position
      Pz(m)       = 19.8 + 2*cosd(m/max(n)*360);%[inches], z position
    end
end

P = [Px;Py;Pz;zeros(1,length(theta))]; %combine into array
```

Calculate EOMs

```
if configuration_6_6
    for j = 1:length(theta)
        for i = 1:6
            %calculate a vector for each base location
            a_i(1,i,j)=a*cosd((i-1)*60);
            a_i(2,i,j)=a*sind((i-1)*60);
            a_i(3,i,j)=0;
            a_i(4,i,j)=1;
            %calculate b vector for each top location
            b_i(1,i,j)=b*cosd((i-1)*60);
            b_i(2,i,j)=b*sind((i-1)*60);
            b_i(3,i,j)=0;
            b_i(4,i,j)=1;
            %convert b from global to rotated

 b_rot(:,i,j)=stewartrot(theta(j),phi(j),psi(j),b_i(:,i,j));
            L_vector(:,i,j)  = P(:,j) + b_rot(:,i,j) - a_i(:,i,j);
            L_length(j,i) = sqrt(L_vector(1,i,j)^2 + L_vector(2,i,j)^2
 + L_vector(3,i,j)^2);
        end
    end

elseif configuration_6_3
    for j = 1:length(theta)
    %calculate b vector for each top location
        for i = [1 3 5]
            b_i(1,i,j)=b*cosd((i-1)*60-joint_angle);
            b_i(2,i,j)=b*sind((i-1)*60-joint_angle);
            b_i(3,i,j)=0;
            b_i(4,i,j)=1;
            %----------------------------------------
            b_i(1,i+1,j)=b*cosd((i-1)*60+joint_angle);
            b_i(2,i+1,j)=b*sind((i-1)*60+joint_angle);
            b_i(3,i+1,j)=0;
            b_i(4,i+1,j)=1;
        end

        for i = 1:6
            %calculate a vector for each base location
            a_i(1,i,j)=a*cosd((i-1)*60);
            a_i(2,i,j)=a*sind((i-1)*60);
            a_i(3,i,j)=0;
            a_i(4,i,j)=1;

            %convert b from global to rotated

 b_rot(:,i,j)=stewartrot(theta(j),phi(j),psi(j),b_i(:,i,j));
            L_vector(:,i,j)  = P(:,j) + b_rot(:,i,j) - a_i(:,i,j);
            L_length(j,i) = sqrt(L_vector(1,i,j)^2 + L_vector(2,i,j)^2
 + L_vector(3,i,j)^2);
        end
```

```
    end

else %for all non-true 6-6 and 6-3 cases
    for j = 1:length(theta)
        for i = 1:6
            a_i(1,i,j)=a*cosd(base_angle(i));
            a_i(2,i,j)=a*sind(base_angle(i));
            a_i(3,i,j)=0;
            a_i(4,i,j)=1;
            %----------------------------------------
            b_i(1,i,j)=b*cosd(top_angle(i));
            b_i(2,i,j)=b*sind(top_angle(i));
            b_i(3,i,j)=0;
            b_i(4,i,j)=1;


 b_rot(:,i,j)=stewartrot(theta(j),phi(j),psi(j),b_i(:,i,j));
            L_vector(:,i,j)  = P(:,j) + b_rot(:,i,j) - a_i(:,i,j);
            L_length(j,i) = sqrt(L_vector(1,i,j)^2 + L_vector(2,i,j)^2
 + L_vector(3,i,j)^2);
        end
    end
end
```

Derivative Calculation

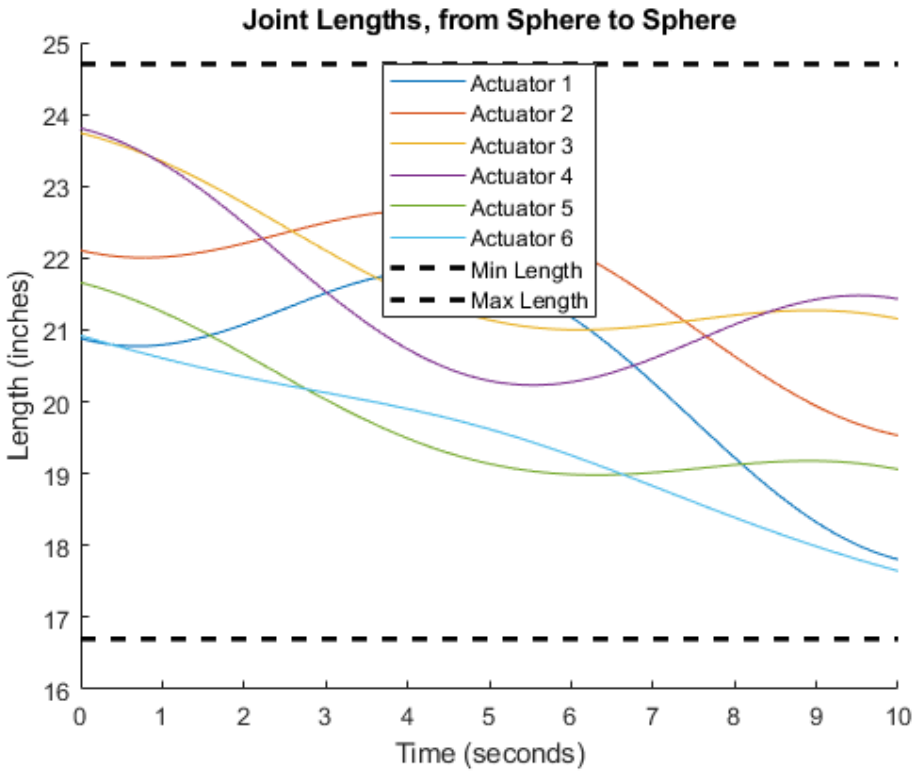
```
[theta_dot,phi_dot,psi_dot,Px_dot,Py_dot,Pz_dot,P_dot]=deal(zeros(1,length(n)-1));

for i=2:length(n)

    Px_dot(i-1) = (Px(i) - Px(i-1))/(time(i)-time(i-1));
    Py_dot(i-1) = (Py(i) - Py(i-1))/(time(i)-time(i-1));
    Pz_dot(i-1) = (Pz(i) - Pz(i-1))/(time(i)-time(i-1));
    P_dot(i-1) = sqrt((Px_dot(i-1))^2 + (Py_dot(i-1))^2 +
 (Pz_dot(i-1))^2);
    theta_dot(i-1) = (theta(i) - theta(i-1))/(time(i)-time(i-1));
    phi_dot(i-1) =(phi(i) - phi(i-1))/(time(i)-time(i-1));
    psi_dot(i-1) =(psi(i) - psi(i-1))/(time(i)-time(i-1));

    for j = 1:6
         L_length_dot(i-1,j) = (L_length(i,j) - L_length(i-1,j))/
(time(i)-time(i-1));
    end
end
```

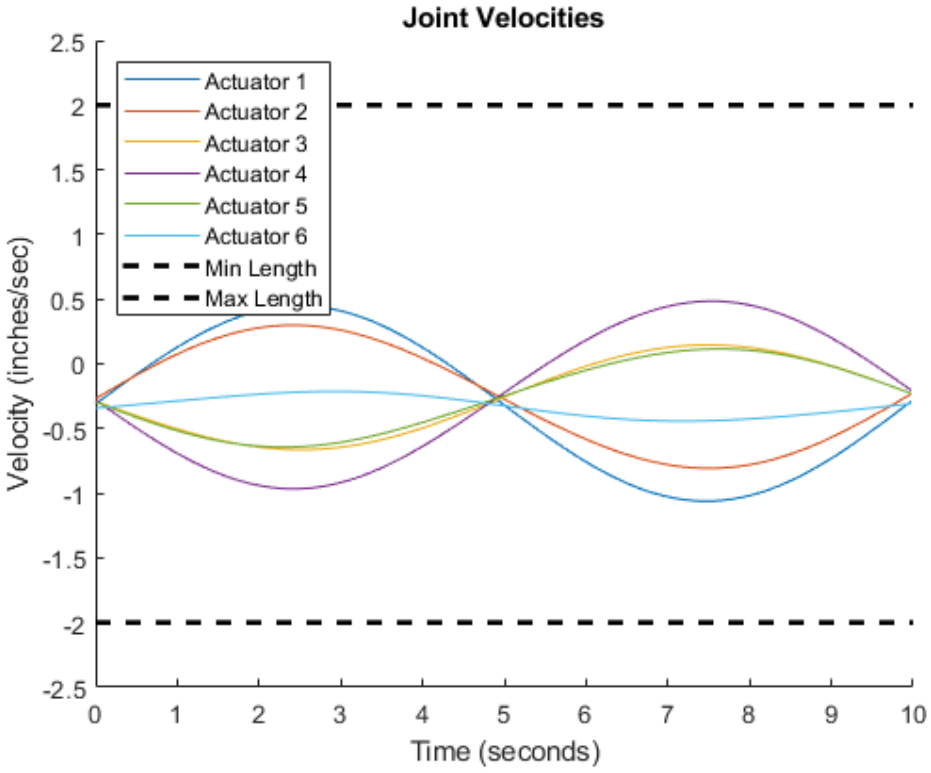
Plots

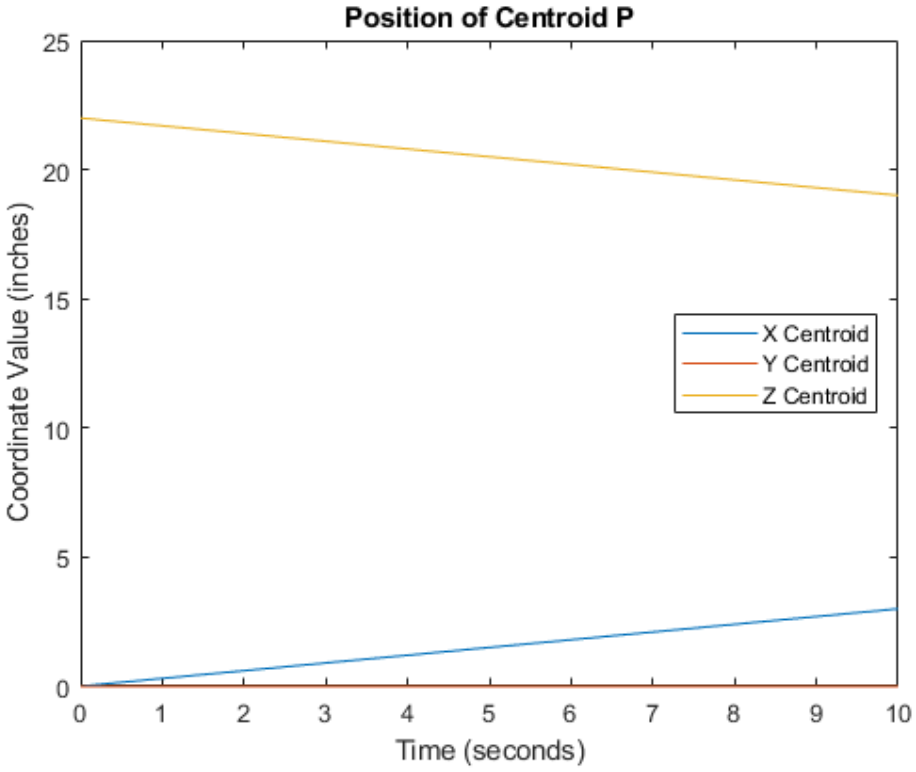
```
if plots
    figure(1)
    hold on
    for i=1:6
        plot(time,L_length(:,i))
```

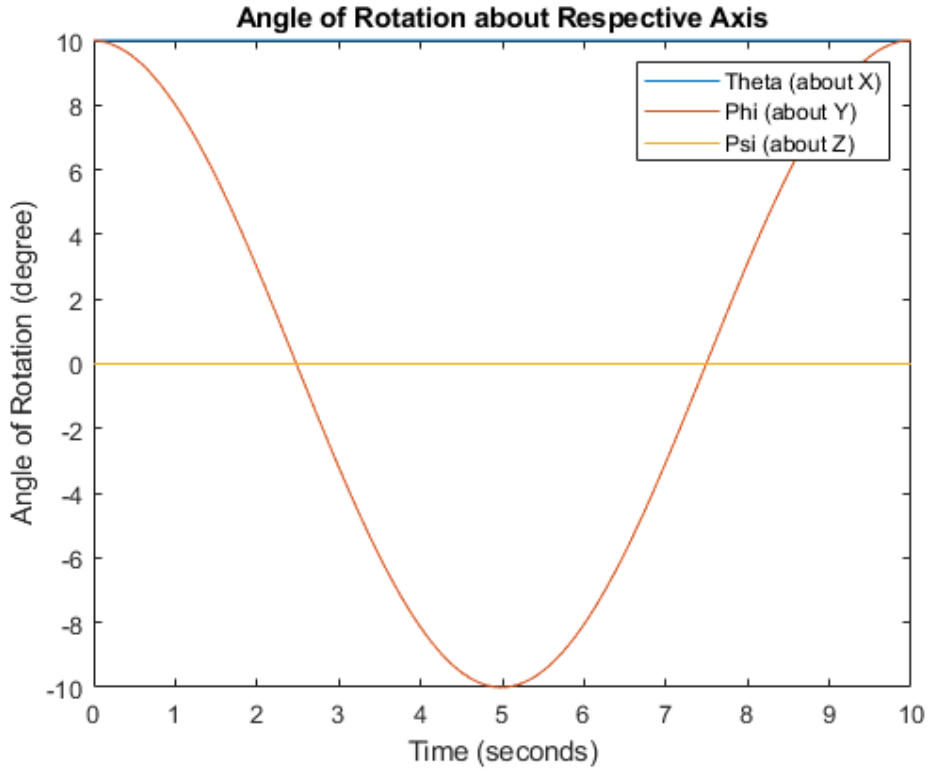
```
    end

    xlabel('Time (seconds)')
    ylabel('Length (inches)')
    plot(time, min_joint_length*ones(1,length(time)),'--k', time,
max_joint_length*ones(1,length(time)),'--k','LineWidth',2)
    legend('Actuator 1', 'Actuator 2', 'Actuator 3', 'Actuator
4', 'Actuator 5', 'Actuator 6', 'Min Length', 'Max
Length', 'Location', 'north')
    title('Joint Lengths, from Sphere to Sphere')

    hold off

    figure(2)
    hold on
    for i=1:6
        plot(time_dot,L_length_dot(:,i))
    end
    plot(time, min_joint_velocity*ones(1,length(time)),'--k', time,
max_joint_velocity*ones(1,length(time)),'--k', 'LineWidth',2)
    xlabel('Time (seconds)')
    ylabel('Velocity (inches/sec)')
    ylim([-2.5 2.5])
    % plot(time, min_joint_length*ones(1,length(time_dot)), time,
max_joint_length*ones(1,length(time)))
    legend('Actuator 1', 'Actuator 2', 'Actuator 3', 'Actuator
4', 'Actuator 5', 'Actuator 6', 'Min Length', 'Max
Length', 'Location', 'northwest')
    title('Joint Velocities')

    hold off

    figure(3)
    plot(time, Px, time, Py, time, Pz)
    xlabel('Time (seconds)')
    ylabel('Coordinate Value (inches)')
    legend('X Centroid', 'Y Centroid', 'Z
Centroid', 'Location', 'east')
    title('Position of Centroid P')

    figure(4)
    plot(time, theta, time, phi, time, psi)
    xlabel('Time (seconds)')
    ylabel('Angle of Rotation (degree)')
    legend('Theta (about X)', 'Phi (about Y)', 'Psi (about Z)')
    title('Angle of Rotation about Respective Axis')

    figure(5)
    plot(time_dot, Px_dot, time_dot, Py_dot, time_dot, Pz_dot,
time_dot, P_dot)
    xlabel('Time (seconds)')
    ylabel('Coordinate Value Velocity (inches/sec)')
    legend('X Centroid', 'Y Centroid', 'Z Centroid', 'Centroid
Velocity', 'Location', 'southeast')
```

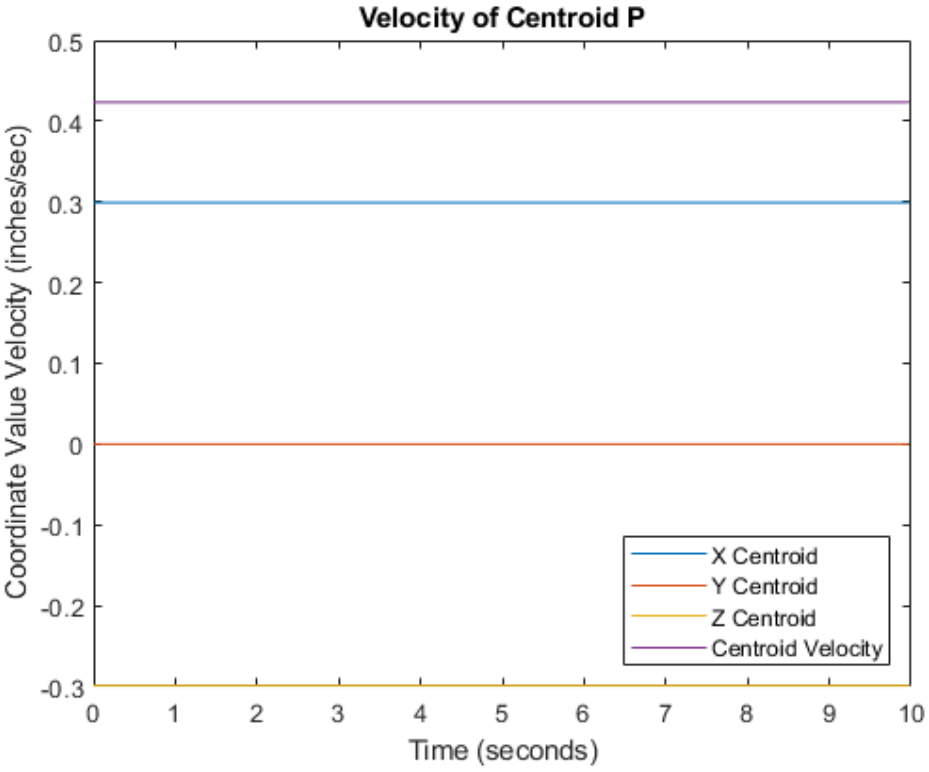
```
    title('Velocity of Centroid P')

    figure(6)
    plot(time_dot, theta_dot, time_dot, phi_dot, time_dot, psi_dot)
    xlabel('Time (seconds)')
    ylabel('Angular Velocity (degree/sec)')
    legend('Theta Dot (about X)', 'Phi Dot (about Y)', 'Psi Dot (about
 Z)', 'Location', 'north')
    title('Angular Velocity about Respective Axis')
end
```

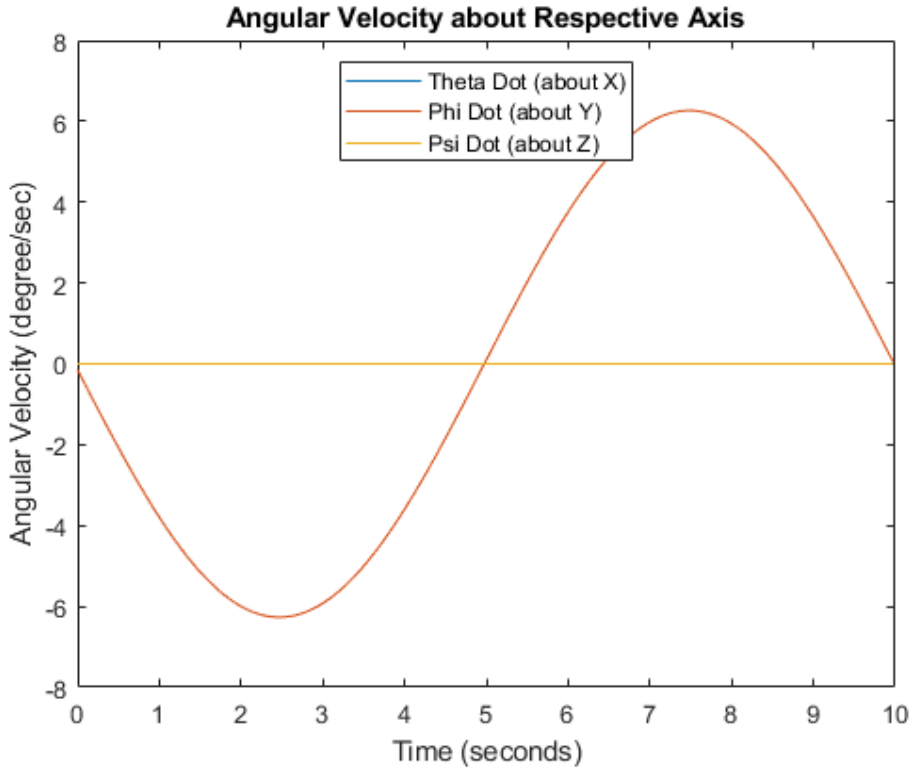



Joint Velocities
Actuator 1
Actuator 2
Actuator 3
Actuator 4
Actuator 5
Actuator 6
Min Length
Max Length
Velocity (inches/sec)
Time (seconds)


Position of Centroid P
X Centroid
Y Centroid
Z Centroid
Coordinate Value (inches)
Time (seconds)


Angle of Rotation about Respective Axis
Theta (about X)
Phi (about Y)
Psi (about Z)
Angle of Rotation (degree)
Time (seconds)
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
1
2
3
4
5
6
7
8
9
10


Velocity of Centroid P
X Centroid
Y Centroid
Z Centroid
Centroid Velocity
Coordinate Value Velocity (inches/sec)
Time (seconds)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
1
2
3
4
5
6
7
8
9
10



Animation

```
if animation
    figure(100)

    alpha_base = 0:10:360; %creates angles for circular base plot
    x_base = base_diameter*0.5*cosd(alpha_base); %x values for
 circular base
    y_base = base_diameter*0.5*sind(alpha_base); %y values for
 circular base
    z_base = zeros(length(x_base)); %z values for circular base

    if configuration_6_3 == 0 && configuration_6_6 == 0
        alpha = base_angle;
    else
        alpha = 0:60:360;
    end

    x = a*cosd(alpha); %x values for actuators base side
    y = a*sind(alpha); %y values for actuators base side
    z = zeros(length(x));
    z1 = ones(length(x));

    for j = 1:length(psi)
    hold off
```

```
    plot3(x_base, y_base, z_base, 'b-', 'LineWidth', 3); %plot
 circular base every time
    hold on
        for i = 1:6
    %         for each instance in time, plot the xyz values of the
 end of the
    %         actuators based on the length
            x_d(i) = x(i) + L_vector(1,i,j);
            y_d(i) = y(i) + L_vector(2,i,j);
            z_d(i) = z(i) + L_vector(3,i,j);
            plot3([x(i) x_d(i)], [y(i) y_d(i)], [z(i)
 z_d(i)], 'LineWidth', 3)
        end

    %     periodic boundary condition to complete top plate drawing
        x_d(i+1)=x_d(1);
        y_d(i+1)=y_d(1);
        z_d(i+1)=z_d(1);
        plot3(x_d, y_d, z_d,'LineWidth', 3)

    %     plot settings
        axis square;
        axis([-(1.5*a) (1.5*a) -(1.5*a) 1.5*a 0 (max(Pz)+b/2)]);
        grid on;
        set(gcf,'WindowState','fullscreen')
        pause(1/100)
        if j < length(psi)
            clf;
        end
    end
end
```

Published with MATLAB® R2018b

```
function b_rot = stewartrot(theta, phi, psi, b_global)
    %This function applies three rotation matrices to determine the
 value
    %of b in the rotated reference frame
    rotation_matrix = [cosd(psi)*cosd(phi),
 cosd(psi)*sind(phi)*sind(theta)-sind(psi)*cosd(theta),
 cosd(psi)*sind(phi)*cosd(theta)+sind(psi)*sind(theta), 0;...
                        sind(psi)*cosd(phi),
 sind(psi)*sind(phi)*sind(theta)+cosd(psi)*cosd(theta),
 sind(psi)*sind(phi)*cosd(theta)-cosd(psi)*sind(theta), 0;...
                        -sind(phi), cosd(phi)*sind(theta),
 cosd(phi)*cosd(theta), 0;...
                        0, 0, 0, 1];
    b_rot = rotation_matrix*b_global;
```

```
Not enough input arguments.

Error in stewartrot (line 4)
    rotation_matrix = [cosd(psi)*cosd(phi),
 cosd(psi)*sind(phi)*sind(theta)-sind(psi)*cosd(theta),
 cosd(psi)*sind(phi)*cosd(theta)+sind(psi)*sind(theta), 0;...
```

Published with MATLAB® R2018b

Appendix J

ARDUINO SKETCH


```
#include <platform.h>

// Main Arduino code for a 6-dof Stewart platform.
// Written for the Arduino Due.
// This code has been taken from the ENPH 459 Engineering Physics group from the University of British Columbia
// and modified by Trent Peterson to work with the Cal Poly ME 423 Stewart Platform robot and the Stewart Platform Motor Driver PCB hat.

// Actuator variables
uint8_t pwm[NUM_MOTORS];            // current PWM for each actuator
MotorDirection dir[NUM_MOTORS];     // current direction for each actuator (EXTEND or RETRACT)
uint8_t speedscale;                 // Input speed from Host PC
uint8_t percentage = 100;           // For turning speed into fraction

// Position variables
int16_t pos[NUM_MOTORS];            // current position (measured by analog read) of each actuator
int16_t movestartpos[NUM_MOTORS];   // latch the start position of a move when serial input is read
int16_t input[NUM_MOTORS];          // intermediate input retrieved from the SerialUSB buffer
uint16_t desired_pos[NUM_MOTORS];   // desired (user-inputted and validated) position of each actuator
uint16_t deltaMax = 0;              // Determine the largest actuator distance to travel for a move
// Feedback variables
int16_t pos_diff;                   // difference between current and desired position
int16_t static_pos_diff;                  // difference between latched and desired position
int16_t prop_diff;                  // position difference used for proportional gain
int16_t previous_diff[NUM_MOTORS];  // last position difference for each actuator; for derivative gain
//int16_t total_diff[NUM_MOTORS];     // cumulative position difference for each actuator; for integral gain
//uint8_t previous_inst[NUM_MOTORS];  // starting/past sample instance for each actuator; for derivative gain
//uint8_t current_inst[NUM_MOTORS];   // new/incrementing sample instance for each actuator; for derivative gain
float p_corr;                       // proportional correction for PID
float i_corr;                       // integral correction for PID
float d_corr;                       // differential correction for PID
float corr;                         // final feedback PWM value from PID correction
bool stop_threshold;
bool stop_array[NUM_MOTORS];
bool printonce;

// Calibration variables
int16_t end_readings[NUM_MOTORS];
int16_t zero_readings[NUM_MOTORS];
bool calibration_valid;

// Time variables (for printing)
unsigned long current_time;         // current time (in millis); used to measure difference from previous_time
unsigned long previous_time;        // last recorded time (in millis); measured from execution start or last print

// Iterator/sum variables
uint8_t motor;                      // used to iterate through actuators by their indexing (0 to NUM_MOTORS - 1)
uint8_t reading;                    // used to iterate through analog reads (0 to NUM_READINGS - 1)
int32_t reading_sum;                // sum of multiple readings to be averaged for a final value


/*
   Runs once at initialization; set up input and output pins and variables.
*/
void setup()
{
  while (!SerialUSB);

  // Initialize pins
  for (motor = 0; motor < NUM_MOTORS; ++motor)
  {
    pinMode(DIR_PINS[motor], OUTPUT);
    digitalWrite(DIR_PINS[motor], LOW);

    pinMode(PWM_PINS[motor], OUTPUT);
    analogWrite(PWM_PINS[motor], 0);

    pinMode(POT_PINS[motor], INPUT);
  }

  // Initialize actuator enable switches
  pinMode(DISABLE_MOTORS, OUTPUT);
  digitalWrite(DISABLE_MOTORS, LOW);

  // For safety, set initial actuator settings and speed to 0
  for (motor = 0; motor < NUM_MOTORS; ++motor)
  {
    //total_diff[motor] = 0;
    desired_pos[motor] = 0;
    pwm[motor] = MIN_PWM;
  }

  // Initialize SerialUSB communication
  SerialUSB.begin(BAUD);

  // Run calibration
```

```
    calibration_valid = true;
    calibrate();

    // Initialize the current print interval
  #if ENABLE_PRINT
    {
      previous_time = 0;
    }
  #endif // ENABLE_PRINT
  }


  /*
     Run tasks in an infinite loop.
  */
  void loop()
  {
    // Read current actuator positions, map them based on actuator-based scaling from calibration
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      pos[motor] = map(getAverageReading(motor), ZERO_POS[motor], END_POS[motor], MIN_POS, MAX_POS);
    }

    // Parse new SerialUSB input if enough is in the buffer (also sets desired position if input is valid)
    if (SerialUSB.available() >= INPUT_TRIGGER)
    {
      readSerialUSB();
    }

    // Print actuator information at a given interval
  #if ENABLE_PRINT
    {
      printOutput();
    }
  #endif // ENABLE_PRINT

    // For each actuator, set movement parameters (direction, PWM) and execute motion
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      move(motor);
    }
    if (stop_array[1] && stop_array[2] && stop_array[3] && stop_array[4] && stop_array[5] && stop_array[6] && printonce)
    {
      SerialUSB.write(0x44);
      printonce= false;
    }
  }


  /*
     Take NUM_READINGS readings of a potentiometer pin and returns the average.
  */
  int getAverageReading(uint8_t motor)
  {
    reading_sum = 0;
    for (reading = 0; reading < NUM_READINGS; ++reading)
    {
      reading_sum += analogRead(POT_PINS[motor]);
    }
    return reading_sum / NUM_READINGS;
  }

  /*
     Move all actuators in a given direction at full PWM.
  */
  inline void moveAll(MotorDirection dir)
  {
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      digitalWrite(DIR_PINS[motor], dir);
      analogWrite(PWM_PINS[motor], MAX_PWM);
    }
  }


  /*
     Read and parse SerialUSB input into actuator positions.
  */
  inline void readSerialUSB()
  {
    printonce= true;
    // Parse ints from SerialUSB
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      input[motor] = SerialUSB.parseInt();
    }
```

```
    speedscale = SerialUSB.parseInt();

    // Check that inputs are valid
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      if (input[motor] < MIN_POS || input[motor] > MAX_POS)
      {
        return;
      }
    }

    // Set the input as the desired positions
    deltaMax = 0;

    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {

      stop_array[motor] = 0;  // reset move complete flag
      desired_pos[motor] = input[motor];
      movestartpos[motor] = pos[motor];

      if (abs(movestartpos[motor] - desired_pos[motor]) > deltaMax)
      {
          deltaMax = abs(movestartpos[motor] - desired_pos[motor]);
      }
    }
  }


  /*
     Print actuator variables over SerialUSB at a given interval.
  */
  inline void printOutput()
  {
    current_time = millis();
    //if (current_time - previous_time > PRINT_INTERVAL)
    if (current_time - previous_time > 100)

    {
      // Print instance header
  #if ENABLE_PRINT_HEADERS
      {
        SerialUSB.print("<COM>\n");
      }
  #endif // ENABLE_PRINT_HEADERS

      // Print desired position
  #if PRINT_DESIRED_POS
      {
  #if ENABLE_PRINT_HEADERS
        {
          SerialUSB.print("      DESIRED POS: ");
        }
  #endif // ENABLE_PRINT_HEADERS

        for (motor = 0; motor < NUM_MOTORS; ++motor)
        {
          SerialUSB.print(desired_pos[motor]);
          SerialUSB.print(" ");
        }

        SerialUSB.print("\n");
      }
  #endif // PRINT_DESIRED_POS

      // Print current position
  #if PRINT_CURRENT_POS
      {
  #if ENABLE_PRINT_HEADERS
        {
          SerialUSB.print("      CURRENT POS: ");
        }
  #endif // ENABLE_PRINT_HEADERS

        for (motor = 0; motor < NUM_MOTORS; ++motor)
        {
          SerialUSB.print(pos[motor]);
          SerialUSB.print(" ");
        }

        SerialUSB.print("\n");
      }
  #endif // PRINT_CURRENT_POS

      // Print PWM values
  #if PRINT_PWM
```

```
    {
#if ENABLE_PRINT_HEADERS
      {
        SerialUSB.print("      PWM        : ");  // align values with other entries
      }
#endif // ENABLE_PRINT_HEADERS

      for (motor = 0; motor < NUM_MOTORS; ++motor)
      {
        SerialUSB.print(pwm[motor]);
        SerialUSB.print(" ");
      }

      SerialUSB.print("\n");
    }
#endif // PRINT_PWM

    // Print PWM values
#if PRINT_DIR
    {
#if ENABLE_PRINT_HEADERS
      {
        SerialUSB.print("      DIR        : ");  // align values wih other entries
      }
#endif // ENABLE_PRINT_HEADERS

      for (motor = 0; motor < NUM_MOTORS; ++motor)
      {
        SerialUSB.print(dir[motor]);
        SerialUSB.print(" ");
      }

      SerialUSB.print("\n");
    }
#endif // PRINT_DIR
    //--------------------------------------------------------------------------------------------

    // Print CR to end message block
#if ENABLE_PRINT_HEADERS
    {
      SerialUSB.print("\r");
    }
#endif // ENABLE_PRINT_HEADERS

    previous_time = current_time;
  }
}


/*
   Compute the PID values given the current actuator position, and use them
   to move the actuator with new direction/PWM values.
*/
inline void move(uint8_t motor)
{
  // Compute the error value
  pos_diff = pos[motor] - desired_pos[motor];

  // Compute error-dependent PID variables
  if (abs(pos_diff) <= POS_THRESHOLD[motor])
  {
    stop_array[motor] = true;
  }
  else
  {
    stop_threshold = false;
  }
  static_pos_diff = movestartpos[motor] - desired_pos[motor];

  // Compute the correction value and set the direction and PWM for the actuator
  corr = MAX_PWM * static_pos_diff / deltaMax;
  dir[motor] = (corr > 0) ? RETRACT : EXTEND;  // direction based on the sign of the error
  pwm[motor] = (constrain(abs(corr), MIN_PWM, MAX_PWM)*speedscale)/percentage;  // bound the correction by the PWM limits

  if (pwm[motor] < 30)
  {
      pwm[motor]=30;
  }

  if (stop_array[motor])
  {
    pwm[motor]=0;
  }
  // Move the actuator with the new direction and PWM values
  digitalWrite(DIR_PINS[motor], dir[motor]);
  analogWrite(PWM_PINS[motor], pwm[motor]);
```

```
}


/**
   Calibration routine run on startup.
   Extends all actuators, reads extended values, then retracts all, and reads their retracted values.
*/
inline void calibrate()
{
  //SerialUSB.print("<COM> Beginning calibration.\n");
  //delay(500);

  // Extend all actuators
  for (motor = 0; motor < NUM_MOTORS; ++motor)
  {
    // Start with extension
    digitalWrite(DIR_PINS[motor], EXTEND);
    analogWrite(PWM_PINS[motor], MAX_PWM);
  }
  delay(RESET_DELAY);

  // Stop the extension, get averaged analog readings
  for (motor = 0; motor < NUM_MOTORS; ++motor)
  {
    analogWrite(PWM_PINS[motor], 0);
    end_readings[motor] = getAverageReading(motor);

    // Check if the motors are powered (reading is valid)
    calibration_valid = (abs(end_readings[motor] - END_POS[motor]) < OFF_THRESHOLD);
    if (!calibration_valid)
    {
      break;
    }
  }
  // Retract all actuators
  for (motor = 0; motor < NUM_MOTORS; ++motor)
  {
    digitalWrite(DIR_PINS[motor], RETRACT);
    analogWrite(PWM_PINS[motor], MAX_PWM);
  }
  delay(RESET_DELAY);

  // Stop the retraction, get averaged analog readings
  if (calibration_valid)
  {
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      analogWrite(PWM_PINS[motor], 0);
      zero_readings[motor] = getAverageReading(motor);

      // Check if the motors are powered (reading is valid)
      calibration_valid = (abs(zero_readings[motor] - ZERO_POS[motor]) < OFF_THRESHOLD);
      if (!calibration_valid)
      {
        break;
      }
    }
  }

  // Set the new calibration values if found to be valid
  if (calibration_valid)
  {
    for (motor = 0; motor < NUM_MOTORS; ++motor)
    {
      END_POS[motor] = end_readings[motor];
      ZERO_POS[motor] = zero_readings[motor];
    }
  }

  // Print the calibration result (new max/min values, or a warning)
#if ENABLE_PRINT_HEADERS
  {
    if (calibration_valid)
    {
      SerialUSB.print("<COM> Finished calibration:\n");


      // Print minimum positions
      SerialUSB.print("      MIN POS: ");
      for (motor = 0; motor < NUM_MOTORS; ++motor)
      {
        SerialUSB.print(ZERO_POS[motor]);
        SerialUSB.print(" ");
      }
      SerialUSB.print("\n");
```

```
      // Print maximum positions
      SerialUSB.print("      MAX POS: ");
      for (motor = 0; motor < NUM_MOTORS; ++motor)
      {
        SerialUSB.print(END_POS[motor]);
        SerialUSB.print(" ");
      }
      SerialUSB.print("\n");

      // Print CR to end message block
      SerialUSB.print("\r");
    }

    else
    {
      // Print an error message for calibration failure
      SerialUSB.print("<COM> Failed calibration (using default values).\n");
      SerialUSB.print("      Please verify that the actuators are powered on.\n\r");
    }
  }
#endif // ENABLE_PRINT_HEADERS

  if (calibration_valid)
  {
    SerialUSB.write(1);
  }
  else
  {
    SerialUSB.write((uint8_t) 0);
  }

}
```

Appendix K

MATLAB GUI CODE

```
classdef StewartPlatformApp_exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        %These properties (GUI object creation) were omitted for
 brevity
    end

properties (Access = private)
    ser;
    Actuator_Current_Pos = [0;0;0;0;0;0];
    serial_state;
    P1=zeros(1,6);
    P2=zeros(1,6);
    P3=zeros(1,6);
    P4=zeros(1,6);
    P5=zeros(1,6);
    P6=zeros(1,6);
    MoveComplete;

end


methods (Access = private)

    function MoveToPoint(app,P)

        speed = round(app.SetPlatformSpeedSlider.Value);



        if app.PathPlanningCheckBox.Value == 1
            j=1;
            printOnce = true;
            move_points = app.PointsEditField.Value;
            pos_array = zeros(length(P), move_points);
            for i = 1:length(P)
                pos_array(i,:) =
 round(linspace(app.Actuator_Current_Pos(i),P(i),move_points));
            end

            if app.serial_state == 1
                while j <= move_points
                    if (printOnce == true)
                       fprintf(app.ser,
 [strjoin({num2str(pos_array(1,j)), num2str(pos_array(2,j)),
 num2str(pos_array(3,j)), num2str(pos_array(4,j)),
 num2str(pos_array(5,j)), num2str(pos_array(6,j)),
 num2str(speed)}), '\n']);
                       printOnce = false;
                    end
                    MoveCheck = fread(app.ser,1);
```

```
                    if MoveCheck == 'D'
                        j = j +1;
                        printOnce = true; %intermediate move is done,
 send the next position
                    end
                end
                app.ErrorMessage.Value = 'Move Complete';
            end
        end
        if app.PathPlanningCheckBox.Value == 0 && app.serial_state ==
 1
            fprintf(app.ser, [strjoin({num2str(P(1)), num2str(P(2)),
 num2str(P(3)), num2str(P(4)), num2str(P(5)), num2str(P(6)),
 num2str(speed)}), '\n']);
            MoveCheck = fread(app.ser,1);
            if MoveCheck == 'D'
                app.ErrorMessage.Value = 'Move Complete';
            end
        end

        app.ErrorMessage.Visible = 'on';
        for k = 1:length(P)
            app.Actuator_Current_Pos(k) = P(1,k);
        end
    end
end

    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            app.ErrorMessage.Visible = 'on';
            app.SerialSwitch.Enable = 'off';
            app.TopJointLocations63Panel.Visible = 'off';
            app.TopJointLocations66Panel.Visible = 'on';
%             app.t=timer();
%             app.t.Period = 1;
%             app.t.ExecutionMode = 'fixedRate';
%             app.t.TimerFcn = @


        end

        % Selection changed function: StewartPlatformGUIButtonGroup
        function StewartPlatformGUIButtonGroupSelectionChanged(app,
 event)

            if(app.GenericConfigurationButton.Value)
                    app.TopJointLocations63Panel.Visible = 'off';
                    app.TopJointLocations66Panel.Visible = 'on';
                    app.BaseJointLocationsPanel.Position =
 [15,255,197,171];
            end
            if(app.ConfigurationButton_2.Value)
```

```
                    app.TopJointLocations63Panel.Visible = 'on';
                    app.TopJointLocations66Panel.Visible = 'off';
                    app.BaseJointLocationsPanel.Position =
[15,324,197,171];
                end

            end

            % Button pushed function: CalibrationButton
            function CalibrationButtonPushed(app, event)

                app.CalcJointButton.Enable = 'off';
                app.ErrorMessage.Value = 'Calibrating...';
```

Serial configuration

```
                ComPortNumber   = num2str(app.COMPortEditField.Value);
                PORT            = strcat('COM', ComPortNumber);
                BAUD_RATE       = 115200;
                DATA_BITS       = 8;
                PARITY          = 'none';
                STOP_BITS       = 1;

                % Establish a new serial connection (clear previous
connections)
                delete(instrfindall);
                app.ser = serial(      PORT,        ...
                 'BaudRate',    BAUD_RATE,   ...
                 'DataBits',    DATA_BITS,   ...
                 'Parity',      PARITY,      ...
                 'StopBits',    STOP_BITS    ...
                );
                fopen(app.ser);
                app.SerialSwitch.Value = 'On';
                app.serial_state = 1;
                pause(14);
                CalibrationState = fread(app.ser,1);
                if CalibrationState == 1
                    app.CalibrationButton.Text = 'Calibrated';
                    app.CalibrationButton.Enable = 'off';
                    app.CalcJointButton.Enable = 'on';
                    app.ErrorMessage.Value = '';
                    app.SerialSwitch.Enable = 'off';
                    app.MovePlatformReset.Enable = 'on';
                elseif CalibrationState == 0
                    app.ErrorMessage.Value = 'Calibration Failed';
                else
                    app.ErrorMessage.Value = 'Serial Compare Failed';
                end

            end

            % Button pushed function: CalcJointButton
```

```
        function CalcJointButtonPushed(app, event)
            app.ErrorMessage.Value = '';
            app.ErrorMessage.BackgroundColor = 'w';
            % SET UP VARIABLES

 %-------------------------------------------------------------
            a_length = 17/2; %Length [inches] of a vector on base
            b_length = 13/2; %Length [inches] of a vector on top
            ball_joint_height = 1.032; %Height from magnetic base to
sphere center [inches]
            ball_diameter = 18/25.4; %[inches]
            platform_thiccness = 0.25; %[inches]
            joint_joint_offset = 15.61; %[inches]
            %need to verify both dimensions
            angleOffset_6_3 =3.5; %[degrees]
            a_i = zeros(4,6);
            b_i = zeros(4,6);
            b_rot = zeros(4,6);
            d_vector = zeros(4,6);
            d_length = zeros(1,6);

            %Take in

            Px = app.PxInput.Value;
            Py = app.PyInput.Value;
            Pz = app.PzInput.Value;

            %Remove offset from base top to ball center
            Pz = Pz - ball_joint_height - ball_diameter/2 -
platform_thiccness;

            P = [Px;Py;Pz;1];

            % SET UP A and B Matrices

 %-------------------------------------------------------------

            a_i(1,1) = a_length*cosd(app.Base_Act_1.Value); a_i(2,1) =
a_length*sind(app.Base_Act_1.Value);
            a_i(1,2) = a_length*cosd(app.Base_Act_2.Value); a_i(2,2) =
a_length*sind(app.Base_Act_2.Value);
            a_i(1,3) = a_length*cosd(app.Base_Act_3.Value); a_i(2,3) =
a_length*sind(app.Base_Act_3.Value);
            a_i(1,4) = a_length*cosd(app.Base_Act_4.Value); a_i(2,4) =
a_length*sind(app.Base_Act_4.Value);
            a_i(1,5) = a_length*cosd(app.Base_Act_5.Value); a_i(2,5) =
a_length*sind(app.Base_Act_5.Value);
            a_i(1,6) = a_length*cosd(app.Base_Act_6.Value); a_i(2,6) =
a_length*sind(app.Base_Act_6.Value);
            a_i(3,:) = 0;
            a_i(4,:) = 1;

            if(app.GenericConfigurationButton.Value)
```

```
                b_i(1,1) = b_length*cosd(app.Top_Act_1.Value);
b_i(2,1) = b_length*sind(app.Top_Act_1.Value);
                b_i(1,2) = b_length*cosd(app.Top_Act_2.Value);
b_i(2,2) = b_length*sind(app.Top_Act_2.Value);
                b_i(1,3) = b_length*cosd(app.Top_Act_3.Value);
b_i(2,3) = b_length*sind(app.Top_Act_3.Value);
                b_i(1,4) = b_length*cosd(app.Top_Act_4.Value);
b_i(2,4) = b_length*sind(app.Top_Act_4.Value);
                b_i(1,5) = b_length*cosd(app.Top_Act_5.Value);
b_i(2,5) = b_length*sind(app.Top_Act_5.Value);
                b_i(1,6) = b_length*cosd(app.Top_Act_6.Value);
b_i(2,6) = b_length*sind(app.Top_Act_6.Value);

            end

            if(app.ConfigurationButton_2.Value)

                b_i(1,1) = b_length*cosd(app.Top_Act_1_2.Value -
angleOffset_6_3); b_i(2,1) = b_length*sind(app.Top_Act_1_2.Value -
angleOffset_6_3);
                b_i(1,2) = b_length*cosd(app.Top_Act_1_2.Value +
angleOffset_6_3); b_i(2,2) = b_length*sind(app.Top_Act_1_2.Value +
angleOffset_6_3);
                b_i(1,3) = b_length*cosd(app.Top_Act_3_4.Value -
angleOffset_6_3); b_i(2,3) = b_length*sind(app.Top_Act_3_4.Value -
angleOffset_6_3);
                b_i(1,4) = b_length*cosd(app.Top_Act_3_4.Value +
angleOffset_6_3); b_i(2,4) = b_length*sind(app.Top_Act_3_4.Value +
angleOffset_6_3);
                b_i(1,5) = b_length*cosd(app.Top_Act_5_6.Value -
angleOffset_6_3); b_i(2,5) = b_length*sind(app.Top_Act_5_6.Value -
angleOffset_6_3);
                b_i(1,6) = b_length*cosd(app.Top_Act_5_6.Value +
angleOffset_6_3); b_i(2,6) = b_length*sind(app.Top_Act_5_6.Value +
angleOffset_6_3);

            end

            b_i(3,:) = 0;
            b_i(4,:) = 1;

            % Rotate the B matrix and calculate the lengths of
 actuators
            for i = 1:6
                b_rot(:,i)=stewartrot(-app.ThetaInput.Value, -
app.PhiInput.Value, -app.PsiInput.Value, b_i(:,i));
                d_vector(:,i)  = P(:,1) + b_rot(:,i) - a_i(:,i);
                d_length(1,i) = sqrt(d_vector(1,i)^2 + d_vector(2,i)^2
 + d_vector(3,i)^2);
            end

            %Remove physical constants so length is on scale from 0
 inches to 8 inches
            d_length = d_length - joint_joint_offset;
```

```
            app.Actuator_1_Length_Set.Value = d_length(1,1);
            app.Actuator_2_Length_Set.Value = d_length(1,2);
            app.Actuator_3_Length_Set.Value = d_length(1,3);
            app.Actuator_4_Length_Set.Value = d_length(1,4);
            app.Actuator_5_Length_Set.Value = d_length(1,5);
            app.Actuator_6_Length_Set.Value = d_length(1,6);

            if max(d_length) > 8 || min(d_length) < 0
                app.ErrorMessage.Visible = 'on';
                app.MovePlatform.Enable = 'off';
                app.ErrorMessage.Value = 'CALCULATED LENGTH(S) OUT OF
 RANGE';
                app.ErrorMessage.BackgroundColor = 'r';
                app.ErrorMessage.FontWeight = 'bold';
            else
                app.MovePlatform.Enable = 'on';
                app.SaveP1Button.Enable = 'on';
                app.SaveP2Button.Enable = 'on';
                app.SaveP3Button.Enable = 'on';
                app.SaveP4Button.Enable = 'on';
                app.SaveP5Button.Enable = 'on';
                app.SaveP6Button.Enable = 'on';
%                 app.ErrorMessage.Visible = 'off';
            end

            output_length = round(interp1([-8 16],[-1024
 2048],d_length));
            app.Actuator_1_Pot_Set.Value = output_length(1,1);
            app.Actuator_2_Pot_Set.Value = output_length(1,2);
            app.Actuator_3_Pot_Set.Value = output_length(1,3);
            app.Actuator_4_Pot_Set.Value = output_length(1,4);
            app.Actuator_5_Pot_Set.Value = output_length(1,5);
            app.Actuator_6_Pot_Set.Value = output_length(1,6);

        end

        % Button pushed function: MovePlatform
        function MovePlatformButtonPushed(app, event)
            output_length(1) = app.Actuator_1_Pot_Set.Value;
            output_length(2) = app.Actuator_2_Pot_Set.Value;
            output_length(3) = app.Actuator_3_Pot_Set.Value;
            output_length(4) = app.Actuator_4_Pot_Set.Value;
            output_length(5) = app.Actuator_5_Pot_Set.Value;
            output_length(6) = app.Actuator_6_Pot_Set.Value;


            MoveToPoint(app, output_length)
        end

        % Value changed function: SerialSwitch
        function SerialSwitchValueChanged(app, event)
            %value = app.SerialSwitch.Value;
            switch app.SerialSwitch.Value
```

```
            case 'On'
            fopen(app.ser);
            app.serial_state = 1;
            case 'Off'
            app.serial_state = 0;
            fclose(app.ser);
        end

    end

    % Button pushed function: Logo_3
    function Logo_3ButtonPushed(app, event)
        web('https://www.calpoly.edu/');
    end

    % Button pushed function: Logo_2
    function Logo_2ButtonPushed(app, event)
        web('https://me.calpoly.edu/');
    end

    % Button pushed function: Logo_4
    function Logo_4ButtonPushed(app, event)
        web('http://heli-cal.com/');
    end

    % Button pushed function: Logo
    function LogoButtonPushed(app, event)
        web('https://www.progressiveautomations.com/');
    end

    % Button pushed function: SaveP1Button
    function SaveP1ButtonPushed(app, event)
        app.P1(1) = app.Actuator_1_Pot_Set.Value;
        app.P1(2) = app.Actuator_2_Pot_Set.Value;
        app.P1(3) = app.Actuator_3_Pot_Set.Value;
        app.P1(4) = app.Actuator_4_Pot_Set.Value;
        app.P1(5) = app.Actuator_5_Pot_Set.Value;
        app.P1(6) = app.Actuator_6_Pot_Set.Value;

        app.P1CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP1Button.Enable = 'on';
        app.MovetoP1Button.Enable = 'on';
    end

    % Button pushed function: SaveP2Button
    function SaveP2ButtonPushed(app, event)
        app.P2(1) = app.Actuator_1_Pot_Set.Value;
        app.P2(2) = app.Actuator_2_Pot_Set.Value;
        app.P2(3) = app.Actuator_3_Pot_Set.Value;
        app.P2(4) = app.Actuator_4_Pot_Set.Value;
        app.P2(5) = app.Actuator_5_Pot_Set.Value;
        app.P2(6) = app.Actuator_6_Pot_Set.Value;
```

```
        app.P2CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP2Button.Enable = 'on';
        app.MovetoP2Button.Enable = 'on';
    end

    % Button pushed function: SaveP3Button
    function SaveP3ButtonPushed(app, event)
        app.P3(1) = app.Actuator_1_Pot_Set.Value;
        app.P3(2) = app.Actuator_2_Pot_Set.Value;
        app.P3(3) = app.Actuator_3_Pot_Set.Value;
        app.P3(4) = app.Actuator_4_Pot_Set.Value;
        app.P3(5) = app.Actuator_5_Pot_Set.Value;
        app.P3(6) = app.Actuator_6_Pot_Set.Value;

        app.P3CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP3Button.Enable = 'on';
        app.MovetoP3Button.Enable = 'on';
    end

    % Button pushed function: SaveP4Button
    function SaveP4ButtonPushed(app, event)
        app.P4(1) = app.Actuator_1_Pot_Set.Value;
        app.P4(2) = app.Actuator_2_Pot_Set.Value;
        app.P4(3) = app.Actuator_3_Pot_Set.Value;
        app.P4(4) = app.Actuator_4_Pot_Set.Value;
        app.P4(5) = app.Actuator_5_Pot_Set.Value;
        app.P4(6) = app.Actuator_6_Pot_Set.Value;

        app.P4CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP4Button.Enable = 'on';
        app.MovetoP4Button.Enable = 'on';
    end

    % Button pushed function: SaveP5Button
    function SaveP5ButtonPushed(app, event)
        app.P5(1) = app.Actuator_1_Pot_Set.Value;
        app.P5(2) = app.Actuator_2_Pot_Set.Value;
        app.P5(3) = app.Actuator_3_Pot_Set.Value;
        app.P5(4) = app.Actuator_4_Pot_Set.Value;
        app.P5(5) = app.Actuator_5_Pot_Set.Value;
        app.P5(6) = app.Actuator_6_Pot_Set.Value;

        app.P5CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP5Button.Enable = 'on';
        app.MovetoP5Button.Enable = 'on';
    end

    % Button pushed function: SaveP6Button
    function SaveP6ButtonPushed(app, event)
```

```
        app.P6(1) = app.Actuator_1_Pot_Set.Value;
        app.P6(2) = app.Actuator_2_Pot_Set.Value;
        app.P6(3) = app.Actuator_3_Pot_Set.Value;
        app.P6(4) = app.Actuator_4_Pot_Set.Value;
        app.P6(5) = app.Actuator_5_Pot_Set.Value;
        app.P6(6) = app.Actuator_6_Pot_Set.Value;

        app.P6CheckBox.Value = 1;
        app.MovePlatformPtoP.Enable = 'on';
        app.ClearP6Button.Enable = 'on';
        app.MovetoP6Button.Enable = 'on';
    end

    % Button pushed function: ClearP1Button
    function ClearP1ButtonPushed(app, event)
        app.P1 = zeros(1,6);
        app.P1CheckBox.Value = 0;
        app.ClearP1Button.Enable = 'off';
        app.MovetoP1Button.Enable = 'off';
    end

    % Button pushed function: ClearP2Button
    function ClearP2ButtonPushed(app, event)
        app.P2 = zeros(1,6);
        app.P2CheckBox.Value = 0;
        app.ClearP2Button.Enable = 'off';
        app.MovetoP2Button.Enable = 'off';
    end

    % Button pushed function: ClearP3Button
    function ClearP3ButtonPushed(app, event)
        app.P3 = zeros(1,6);
        app.P3CheckBox.Value = 0;
        app.ClearP3Button.Enable = 'off';
        app.MovetoP3Button.Enable = 'off';
    end

    % Button pushed function: ClearP4Button
    function ClearP4ButtonPushed(app, event)
        app.P4 = zeros(1,6);
        app.P4CheckBox.Value = 0;
        app.ClearP4Button.Enable = 'off';
        app.MovetoP4Button.Enable = 'off';
    end

    % Button pushed function: ClearP5Button
    function ClearP5ButtonPushed(app, event)
        app.P5 = zeros(1,6);
        app.P5CheckBox.Value = 0;
        app.ClearP5Button.Enable = 'off';
        app.MovetoP5Button.Enable = 'off';
    end

    % Button pushed function: ClearP6Button
```

```
        function ClearP6ButtonPushed(app, event)
            app.P6 = zeros(1,6);
            app.P6CheckBox.Value = 0;
            app.ClearP6Button.Enable = 'off';
            app.MovetoP6Button.Enable = 'off';
        end

        % Button pushed function: MovePlatformPtoP
        function MovePlatformPtoPButtonPushed(app, event)
            output_length(1,:) = app.P1;
            output_length(2,:) = app.P2;
            output_length(3,:) = app.P3;
            output_length(4,:) = app.P4;
            output_length(5,:) = app.P5;
            output_length(6,:) = app.P6;

            % This checks for any rows of 0 and removes, leaving an
array of non-zero positions
            % Limitation of inability to go to reset position as part
of
            % the sequence
            output_length=output_length(any(output_length,2),:);

            for w = 1:size(output_length,1)
                MoveToPoint(app, output_length(w,:))
            end
        end

        % Button pushed function: MovePlatformReset
        function MoveReset(app, event)
            Home_Pos = [10,10,10,10,10,10];
            if app.serial_state == 1
                fprintf(app.ser, [strjoin({num2str(Home_Pos(1)),
num2str(Home_Pos(2)), num2str(Home_Pos(3)), num2str(Home_Pos(4)),
num2str(Home_Pos(5)), num2str(Home_Pos(6)), num2str(60)}), '\n']);
            end

            app.ErrorMessage.Visible = 'on';
            app.ErrorMessage.Value = [strjoin({num2str(Home_Pos(1)),
num2str(Home_Pos(2)), num2str(Home_Pos(3)), num2str(Home_Pos(4)),
num2str(Home_Pos(5)), num2str(Home_Pos(6)), num2str(60)}), '\n'];

            for k = 1:length(Home_Pos)
                app.Actuator_Current_Pos(k) = Home_Pos(k);
            end
        end

        % Button pushed function: MovetoP1Button
        function MovetoP1ButtonPushed(app, event)
            MoveToPoint(app, app.P1)
        end

        % Button pushed function: MovetoP2Button
        function MovetoP2ButtonPushed(app, event)
```

```
            MoveToPoint(app, app.P2)
        end

        % Button pushed function: MovetoP3Button
        function MovetoP3ButtonPushed(app, event)
            MoveToPoint(app, app.P3)
        end

        % Button pushed function: MovetoP4Button
        function MovetoP4ButtonPushed(app, event)
            MoveToPoint(app, app.P4)
        end

        % Button pushed function: MovetoP5Button
        function MovetoP5ButtonPushed(app, event)
            MoveToPoint(app, app.P5)
        end

        % Button pushed function: MovetoP6Button
        function MovetoP6ButtonPushed(app, event)
            MoveToPoint(app, app.P6)
        end

        % Callback function
        function CalibrationButton_2Pushed(app, event)
             MoveDone = fread(app.ser,1);
            if MoveDone == 'D'
                app.ErrorMessage.BackgroundColor = 'r';
            end
        end
    end

    % App initialization and construction
    methods (Access = private)

                %These properties (GUI object creation) were omitted
for brevity

    end

    methods (Access = public)

        % Construct app
        function app = StewartPlatformApp_exported

            % Create and configure components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)
```

Appendix L

STEWART PLATFORM LAB MANUAL

EXPERIMENT ONE

STEWART-GOUGH PLATFORM

INTRODUCTION:

In this experiment, you will be working with a type of parallel robot called a Stewart Platform. The platform was initially developed from the late 1940s to mid-1960s. Eric Gough established the principles of a closed-loop kinematic mechanism in 1947 and built a prototype for tire testing in 1955. In 1965 D. Stewart published “A Platform with Six Degrees of Freedom" as flight simulators rose in popularity. Despite significant contributions by both men the platform has eventually become known as the Stewart Platform. Figure 1 shows the Stewart Platform that will be used in this experiment.

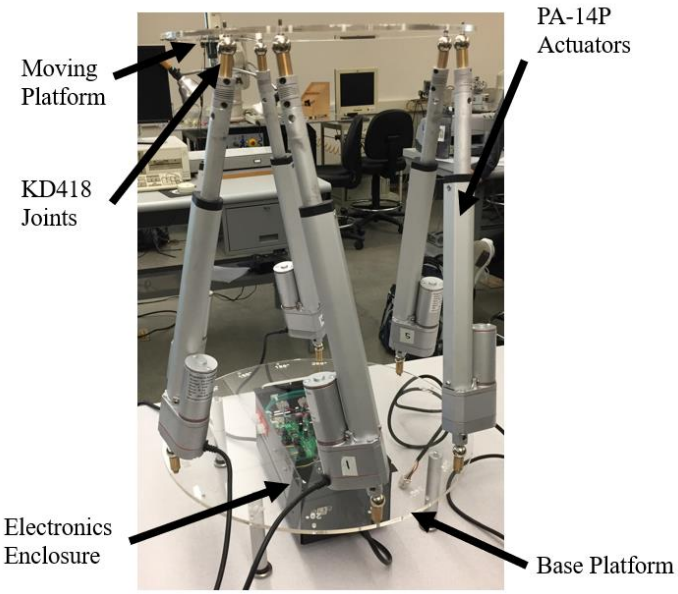


Figure 1: Stewart Platform

The following sections introduce the primary components used in the 6 DOF SPS Stewart Platform.

Progressive Automation PA-14P Linear Actuators

This Stewart Platform is comprised of 6 linear actuators made by Progressive Automations. These actuators have a stroke of 8 inches and are powered by 12VDC motors, enabling extension and retraction at up to 2 in/second. They feature internal potentiometers for position feedback and limit switches that prevent extending or retracting beyond the actuator design. These comprise the prismatic portion of the Stewart Platform's kinematic chains.

KD418 Magnetic Ball Joint

Magnetic ball joints are used at either end of the actuators to fulfill the S portion of the SPS mechanism. The joints are comprised of two pieces: a steel sphere with a threaded rod, and a cylindrical magnet that interfaces with the sphere. Magnetic ball joints are available commercially at various sizes and strengths. The KD418 has a holding force of 5kgs (11 lbf) and will separate under greater loading. This holding force allows the joint to operate as intended under normal conditions, and separate if a crash or binding occurs. They are designed to be the failure point, preventing excess loading on the actuators and acrylic plates.

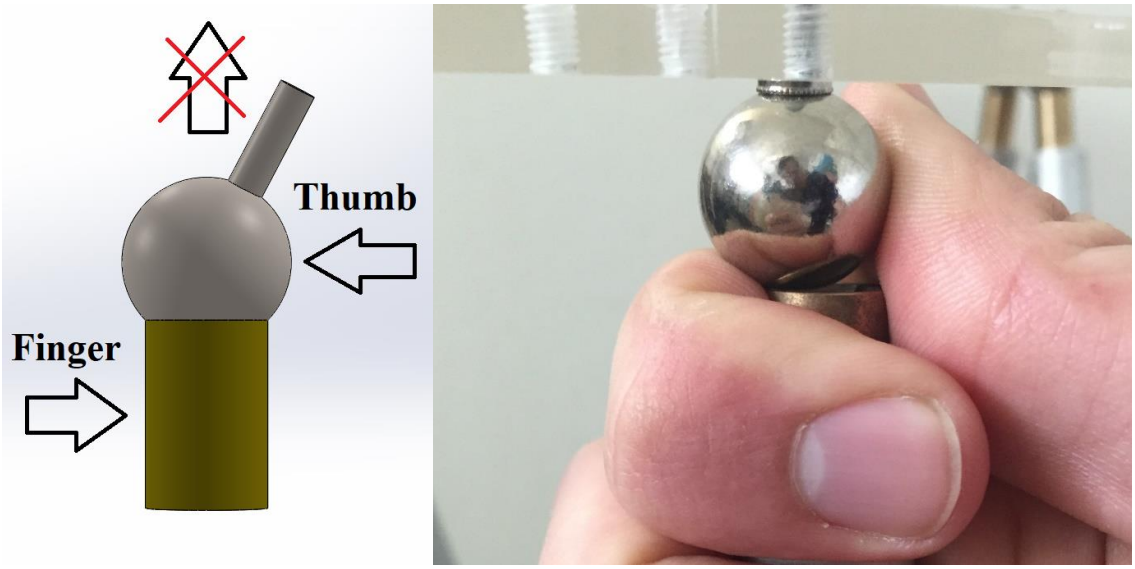


Figure 2: KD418 Separation Technique

There will be a portion of the experiment that requires separating the magnets and spheres. It is recommended to follow Figure 2 for easiest removal. Do not separate along the cylindrical axis; hold the cylinder with a finger or two and use the base of your thumb to push the sphere laterally. The magnets were tested for a crushing or pinching risk by placing a finger between the sphere and magnet, and no magnetic clamping force was experienced.

Arduino Due

The controller behind the Stewart Platform is an Arduino Due, powered by an AT91SAM3X8E microcontroller that operates on 3.3VDC at a clock speed of 84 Mhz. The Due has 54 digital I/O pins, of which 12 can be configured to provide PWM output. It also has 12 analog input pins.

The Stewart Platform uses 13 digital output pins: 6 direction pins to specify extension or retraction, 6 PWM outputs, and an enable pin. 6 analog input pins will read the potentiometer's signal voltage and correlate it to position in software. The input pins are set up for 10-bit analog-to-digital (ADC) conversion meaning the 8" stroke will have a range of 1024 position values, equal to 0.0078"/count. The Due has a control loop programmed that will act on inputs from the host PC.

HexaMoto Shield

The Arduino Due is unable to output any significant amount of current to power the linear actuators, so an external power source and motor drivers are needed. The HexaMoto Shield is a custom PCB that attaches to the top of the Due and has 6 motor drivers. They are connected to a 12VDC power supply and use the Due's PWM and direction signals to power the actuators in a controlled manner. The HexaMoto also features screw terminals to easily connect the potentiometers to the Due's analog inputs. These main components are labeled in Figure 3.

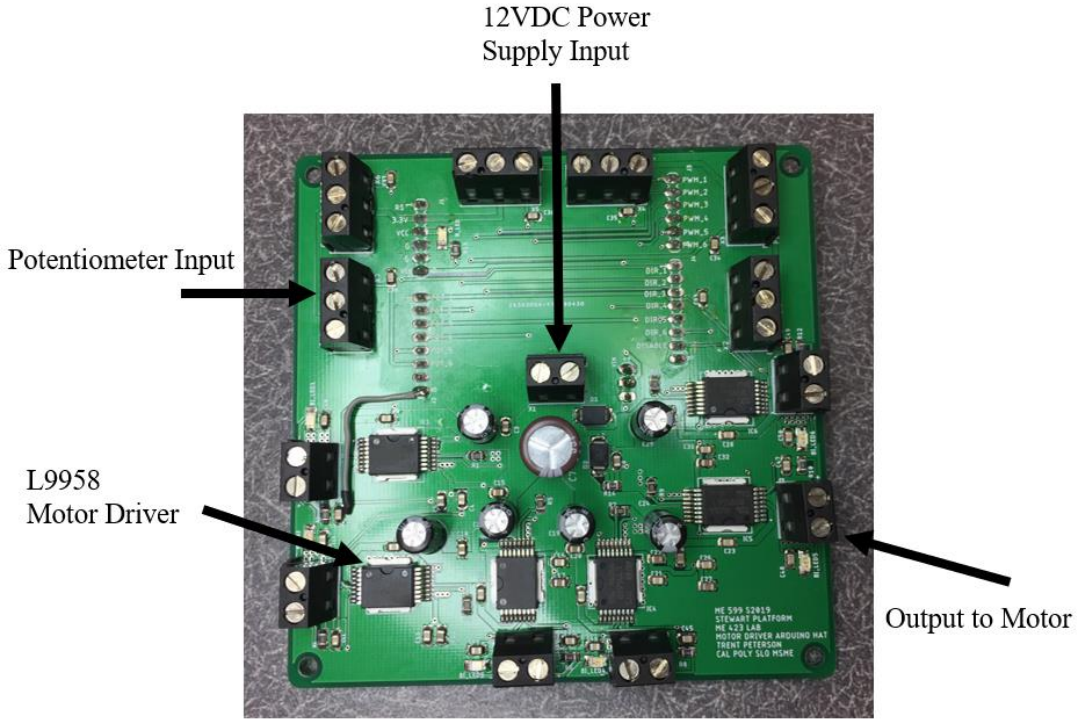


Figure 3: HexaMoto Components

Inverse Kinematics:

The Stewart Platform has 6 degrees of freedom which allows control over both position and orientation of the moving platform. Forward kinematics for the Stewart Platform involves determining position and orientation of the platform given actuator link lengths which requires advanced numerical analysis methods beyond the scope of this course.

The inverse kinematics return the link lengths given position and orientation and are much more straightforward for the Stewart Platform. Figure 4 sets up the derivation of the inverse kinematic relations.

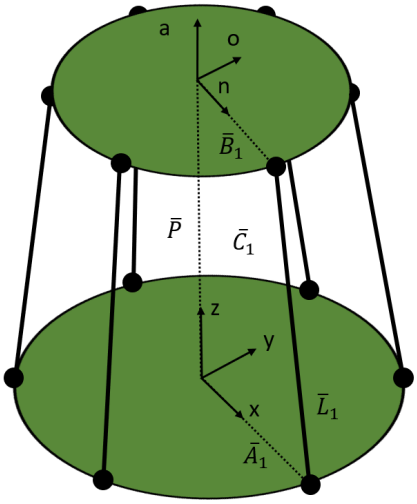


Figure 4: Vector Definition for Inverse Kinematic Derivation

The desired vector for each actuator *i* is

$$\overline{L_i} = \overline{P} + \overline{B_i} - \overline{A_i} \qquad \text{(Equation 1)}$$

where

$$\overline{A_i} = \begin{bmatrix} Acos(i_{th}) \\ Asin(i_{th}) \\ 0 \\ 1 \end{bmatrix} \qquad \text{(Equation 2)}$$

and

$$\overline{B_i} = \begin{bmatrix} Bcos(i_{th}) \\ Bsin(i_{th}) \\ 0 \\ 1 \end{bmatrix} \quad \text{(Equation 3)}$$

However, $\overline{\boldsymbol{B}}$ changes as the top platform rotates about any of the three axis and must be accounted for with rotation matrixes. Note that theta, phi, and psi correspond to the x-, y-, and z-axis respectively.

$$\begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{rot} = \begin{bmatrix} Rot(z,\psi) \end{bmatrix} \begin{bmatrix} Rot(y,\psi) \end{bmatrix} \begin{bmatrix} Rot(x,\theta) \end{bmatrix} \begin{bmatrix} B_ix \\ B_iy \\ B_iz \\ 1 \end{bmatrix}_{home} \quad \text{(Equation 4)}$$

Substituting the rotated $\overline{\boldsymbol{B}}$ matrix into the vector equation yields

$$\begin{bmatrix} L_{ix} \\ L_{iy} \\ L_{iz} \end{bmatrix} = \begin{bmatrix} P_{ix} \\ P_{iy} \\ P_{iz} \end{bmatrix} + \begin{bmatrix} B_{ix} \\ B_{iy} \\ B_{iz} \end{bmatrix} - \begin{bmatrix} A_{ix} \\ A_{iy} \\ A_{iz} \end{bmatrix} \quad \text{(Equation 5)}$$

Simulation and Verification of Equations - Foreword

The first part of the experiment, you will use Matlab to first run a simulation of the Stewart Platform. A majority of the simulation will be provided, with some intentional omissions. To get the simulation working, you must:

1. Write a function called *stewartrot* that uses the rotation matrices in Equation 4 to transform $\bar{\boldsymbol{B}}_{\boldsymbol{home}}$ to $\bar{\boldsymbol{B}}_{\boldsymbol{rot}}$. The inputs to *stewartrot* must be *theta*, *phi*, *psi*, and *b_home*. The function will apply the rotation matrixes to *b_home* and output a 4x1 matrix *b_rot*.

   It is strongly advised to create and test this function before the lab. Save it to a flash drive to load it into the Matlab folder. This will help complete the simulation portion quickly.

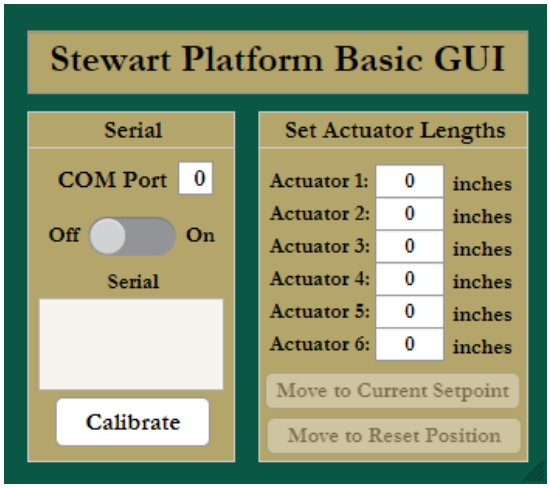
2. In lab you will complete the inverse kinematics for the generic case. The inverse kinematics for the 6-6 and 6-3 Stewart Platform configurations are already included in the simulation. You may refer to the setup and usage of these two versions of the inverse kinematics to create the general case kinematics. Further detail will be given in the experiment.

EXPERIMENT

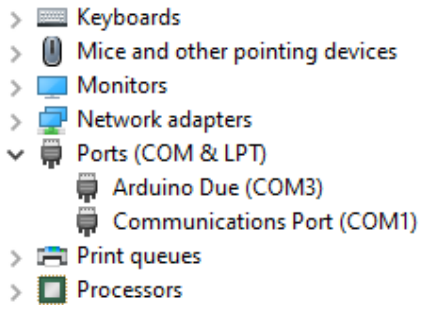
Starting the Stewart Platform

- Ensure the power cable is plugged into an outlet and to the electronics enclosure. Connect the USB cable between the Stewart Platform and the host PC. Verify the Arduino Due is connected by observing a green LED on the board. Turn on the power switch and confirm the power supply is on by observing a green LED on the unit.

- Open the *StewartPlatformStarterApp* shortcut on the desktop. The following GUI should appear:



- In order to communicate with the Arduino Due, a serial connection must be established. From the Start Menu in Windows, open Device Manager and locate the COM Port used by the Arduino Due. Insert this value into the GUI. In the provided example, the value to enter would be '3'. Exit this window once the value is obtained.



- Press the Calibrate button. This will open the serial connection and the robot will extend the actuators to their maximum length, pause, and retract to their minimum length. The Arduino is reading the potentiometer readings and verifying proper functionality. A calibration status message will be returned. The Stewart Platform is now ready to move.

- Input the following values into the GUI

	Move 1	Move 2	Move 3	Move 4	Move 5
Actuator 1	2.0	2.0	2.0	2.0	6.8
Actuator 2	2.5	3.0	3.0	3.0	7.9
Actuator 3	2.0	2.3	2.3	2.3	6.8
Actuator 4	2.5	5.2	5.2	5.2	7.9
Actuator 5	2.0	2.8	3.8	1.8	6.8
Actuator 6	2.5	2.8	2.8	2.8	7.9

- Observe the resulting position and motion of the Stewart Platform. Note that you are commanding actuator lengths, and it would take complicated forward kinematics to determine the platform position and orientation from these lengths. This lab will focus on the use of inverse kinematics to calculate the link lengths from a specified the platform's position and orientation.

- Note that only actuator 5 changes in Move 3 and Move 4. Note the effect that one actuator's movement has on the platform position and orientation.

- Before closing the GUI, move the platform to reset position.

Simulation and Verification of Equations

- Open Matlab and open *Stewart_Platform.m*. Transfer your written and tested function *stewartrot.m* to the same folder as *Stewart_Platform.m.*

- In the **%%Set Constants** section of the simulation, *base_diameter* and *top_diameter* are missing. Using a tape measure, measure the bolt circle diameters of the base and moving platform, rounded to the nearest tenth of an inch. Update these variable values.

- Under **%Set base and top angles** create *base_angle* and *top_angle* which are 1x6 matrices containing the angular positions of the spherical joints, in ascending order.
    - Use base angles of 0°, 80°, 120°, 200°, 240°, and 300°.
    - Use top angles of 0°, 60°, 120°, 180°, 240°, and 300°.

- In **%% Calculate EOMs** locate the inverse kinematics for the 6-6 and 6-3 configurations and observe their usage. Using these as a reference:
    - Create *a_i* and *b_i* in a similar manner, using *base_angle* and *top_angle* as the inputs to sine and cosine.
    - Using your tested function *stewartrot.m*, rotate *b_i*. Store the output of this function in *b_rot*.
    - Using Equation 5, calculate the array *L_vector* containing the x, y, and z lengths of the actuators using *P, a_i,* and *b_rot.*
    - Calculate the magnitude of *L_vector* and store it into *L_length* and end both nested for loops and the outer if statement.

- Run the simulation with *animation* true and *plots* false. The platform should increase in Z and in theta.

- Set *configuration_6_3* to true. The geometry of the 6-3 configuration is already set. Verify the configuration change but the motion remains the same.

- Under **%% Motion Profile Creation**, there are several motion profiles that specify the values of each degree of freedom throughout the simulation. Select a motion profile to simulate. An example of a motion profile is shown below.

```
for m = 1:length(n)
    theta(m)    = -30 + 30 * m/max(n);      %[degrees], relative to x axis
    phi(m)      = 0;                        %[degrees], relative to y axis
    psi(m)      = -30 + 60 * m/max(n);      %[degrees], relative to z axis
    Px(m)       = 0;                        %[inches], x position
    Py(m)       = 0;                        %[inches], y position
    Pz(m)       = 20 + 2*cosd(m/max(n)*360);%[inches], z position
end
```

- With *animation* true and *plots* false, run the script. Observe the animation and verify the motion matches the motion profile.

- With *animation* false and *plots* true, run the simulation again to generate the plots. Save them for inclusion in the report. In the report, identify if this motion profile would be possible on the actual Stewart Platform and justify your determination.

- Using a configuration of your choice, make a final motion profile that includes motion in all degrees of freedom. Ensure that this motion profile stays within the Stewart Platform link length limits and actuator speed limits. You may use linear motion, as seen in the *theta* and *psi* equations, but for more interesting plots and animations, attempt a complicated motion profile that uses sin, cos, or other continuous functions. When complete, show the instructor your animation, and save the plots for inclusion in your report.

Instability Investigation

- Configure the Stewart Platform to be a true 6-6 configuration, where all joints are evenly spaced at 60° apart. With at least two people, disassemble the Stewart Platform:
    - First, turn off power and unplug all connections.
    - Remove one linear actuator by separating the magnet at the end of the actuator from the steel sphere on the moving platform. The top plate is now unconstrained and must be held. Finish removing the actuator by separating the sphere at the base of the actuator and the magnet attached to the base platform and set it on the desk.
    - Finish removing remaining 5 actuators and set the moving platform down.

- Move the magnets on the base platform to 0°, 60°, 120°, 180°, 240°, and 300°. Move the steel spheres on the moving platform to the same angles.

- Reconnect the linear actuators to both platforms. Connect actuator 1 to the joint at the smallest angle on both the moving and base platforms. In this case, actuator 1 would connect to both joints at 0°, and actuator 6 would connect to both joints at 300°. Follow this pattern at all times when reconfiguring the robot. **Keep hold of the top platform even when this process is complete.**

- Is the robot stable? Try to position the robot into a stable position using your hands. You can let go slightly to check stability, but do not let the platform fall. If you find a stable position, what happens when it is twisted slightly? Observe carefully and in your report, provide an explanation regarding the stability of the Stewart Platform in a true 6-6 configuration.

Range of Motion: 6-6 Configuration

Reconfigure

- Change the Stewart Platform configuration to a modified 6-6 configuration. Disassemble the Stewart Platform as before and move the magnets on the base platform to 0°, 80°, 120°, 200°, 240°, and 320°. Move the steel sphere joints on the moving platform to 0°, 40°, 120°, 160°, 240°, and 280°.

Startup

- Ensure the power cable is plugged into an outlet and to the electronics enclosure. Connect the USB cable between the Stewart Platform and the host PC. Verify the Arduino Due is connected by observing a green LED on the board. Turn on the power switch and confirm the power supply is on by observing a green LED on the unit.

- Open *StewartPlatformApp.mlapp* from the desktop shortcut. The following GUI should appear.

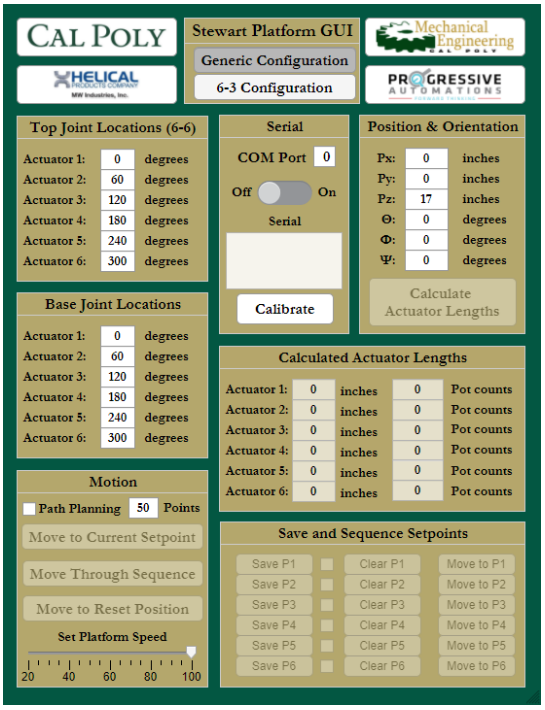


Figure 5: Stewart Platform GUI

- Set the configuration to Generic Configuration. This is a modified 6-6 that allows all 12 joint locations to be input. Input the values in ascending order according to the Stewart Platform setup.

- Reconnect all cables. Enter the COM port of the Arduino Due and calibrate the robot using the same procedure with the basic GUI.

Testing

- Verify that the top and base joint locations are correct. Find the maximum value the Stewart Platform can achieve about each axis, one at a time. Achieve this by using the "Position and Orientation" panel and "Motion" panel. First, input desired position and orientation values and select "Calculate Actuator Lengths". The output of the calculation appears on "Calculated Actuator Lengths" for your review.

- Under the "Motion" panel, use "Move to Current Setpoint" to move the Stewart Platform to the position currently calculated. Throughout the experiment, use the 'Path Planning' and 'Speed' features and observe the motion differences. If you desire to move the Stewart Platform back to its reset position, use the "Move to Reset Position" button at any time.

- You may alter other axis values to maximize the desired axis value. For example, Figure 6 shows that actuators 2, 4, and 6 have effectively reached their 8-in, 1024 potentiometer count limit with all other "Position and Orientation" inputs equal to zero. However, changing the value of Ψ from 0° to -20° allows Pz to increase by 0.4 inches.

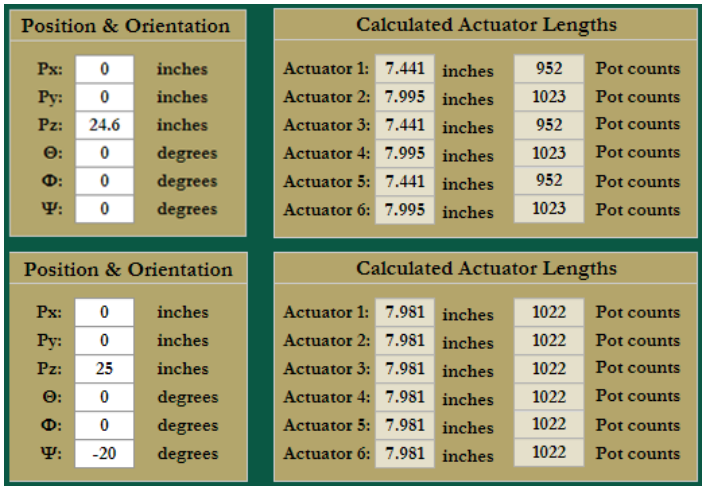


Figure 6: Tip for Maximizing Range of Motion

- It's recommended that you find the maximum z-coordinate, theta rotation, and phi rotation first. These can be found without the robot actuating into an unstable position.

- Finding the limits of Px, Py, and Ψ are more difficult. You cannot rely on the actuator lengths calculation to determine the maximum value, as doing so may cause the robot to collapse. Find the value by incrementing it, moving the robot and checking stability, and incrementing again. You may want to have someone keep their hands close to the moving plate in case the robot has been extended too far. If it does overextend, a magnetic joint or more will separate, and the platform will fall. Simply reattach the joints and move the platform location to a more centered value. Hold the moving platform initially to support it as it moves to a more stable position.
- In this configuration the Stewart Platform can achieve a much larger Ψ value in the negative direction than in the positive direction. Verify this is the case and report the largest magnitude of Ψ.

Value to Maximize	Px (in)	Py (in)	Pz (in)	Θ (°)	Φ (°)	Ψ (°)
Px						
Py						
Pz						
Θ (about n-axis)						
Φ (about o-axis)						
Ψ (about a-axis)						

Range of Motion: 6-3 Configuration

Reconfigure

- Change the configuration of the robot to a 6-3 configuration. Disassemble the Stewart Platform as before and move the magnets on the base platform to 0°, 60°, 120°, 180°, 240°, and 300°.
- On the moving platform attach two spherical joints at 0°, 120°, and 240° using the pair of threaded holes adjacent to these angles. Connect actuators 1 and 2 to the 0° joints, actuators 3 and 4 to the 120° joints, and actuators 5 and 6 to the 240° joints. This is also specified in the GUI.

Note that if two magnetic cylinders were to be attached to the same spherical joint, they cylinders would be greater than 90° apart, as shown in Figure 7. This cannot be achieved with the given Stewart Platform geometry, so pairs of spherical joints are used to best represent a true 6-3 configuration. The resulting angular offset from the nominal 0°, 120°, and 240° should be neglected in the inverse kinematic equations. In *Stewart_Platform.m*, the 6-3 inverse kinematics utilize *joint_angle* to account for the offset. This joint angle is utilized by the GUI as well.

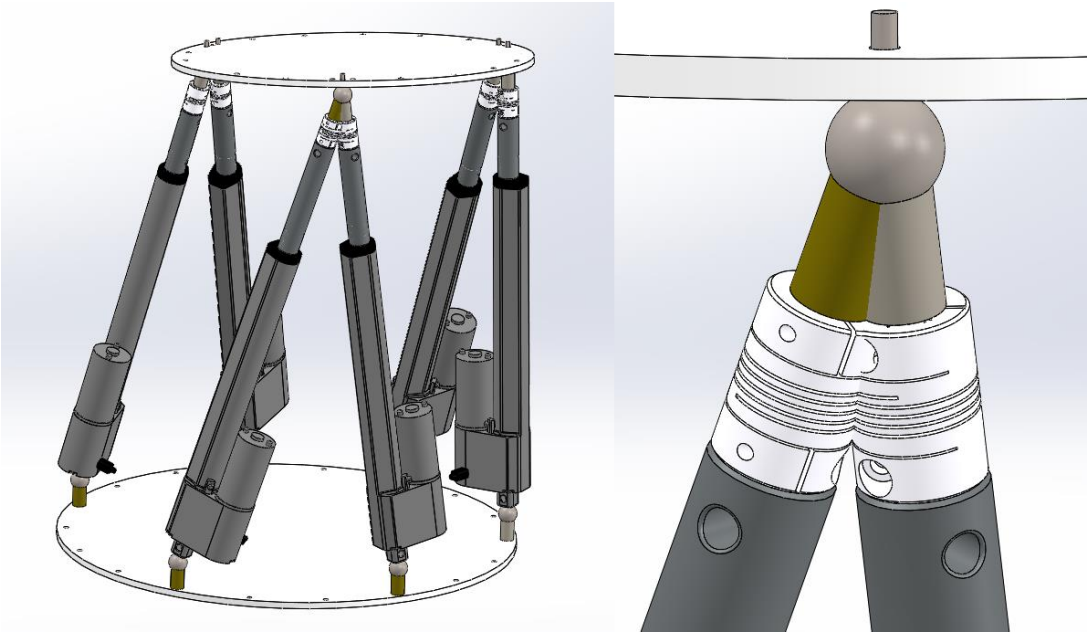


Figure 7: Interference Resulting from True 6-3 Configuration

- Perform the same range of motion tests for the six degrees of freedom and record them.

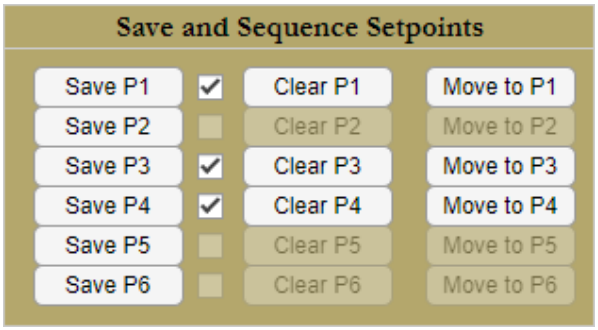
Value to Maximize	Px (in)	Py (in)	Pz (in)	Θ (°)	Φ (°)	Ψ (°)
Px						
Py						
Pz						
Θ (about x-axis)						
Φ (about y-axis)						
Ψ (about z-axis)						

- In the report, compare the range-of-motion differences between the two configurations and give a brief summary describing how the Stewart Platform geometry affects its range of motion.

Flight Simulator

This portion of the experiment investigates one of the original uses of a Stewart Platform: a flight simulator. This portion will build upon the experiment thus far by incorporating motion sequencing.

After a position and orientation has been input and calculated, it can be saved using the buttons found in the "Save and Sequence Setpoints" panel. If a saved setpoint is incorrect or needs to be changed, it can be saved again and overwritten. If no longer desired, the saved setpoint can be cleared.



When the "Move Through Sequence" button is selected, the platform will move sequentially through all the saved setpoints. It will ignore any unsaved or cleared setpoints.

Using the axis specified in Figure 8, simulate:

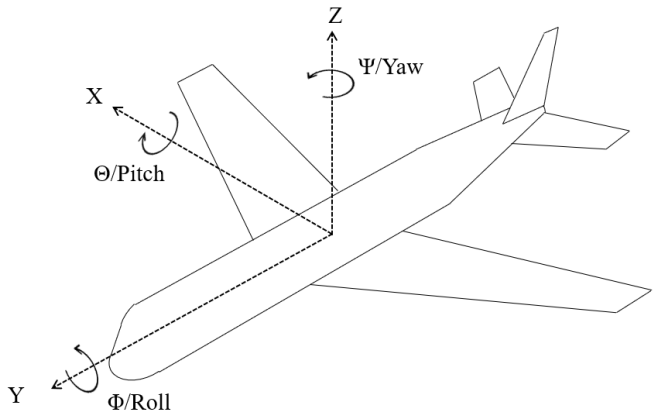


Figure 8: Axis Specification for Aircraft Flight Simulator

- A takeoff, in which the plane's pitch and acceleration causes the passengers to feel a combined total of 0.5 Gs of acceleration in the -Y direction. Calculate the Stewart Platform pitch necessary to simulate this acceleration by using gravity and save your result in P1.
- While still accelerating and rising, a turn to the left, in which the aircraft yaws 10° and rolls -8°. Save this calculated setpoint to P2
- The aircraft leveling out, and yawing -45°. Save this to P3
- The aircraft banking to the right with a roll of 15° (maintaining yaw). Save this to P4
- A landing approach, in which the aircraft has a negative pitch and no roll or yaw. The passengers experience 0.75Gs in the -Z direction (which includes acceleration due to gravity). Determine the pitch necessary to simulate this and save it to P5.
- A landed aircraft on a level tarmac. Save this to P6.

While only changing pitch, roll, and yaw is necessary to complete this simulation, you may wish to add translation along the X, Y and Z axes to mimic the aircraft's relative position. For example, a takeoff could include an increase along Z by a couple inches, or a banking/turning left would result in a decrease along X. You have all 6 degrees of freedom at your disposal, be creative!

- Run the sequence of setpoints to run the flight simulator, and have your instructor verify your motion.

Cleanup Procedure

- When completed with the experiment, move the Stewart Platform to the reset position for the next lab group.
- Turn off the switch on the electronics enclosure to power off the motors and unplug the USB cable from the computer to disconnect serial and power off the Arduino.
- Close the GUI and delete all generated files from this lab session.