SSN – PKM1

Design and Realization of a 100G Parallel Kinematic Manipulator

A Major Qualifying Project Report
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

M. Spenser Brouwer

John W. Cushion

Elizabeth M. De Zulueta

Joshua L. Janssen

Sean F. Townsend

Date: April 25, 2012

Approved:

April 26, 2012

Professor Stephen S. Nestinger

Parallel Kinematic Manipulator

Submitted by:

Spenser Brouwer

John Cushion

Elizabeth De Zulueta

Joshua Janssen

Sean Townsend

Submitted to:

Professor Stephen Nestinger

April 26, 2012

Abstract

The goal of this project was to design, fabricate and implement a parallel kinematic manipulator robot with open architecture to be used in the Industrial Robotics course, ME/RBE 4815, curriculum for inverse kinematics and other classroom projects. The robot was designed using solid modeling software and fabricated in the WPI machine shop with use of computer aided manufacturing techniques. A programming architecture was also developed in concert with the mechanical-electrical system to allow control of the robot via a touchscreen GUI.

Table of Contents

Table of Figures

1. Introduction

Industrial robots are divided into two major categories – serial and parallel kinematic robots. The WPI Robotics Lab in Washburn Shops has several serial robots, which are used extensively in the Industrial Robotics course, ME/RBE 4815, but currently has no parallel robots. Parallel robots are particularly useful in a teaching setting, as they provide a gentler introduction to inverse kinematics than their serial counterparts; having one available for student use would be an excellent learning tool. The major limiting factor is price: a typical parallel kinematic robot can cost upwards of $30,000, even with an educational discount. This project endeavored to completely manufacture, assemble, and program a parallel kinematic robot for student and classroom use at a fraction of the cost of a proprietary unit. As a secondary goal, the robot was designed to be capable of accelerations up to 100Gs under ideal conditions, comparable to the abilities of most advanced parallel robots currently available.

1.1 Project Statement

The goal of this project was to develop an open source, reliable parallel kinematic manipulator for use in the Industrial robotics course at WPI. This robot will be designed to have an open architecture, and will allow for students to utilize their own microcontrollers to pass control signals to the robot which will allow for greater flexibility in the operation of the robot.

1.2 Report Layout

The rest of the report is organized as follows. Chapter 2 provides the history of parallel kinematic industrial robotics. Chapter 3 covers the methods performed in pursuit of the project goals, particularly in three major areas: mechanical, electrical and computers. Chapter 4 summarizes the results of the previous chapter. Chapter 5 relates prototyping of preliminary designs for both three-arm and four-arm styles of robot. Chapter 6 covers the kinematics of the

four-arm system that was selected to be pursued as the final design of the robot. Chapter 7 covers the final design of the system and the process followed for manufacturing of the robot, including the support table and electronic subsystems. Chapter 8 covers the software architecture, including the process of GUI design and passing commands to the motor controllers. Chapter 9 provides an overall summary of the project. Chapter 10 provides recommendations for future work across all three of the main components of the robot: mechanical, electrical and computer programming. Finally, chapter 11 provides the conclusions that were determined during the project, such as what worked, what did not work and a final summary of future improvements.

2 Background

In order to determine the state of the art with regards to parallel kinematic manipulators, it was decided to research the history of industrial robotics in general and PKMs in particular, from their earliest roots in 1942.

2.1 History of Parallel Robotics

Parallel kinematics has its roots at the beginning of industrial robotics. In 1942, Willard L.V. Pollard was awarded a patent for the first spatial industrial robot, designed for spray painting. The design of the manipulator relied upon parallel kinematic chains to provide 5 degrees of freedom, but was never actually built. Pollard's son, Willard L.V. Pollard Jr., was the one who finally designed and built the first industrial parallel robot. The next development was parallel robots that rely not on motors or rotary joints, but upon hydraulic rams (Zhang, 2).

In 1965 D. Stewart described a design for a 6-DOF robot meant for flight simulation. These devices consist of a raised platform atop six prismatic actuators, which allow for manipulation of the orientation of the platform with six degrees-of-freedom (three translational and three rotational) (Zhang, 4). Such devices are usually referred to as Stewart platforms or hexapods (for their six "legs"). The first device of this type for flight simulation applications was built by Klaus Cappel in the mid-sixties, and was a basic form of modern flight simulators. While Stewart is often credited for their invention, the first octahedral hexapod was actually developed in 1954 by Dr. Eric Gough. It was called the "Universal Tyre Testing Machine" or just "Universal Rig," and had been designed and implemented for uses in tire load testing. Derivatives of these two parallel robots were the standard for about 30 years.

Until recently, most of the work in parallel kinematic robots was built upon the "Gough-Stewart" design type. However, the early 1980s saw the invention of the Delta robot by

Reymond Clavel. Created in response to industry demand for a way to quickly move light objects, it consisted of three arms attached to a base above the work space. There were two features of this design which made it particularly effective; first, the positioning of all actuators at the base (as opposed to along its arm, as in a serial robot) allowing for reduced arm weight and thus much faster movement; second, the use of two parallel linkages in each arm restrict rotational movement of the end effector (rotation at the end effector is achieved with a separate actuator, either at the base or the end effector). The company Demaurex produced earliest delta robots for use in industrial packaging. (Clement, 3)

There has also been a recent surge in development of mechanisms that use rotary joints as opposed to prismatic joints. Some examples of these are the "Hexa" robot developed by Uchiyama in 1994, which has 6 DOF (Zhang, 15); the Adept Quattro, which has 4 DOF; and the ABB Flexpicker, which also has 4 DOF. Some key points of the design of these robots are: an overhead style, high speed, high acceleration, stable coordinate translation, and increased stiffness of the entire robot.

Parallel kinematic robots differ from serial kinematic robots, where each joint controls one degree of freedom. Serial robots are far simpler and have their own set of pros and cons. They are far easier to control, can have larger work cells with regard to their own volume, and can approach more locations from a wide array of angles. Conversely, they suffer from additive errors in each of their links as well as being limited in payload due to the fact that the robot must support the weight of successive motors as well as the item being manipulated. As a result, they are not the best solution for pick and place operations with short cycle times.

Currently, the companies ABB and Adept produce a majority of parallel kinematic robots. ABB introduced its FlexPicker robot in 1999. In 2006, Adept introduced the Quattro, a

four-arm delta-style robot; its additional arm allowed for increased accuracy and speed. Although parallel robots are still produced for packaging purposes by SIG Pack, a subsidiary of Bosch, the Flexpicker and the Quattro are considered the standard for parallel industrial robotics. These two robots were designed for industrial pick-and-place applications. Both have capacities between 3 and 6 kg and have cycle times between 0.3 and 0.7 seconds. They are also capable of repeatability of motion to within 0.1 mm in any direction. It is because of these qualities that this project was tasked with the design and manufacture of a parallel kinematic manipulator. In order to get a better idea for the capabilities of both the ABB FlexPicker and the Adept Quattro, research was performed using the marketing materials provided by both companies. These materials gave an idea of the sizing for individual components, as well as the benefits and drawbacks to each design

2.2 Three-Arm Parallel Kinematic Manipulator

The first three-arm parallel robot was built in 1988, based off Reymond Clavel "delta" design, a parallel kinematic manipulator with three arms driven by revolute motors. This system allows 3 degrees of freedom of the end effector, all spatial translation. The delta utilizes symmetry to create a simplistic system that can perform a pick and place operations at high speeds and high accuracy. The motors are mounted to a base plate that is fixtured above the work cell. The arms are attached with revolute joints, which allow the end effector to attain 3 degrees of freedom. The delta was a big development for the industry. The speeds that the delta could attain while maintaining its precision and rigidity made parallel kinematics the ideal choice for pick and place operations on small objects.

Since the Delta's appearance in the late 80's, many companies have been investigating this technology and many companies have their own versions of a three-armed parallel kinematic

manipulator. ABB's FlexPicker is one of the industry leaders in three arm parallel kinematic manipulators.

Figure 1 - ABB FlexPicker

ABB offers "clean room" specifications for their robots so parallel robots have made their way into the health and food industry. ABB is only one of many companies that have noted the efficiency of these systems and developed versions of their own.

2.3 Four Arm Parallel Kinematic Manipulator

A four arm PKM has four motors that attach to four individual driver links; these driver links then attach to four separate arms. These arms tend to be attached by ball joints or ball-and-socket joints at both ends of the driver link, meaning that for every motor there is one driver link and for every driver link there are 2 links that make up one arm. The arms are then connected to the end effecter; thus, the end effector is connected to eight links but only four arms.

This system is over-constrained; when you set the angles of three of the arms the angle of the fourth arm is determined. Although an over-constrained system has less flexibility, the over-constrained system is considered to be "redundant" and the additional links provide an additional

element of stability and control to the entire kinematic system (Clement, 5). Movement in the cardinal directions is created by moving one motor forwards and its opposite motor backwards; whether the robot is moving in the x-direction or the y-direction depends on which motors are moving. Movement along the z-axis requires coordination motion of all four joints in the same direction.

The Adept Quattro is the current leader in four-arm parallel robots. It features an operating payload of 2 kilograms, operating speeds of up to 10 m/s and repeatability of motion within .1 millimeter. The workspace of the Quattro is also the largest in the industry, with diameters of over 50 inches. These specifications made it a worthy goal for this project to match in terms of performance.

2.4 Summary

The history of parallel robotics is quite recent, although there have been great strides in the field especially since their inception in the 1950s. There are two major types of parallel manipulators, 3-arm delta robots or 4-arm Quattro style robots. These robots have become increasingly common and there are a number of them on the market, such as the ABB FlexPicker and the Adept Quattro. Through this research, it was determined that a four-arm manipulator had several important benefits, including requiring less torque to generate high accelerations due to the addition of the fourth motor and providing a greater level of stiffness for the end effector. The four-arm style has several drawbacks, however, including increased weight of the moving system, the necessity to purchase four potentially expensive motors as opposed to only three, and more material required to manufacture the fourth arm. It was noticed that the three-arm design also had their own benefits, including a light end effector and requiring fewer raw materials to

construct. It suffers from several drawbacks, including requiring more powerful motors, less payload capability, and reduced repeatability of motion.

3 Methodology

To achieve the goals proposed by this project, a methodology had to be determined and followed. Once the system goals were defined, the various options were prototyped and a final design style was selected. A mechanical design phase then proceeded in parallel with electrical and software developments. The mechanical designs proceeded from simple first drafts and concept sketches to final models designed for ease of manufacturing as well as amount of material needed. The electrical system proceeded from a functional analysis to component selection, and finally to production of a working control system. The software development proceeded from determining what choices for programming language would work for the project, to determining what motor control communications protocol would allow for synchronous motion of all four motors, to the creation of a graphical user interface that will allow for users to interface with the robot.

4 System Goals and Requirements

The final mechanical system goals were defined as needing a workspace of roughly 30 inches in diameter and a height of 20 inches. The system should also be capable of unloaded accelerations of 100G, carry a payload of 2 kilograms and operate at speeds up to 10 m/s at full payload. The electrical system had the goals of allowing external microcontroller architectures to be utilized in an open system, and as such required adequate system protection for both the third party microcontroller and the internal system. The computer system was designed to allow for a comparable control system to other industrial robots, allow for the creation of movement programs, and be relatively simple for users to utilize, through both mouse and keyboard or touch screen input.

4.1 Mechanical Design Constraints

The goal of this project from the beginning was to design a Parallel Kinematic Manipulator that could be implemented for educational purpose in the Mechanical and Robotics Engineering curriculums at WPI. For this reason, a table top design was needed that could easily be added to WPI's existing robotics laboratories. This led to the first design constraint: the final product must have a foot print that is 30 in x 30 in and a work space height of 36 in.

The Current State-of-the-art technology of parallel kinematic manipulation is the Adept Quattro, being credited as the fastest robot on the market. It was desired that the system to be comparable with the current industrial PKM's. Therefore, it was desired that the system operate at speeds up to 10 m/s with a peak acceleration of 15g (roughly 150 m/s^2).

Lastly, it was a goal to design a system that was capable of reaching a peak acceleration of 100g, or roughly 980 m/s^2 for research purposes. One of the focuses of the preliminary

research was: whether it was desirable to follow a three-armed “delta” or four-armed “Quattro” approach.

4.2 Target Users and Audience

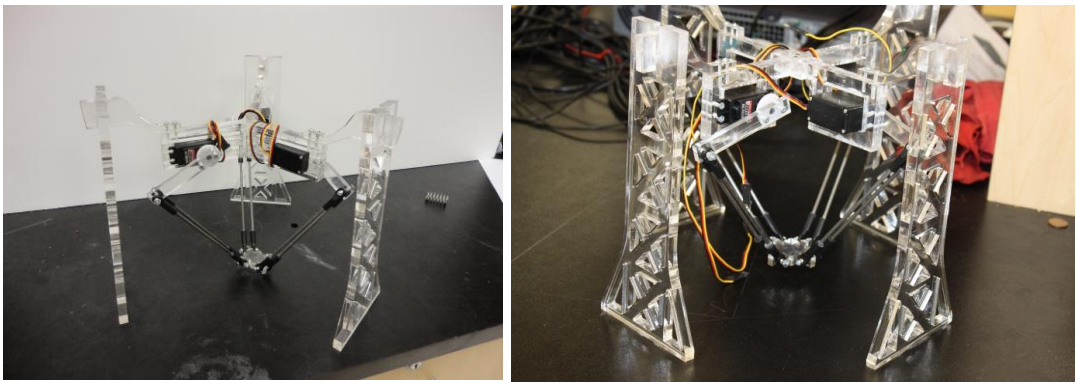
To successfully design and implement this robot, understanding who is going to use the robot and its intended use are extremely important. This robot will be used as an educational tool in the Industrial Robotics course, ME/RBE 4815; it will be used to teach students inverse kinematics in a closed kinematic loop. Therefore, it needs to be user-friendly, safe, and have the ability to be seamlessly integrated into the course curriculum. The course currently utilizes a Fanuc 200iB robotic arm which allows students to manually jog to positions or code into the system; as well, it interfaces with a National Instruments Data Acquisition Module (NI-DAQ) which allows students to connect sensors and send signals to the system. To integrate this robot into the course, many of those features were emulated such as the GUI interface which is similar to the teach pendant as well as the interface board which is similar to the available NI-DAQ. System integration into the current curriculum will be greatly facilitated by designing the robot with the goal of supplying a system tailored to the target users’ needs.

5 System Design Evaluation and Prototyping

Simplified models of the two designs were developed using SolidWorks© to gain an idea of the basic proportions of both robots, as well as the required input for both designs to be completely constrained. The kinematic chains of the models consisted of 3 lines with the main rotation joint being constrained into planar motion. A ratio of approximately 1:2.5 of the driver arm to the follower arm was selected based on researching other, existing designs and determining what factors contributed to workspace size and shape.

To gain firsthand experience regarding the pros and cons of the three-arm and four-arm designs of a parallel kinematic robot as well as to parallelize software development, the group built small scale prototypes of both parallel kinematic configurations, as shown in a) Prototype of the delta style robot. b) Prototype of the Quattro style robot.

Figure 2.



a) Prototype of the delta style robot. **b) Prototype of the Quattro style robot.**

Figure 2 – Snapshots of the miniature acrylic prototypes.

The prototypes were fabricated out of acrylic, due to the low cost, availability of a laser cutter on campus, familiarity with the properties of the material, familiarity with the assembly methods and time constraints. Hitec 485GH servos were used, which provided good location control as well as appropriate amounts of torque to control the prototypes. These also have the benefit of being inexpensive and easy to use as they take a PWM signal to control their current position. A flat layout was created for the bases of both prototypes, to hold the servos and provide support for the rest of the mechanism; a set of legs; and the driver arms were created, and were sent to the laser for cutting. The driven links were created using Traxxas turnbuckles from a radio control car and steel rods to create a four-bar linkage, which was attached to the driver arm and a small manipulator platform.

5.1 Programming for Proof of Concept

To program the prototype our team decided to use the DyIO (Dynamic Input/Output) by Neuron Robotics. The DyIO is a small input and output board which can be utilized when programming small robots.

When we began testing the servos, the arms were still attached. This made it difficult to discern which servo was moving because, due the construction of the system, when one servo moves it causes the arms attached to it to move and they pull the other arms and servos into motion. We detached the arms, making it easy to see which servo was moving and how it was moving. After experimenting with the motion of each servo, both individually and in relation to the other servos, we decided that a center for each servo had to be declared to make coordinated movement possible. The relative center for each servo had a slightly different potentiometer value, but the position was determined by moving the servo until the driver arm was parallel to the servo. The servos were then labeled 0 through 3 to make it easier to identify which servo was connected to Port0, Port1, Port2, or Port3. When one servo moves, the opposite servo must move, as well, or pressure will be put on the system.

The proof of concept program is written in the Java programming language, the native language for the DyIO. Selecting Java simplified communications with the DyIO since no extra libraries or classes had to be downloaded or written to interface with board.

Three classes were developed to control both the three-arm and four-arm prototypes. These classes take the predefined input, ServoChannels, which is provided by the DyIO. Channels 0 through 3 were chosen to provide output to the servos, and variables were then declared in order to center the servos. Methods were then written to take place inside two separate classes, ThreeArm and FourArm. Two methods that were written for basic motion

include recenter() and setAngle(). The method for recenter() merely sends each servo to its home position using the preset center variable. The setAngle() method allows for the program to adjust the angle of each individual servo, which allows for a coordinated motion program to be created.

In order to test the programming written for the prototypes and to show that coordinated motion is possible, a Test class is created. Inside this class, new instances are created of FourArm and ThreeArm, and a main() is written. This class is where commands sent to the prototypes are written, and due to the organization of Java, code outside of the main() is not executed, so therefore all test programs are created inside of the main() method.

It was determined from this small scale program testing important factors for initialization of the full scale robot, including the requirement for the motors to be tested individually prior to the attachment of the end effector and driver arms. This is an important factor to prevent accidents during assembly and testing of the control logic and electronic components.

5.2 Summary

The prototypes were found to be very valuable because they provided a valid starting point for the design of the full scale robot. By utilizing the prototypes for small scale tests as well as a focus of discussion, it was determined that the additional power provided by the four motors of the “Quattro” design would give it a larger payload and would make it more likely that the full size robot would be able to reach the goal acceleration of 100G. As a result, it was decided to proceed with the “Quattro” style Parallel robot for the final design.

6 Kinematics and Dynamic Modeling

Unlike the serial manipulator, determining the position of the parallel robot is quite difficult. A serial manipulator has one open kinematic chain that can be easily handled with the application of the Denavit-Hartenberg representation. This method, introduced by Jacques Denavit and Richard S. Hartenberg, reduces each joint of the robot down to two distance values, d and r, and two rotational values, theta and alpha. With this minimalist representation in place, four basic transformation matrices are used to determine the position of each joint. However, with a parallel robot, a forward kinematics solution is not as straight forward. The parallel kinematic manipulator has multiple closed kinematic chains and a direct kinematic solution is very complex and computationally intensive. For this reason, we have chosen to rely on an inverse kinematic approach. This method relies on both geometry and vector analysis, and no single solution exists. Inverse kinematics is the opposite of direct kinematics. In this model, the position of the end effector and the geometry of the robot are used to calculate the input angles of the motors.

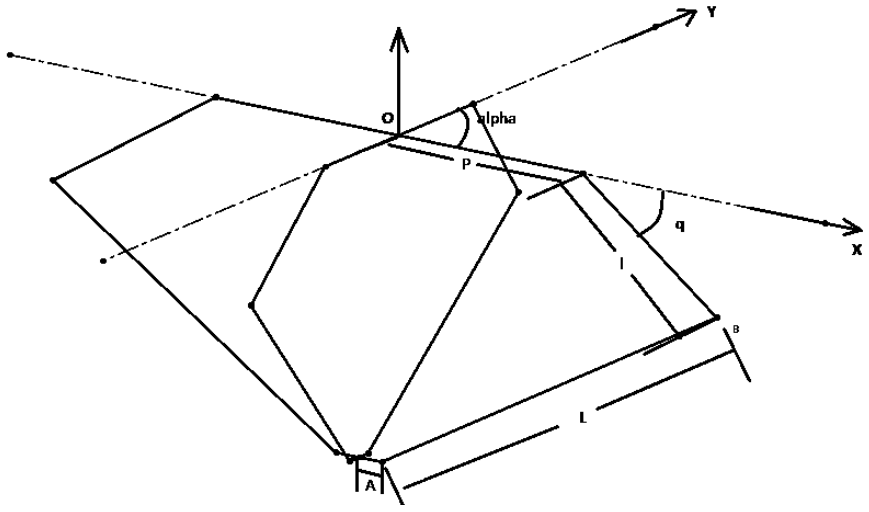


Figure 3 - System Parameter Definitions

The inverse kinematics of the four-armed parallel robot is calculated using both geometry and vectors. By knowing the position of the end effector and the position of the motors, the position of the middle joint ("knee" joint) can be calculated which leads to the calculation of the input angles of the motors. Figure 3 shows the setup of the kinematic system and the nomenclature of the design parameters.

Mathcad was used to calculate the angular position of each motor, **q**. The first section of the Mathcad file defines all of these constants. P is defined as the length from the origin of the robot to the rotational axis of the motor. The variable l defines the length of the driver arm, while L defines the length of the lower arm assembly. A defines the length from the point of rotation of the lower arm assembly to the center of the end effector platform. The user inputs the desired position of the end effector and Mathcad automatically calculates the necessary angular orientation of each motor.

The calculation is based off of the following equation.

$$\left||A_i B_i|\right|^2 = L^2|.$$

Simply, the magnitude of vector AB squared is equal to the length L squared. This is common sense but allows for an equation for **qi** (where **i**=1,2,3,4) to be derived from the definition of a vector.

$$(B_x - A_x)^2 + \left(B_y - A_y\right)^2 + (B_z - A_z)^2 = L^2.$$

Equations for Bx, By and Bz are created based on the geometry of the system and the unknown variable q. The three equations are as follows:

$$B_{xi} = P * \cos(\alpha_i) + l * \cos(q_i) * \cos(\alpha_i)$$

$$B_{yi} = P * \sin(\alpha_i) + l * \cos(q_i) * \sin(\alpha_i)$$

$$B_{zi} = l * \sin(q_i)$$

These equations are substituted back into the original equation and then a solution for q may be calculated.

There are two possible solutions for each motor. In each section, both are calculated. One of the positions will cause an inverted positioning of the arms that is not physically possible. To find the impossible solutions, a simplistic stick model was created in SolidWorks to verify the results of the calculations and to help visualize the movements of the arms.

Another goal of this project was to create an industrial robot with a workspace with a diameter of approximately 30 inches and a height of around 20 inches. In order to select the proper size of each arm of the robot, the maximum extents of each set of lengths had to be calculated.

The size of the workspace was determined geometrically through the input of the extreme angles for each arm using the properly scaled stick model. This geometric analysis revealed that the resulting workspace of the robot as designed meets the criteria laid out in the project proposal of a diameter of 30 inches and a height of 20 inches. The shape of the workspace was found to not be a perfect cylinder, but an upward facing parabolic shape. This means that at the work surface, the diameter of the workspace is about 8 inches, while at the maximum vertical extents it meets the desired 30 inch diameter.

6.1 Required Torque for 100G Acceleration

To determine the required motor and gearbox, it was needed to calculate the torque profile of the robot. As the goal of accelerations of 100G would be the most demanding of the motors as far as the torque required, the analysis performed focused on that scenario. To simplify the problem, it was decided that the optimal path for the 100G tests would be an oscillation along the Z-axis. This would allow for all four motors to work together in order to

attain that maximum speed and acceleration. With this simplification, the motion was analyzed as a four bar crank-slider mechanism with an offset. From this it is possible to find the torque profile along the Z-axis. An input of 1 Nm was used to show the correlated output in Newtons. The Mathcad file used to calculate and plot the torque profile along the Z-axis is provide in Appendix B. The curve shows the output of force at the end effector along the Z axis. It was found that at a range of Z=-28..-29, the greatest force output can be achieved. It was concluded that this location in the workspace would be optimal for achieving maximum acceleration.

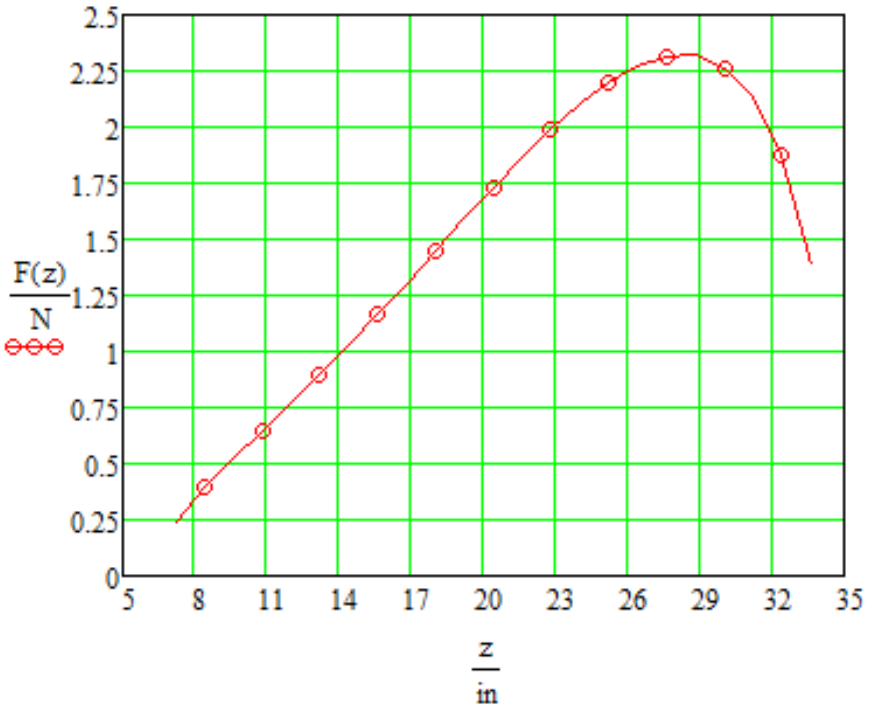


Figure 4 - Torque Profile along the Z-axis

7 System Design and Manufacturing

Once the four-arm design was chosen through prototyping, a detailed design was then developed using SolidWorks. Several paper design iterations were used which lead to the conceptualization process using computer software. The main design constraint behind the

design was to develop a design that could be maintained within the limited budget as well as fit the strict weight requirements set to meet the performance specifications.

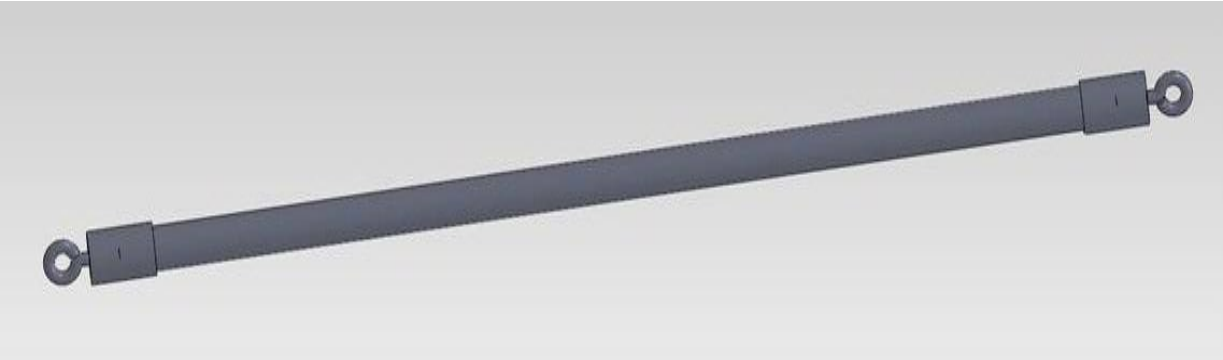


Figure 5 - Initial Arm Design

The initial designs were constructed with the idea that off-the-shelf parts would be easier to obtain, replace, and ultimately be a cheaper cost alternative to custom manufactured parts. As seem in Figure 5, the arm design uses a simple rod with caps on the end that are attached to eyebolts. The idea behind this was that two of these rods can be held together around the drive arm and platform using springs that would act as a backup safety mechanism should the robot venture out of its mechanically limited work plane. Such an extreme move would cause the springs to stretch and allow the platform to fall in order to prevent damage.

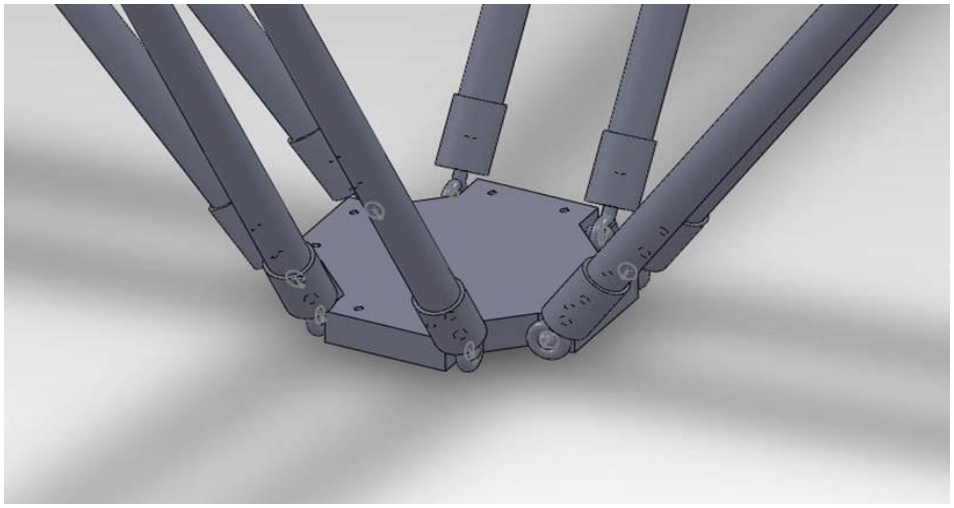


Figure 6 - Initial Connection Design Concept

In Figure 6 - Initial Connection Design Concept the off-the-shelf arms can be seen connected to the platform. The off-the-shelf concept proved to be generally effective in terms of cost analysis, however, when it came to material analysis and meeting the 1kg end-effector weight goal it became nearly impossible to use off the shelf parts. This factor forced a complete restructuring of the effective design possibilities that could be utilized to create a more robust, all-inclusive system of affordable and lightweight parts.

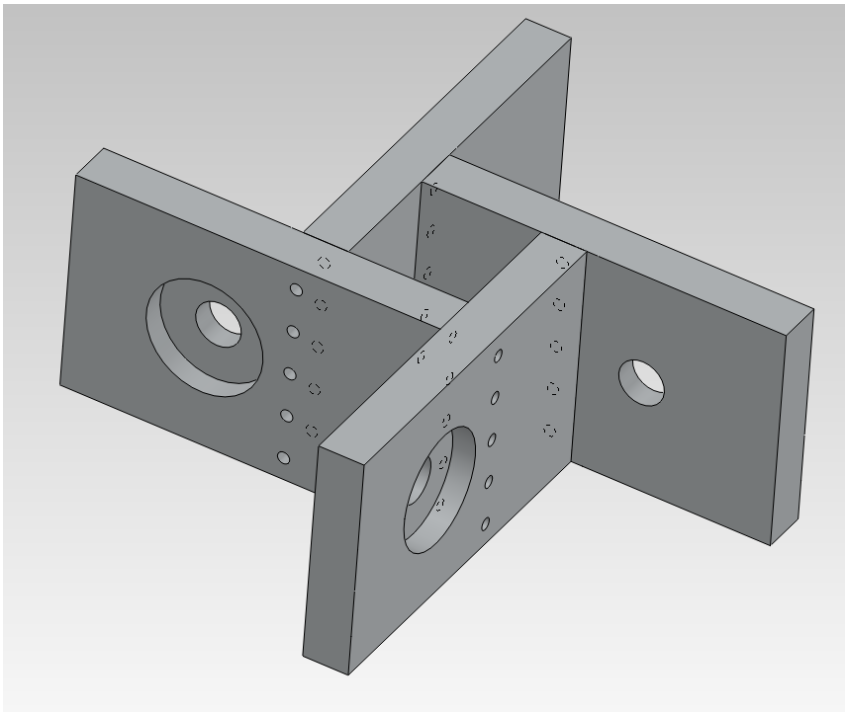


Figure 7 - Motor Mount Base

Being the first design concept for the base, in which all the motors would mount, it is the only component that could be designed without worrying about weight. The main design considerations for the base were that it had to be strong enough to support the Maxon motors and the torques that would be applied from the weight of the system. This being said the design concept was simple and utilized four 1”x8” slabs of aluminum that would be mounted together to form a rigid frame. Machined into each slab would be slots for the motors to mount into as well as tapped holes on the ends for each plate to mount to the other. When all was said and done the final motor mount base would look like Figure 7. This would weigh in at 14.5lbs and be by far the heftiest and strongest aspect of the entire design.

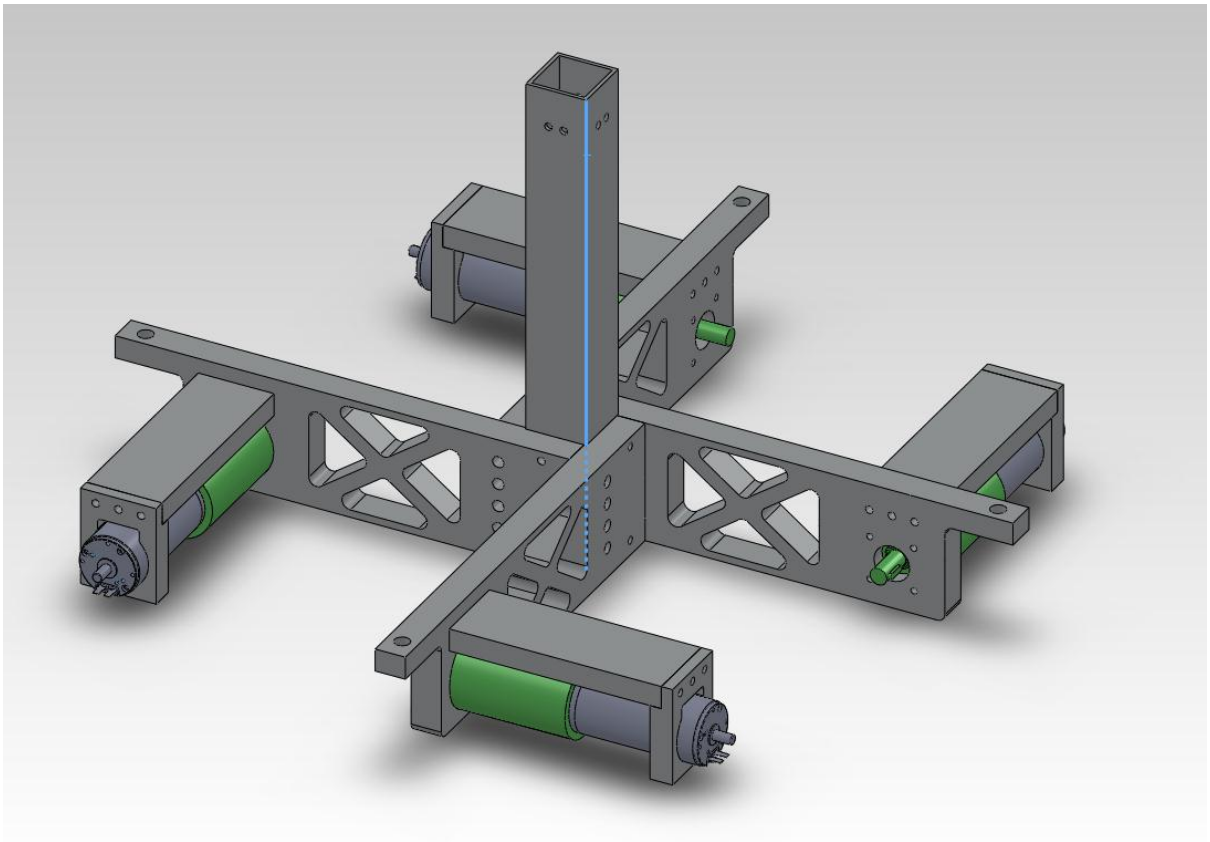


Figure 8 - Final Base Assembly (with Motor Supports)

Despite many series and revisions of the base mounting system, the final base design does not differ much from the initial design. Lengthening the base to meet the workspace criteria and thinning down/pocketing the base are among a few of the changes that were undertaken. The new base design allows for the drive arm axes to line up as well as become adjustable should the work plane need to change. This change was driven by the fact that the gear boxes had not arrived prior to final design approval, and with inconsistent CAD files from Maxon Motors, the drive shaft lengths available could not be trusted until the actual gear boxes arrived. The final base design also utilizes two different mounting points allowing for future modification of the robot workspace and work cell. This includes mounting the frame through the middle of the base plates, a 2”x2” square which mates with a piece of aluminum box extrusion that then attaches to the support frame, and four ½” mounting holes on the ends of each

base plate to allow for future work to be done on the robot.. The final design also incorporated a new feature, motor supports. These supports help alleviate stress on the mounting holes at the front of the base, keeping the motor shafts in line with their holes on the frame. Knowing that the entire assembly will be reaching close to 100Gs requires that the motors need to be securely attached to prevent catastrophic failures.

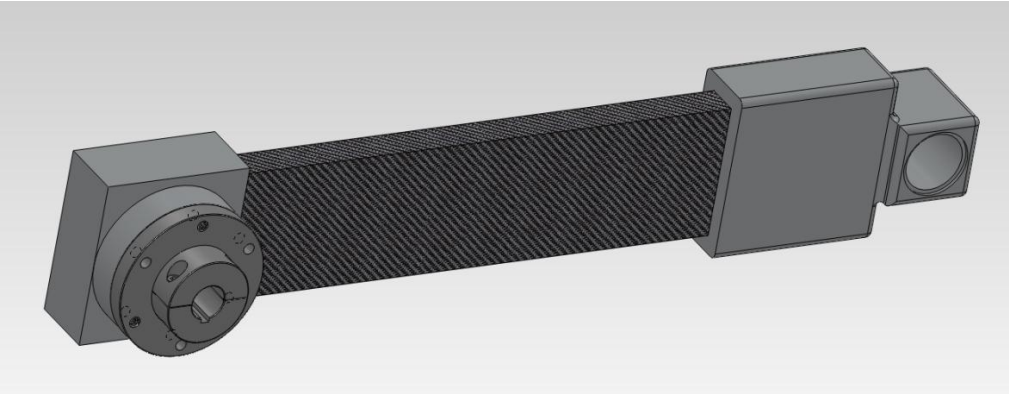


Figure 9 - Driver Arm Assembly

The driver arm was made by using two aluminum caps to sleeve over a 0.5”x 1’ x 0.018” hollow carbon fiber bar. As seen in Figure 9 the left side of the driver arm houses the connection to the steel coupler that would be used to connect to the drive shaft of the gearbox. Using carbon fiber significantly improved the strength of the driver arm while reducing the weight of the drive arm assembly by 2lbs, in comparison to a design that utilized aluminum that demonstrated one quarter of the strength.

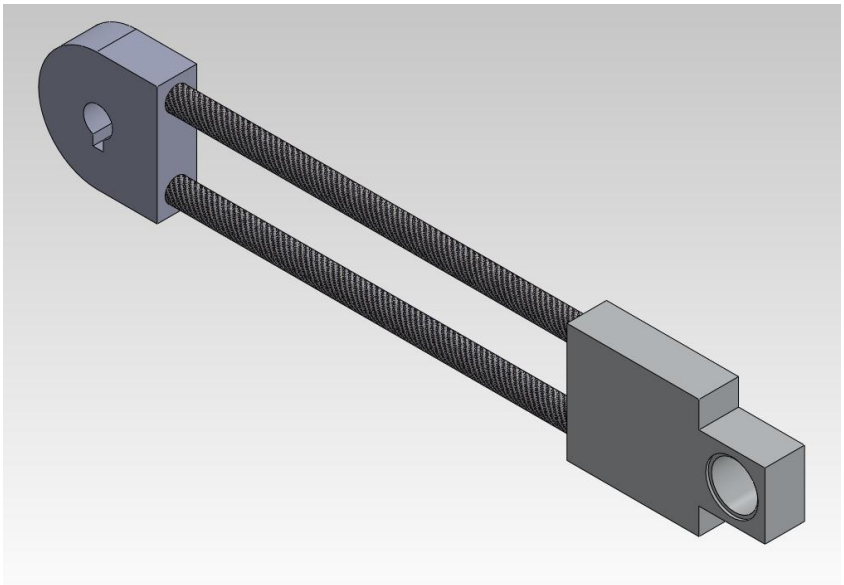


Figure 10 - Final Drive Arm Design

The final system however, called for lighter, stronger, smaller drive arms that could be manufactured with parts already purchased and without the addition of expensive off the shelf parts, as well as being capable of being machined. Square internal pockets are impossible to make through the machining processes available at WPI. Carbon fiber is also cheaper and stronger in round shapes, such as tubes. These factors contributed to a need for a redesign of the driver arms. The new design as shown in Figure 10 was the solution that was developed. By reducing the weight by a factor of two-thirds and adding twice the strength, the final drive arm design proved both easy to manufacture and the lightest design yet adding a big win to the final weight category of the robot.

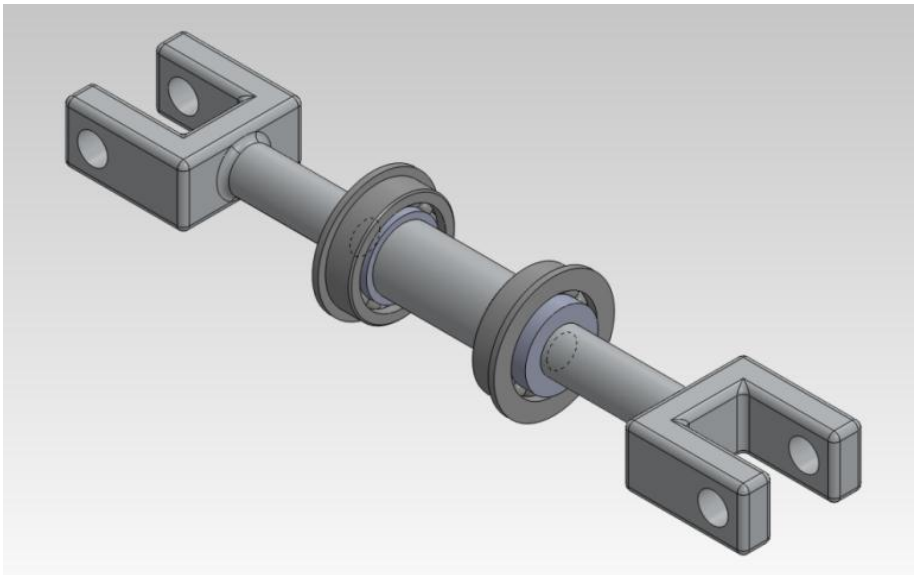


Figure 11 - U-Joint Connection Design

Throughout the design considerations, the lower arms and custom manufactured U-joints remained relatively the same. The same bearings are used in the final design; the same female U-joint lengths and even the male U-joint lengths are used in both the prototype design as well as the final design.

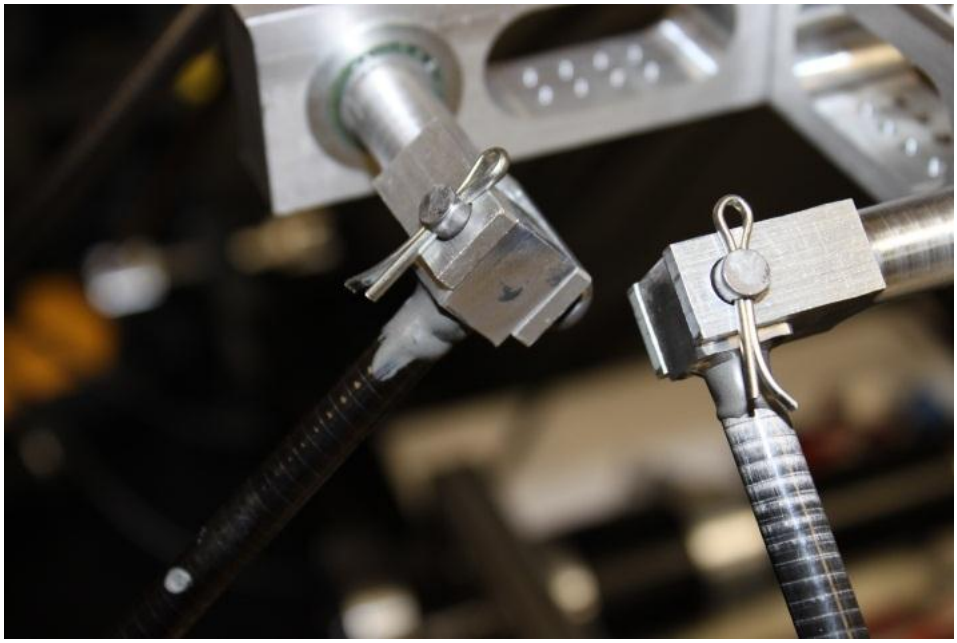


Figure 12 - Final U-joint Assembly

The only modification came in the center shaft and bearing spacing. Due to the impossible manufacturability of a two-diameter shaft as small as the one required for the given design, a new strategy had to be devised. The answer was to use the sub-assemblies' own tolerances to secure it together. The bearings, center shaft, and U-joints were all press fit together which saved time and money compared to the initial design. The "drive shaft" was the turning point in realizing that every part needed to be manufacturing friendly in order to make the overall product successful.

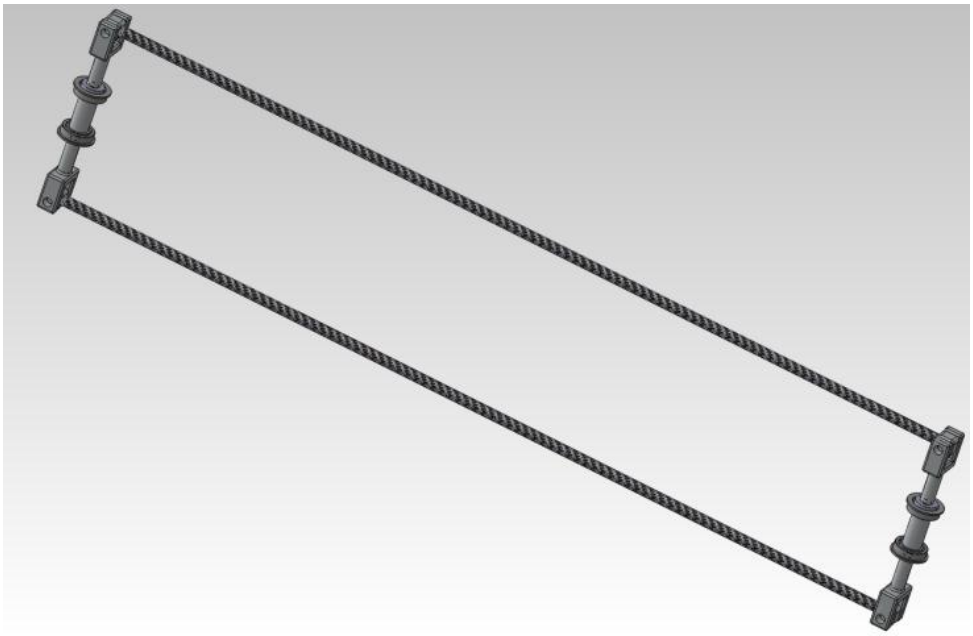


Figure 13 - Arm Assembly

Weighing about five pounds, the arm assembly was a work of art and the lightest assembly of the entire robot, despite being the largest subassembly in the robot.

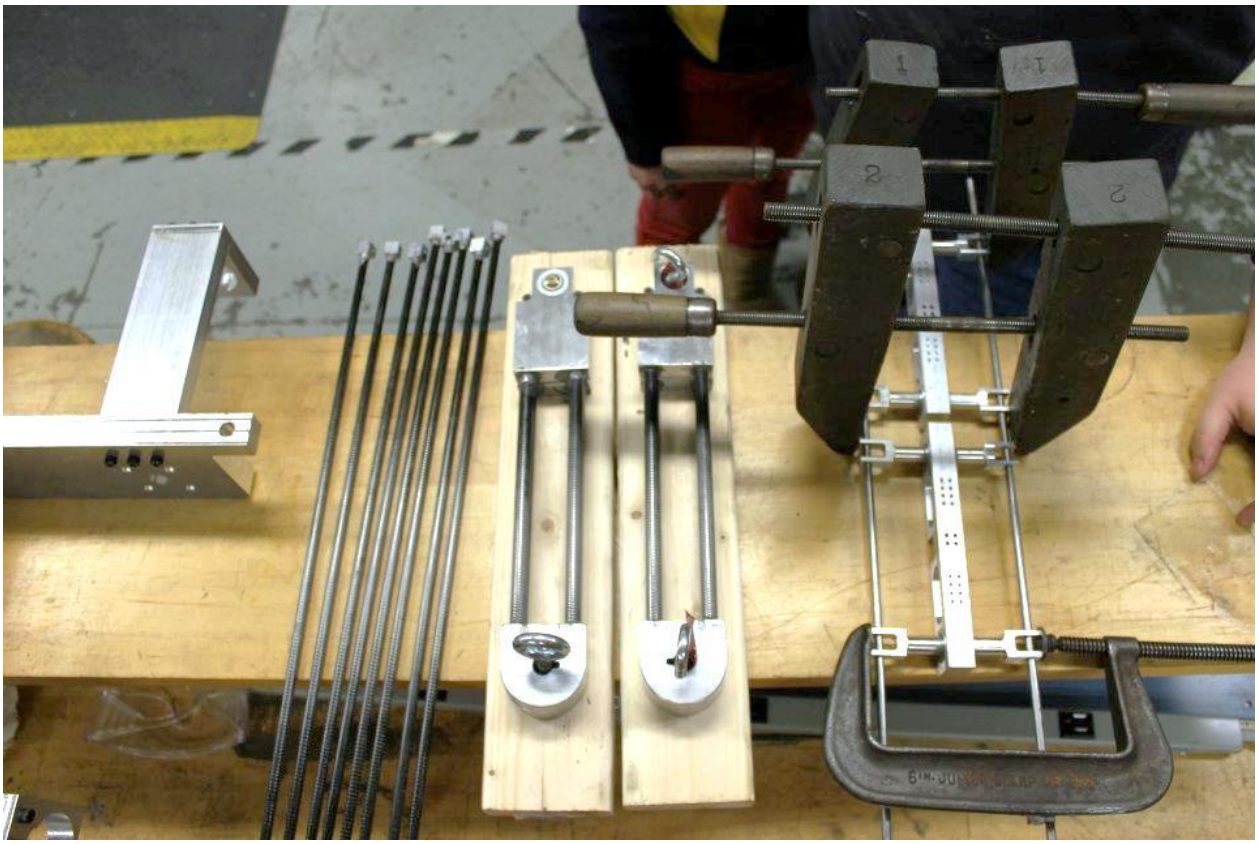


Figure 14- Subassemblies during epoxy setting

Assembly of the U-joints, drive shafts and arm assemblies were done in stages, these stages utilized jigs to ensure that every arm and u-joint assembly was spaced evenly. This was a crucial factor, as one arm being longer than the other would throw off the entire assembly of the platform. Due to the fact that carbon fiber is extremely strong when compressed and not expanded from within, the rods were never press fit onto or into any other part to ensure that they would not crack. After research and testing, it was determined that JB Weld was the best solution to bond the carbon fiber to the aluminum components. JB Weld is a two-part epoxy that has a tensile strength of 27.3MPa and adhesion strength of up to 12MPa. Due to its versatility, JB Weld was used on all parts where carbon fiber met aluminum.

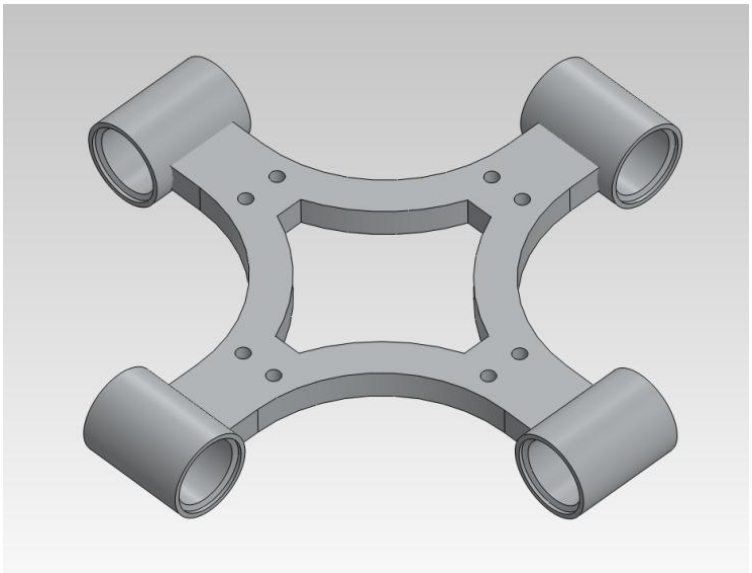


Figure 15 - Platform Assembly

The initial platform assembly was not only lightweight, but also had a better visual appeal than anything on the market today. However, after some serious machine time, coming back to review the platform had the group instantly realizing that such a picture perfect part was not a machine friendly part.

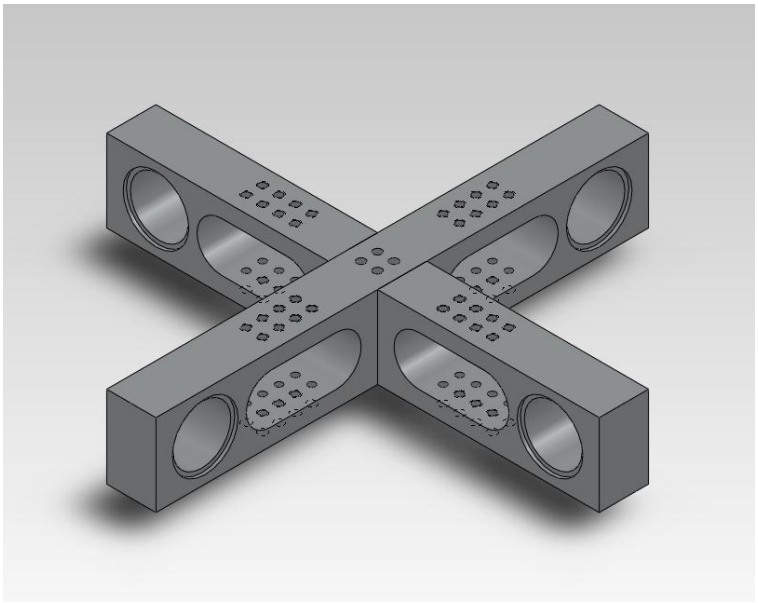


Figure 16 - Final Platform Assembly

The final platform assembly utilized parts that had already been purchased and was composed of two parts that could be easily machined as well as quickly assembled. The number of mounting holes on the re-design of the end effector was increased by a factor of four, making the overall robot more universal with a wider variety of end effector attachments.

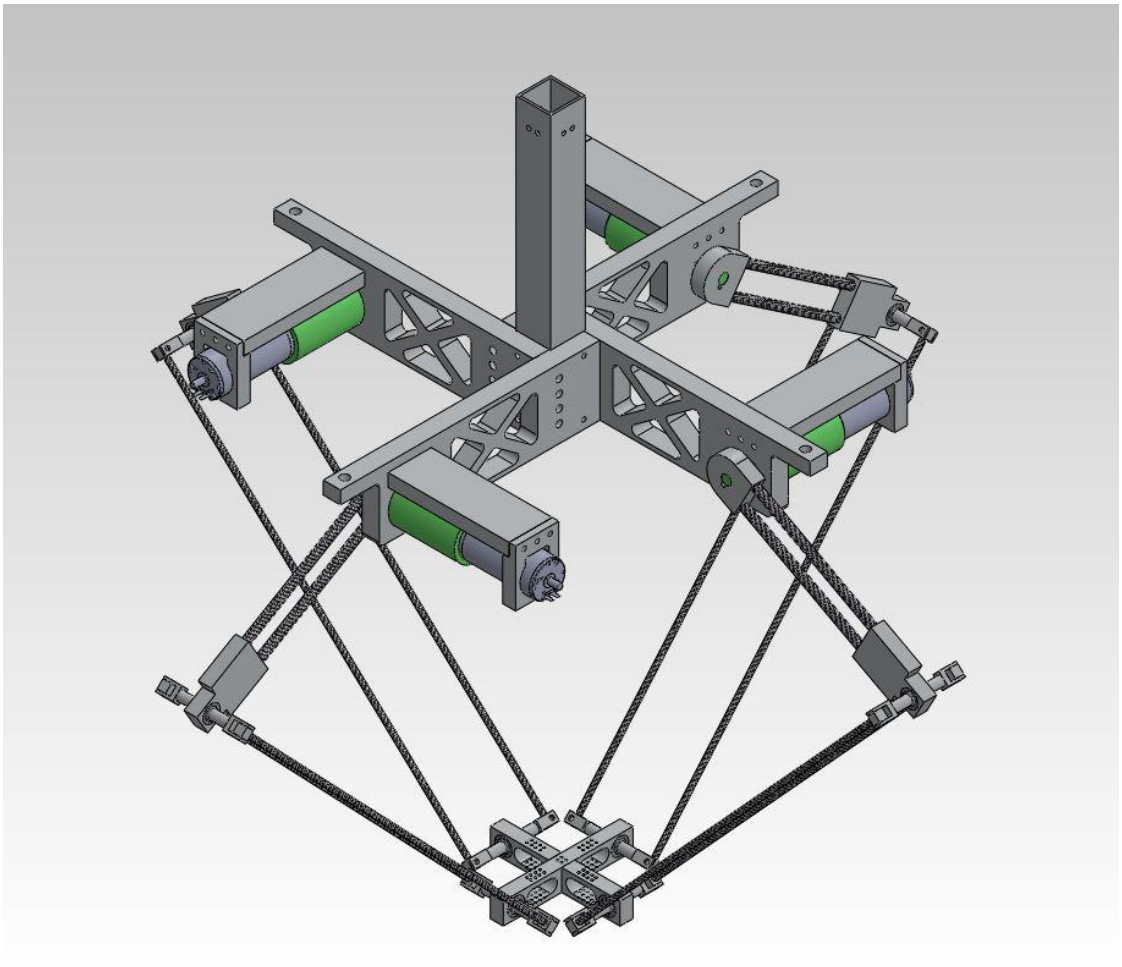


Figure 17 - Final Robot Assembly, as built

7.1 Design of Support Table

In order to support the robot as well as provide a workspace for users to perform their operations, a frame with integrated table was designed and built. Various options were pursued for the design of the table; in particular, a design utilizing materials from 8020 Inc. was favored for its ability to be reconfigured for different workspace needs. After a cost/benefit

analysis, it was determined by the team that similar table designs utilizing much cheaper inch angle steel would serve a similar purpose while still remaining within the project's budget.

The functional objectives to the design of the table were that it is able to support the weight of the robot statically, provide a full enclosure of the robot's work envelope during operation for safety, provide minimal dynamic motion during operation of the robot, provide the ability to securely fasten the table to the ground during operation as well as be capable of being repositioned through the work of three or four people to fulfill the request of Torbjorn Bergstrom, the Operations Manager of the Manufacturing Labs.

Figure 18 - Final Support Frame Design

The design seen in Figure 18 is simple, utilizing three levels, four feet square, at the ground, three and a half feet, and at seven feet, all welded to the verticals of the table. The top level supports the weight of the robot, and is braced at the corners with pieces of steel to provide rigidity to the entire table. The middle level provides a work surface for mounting of components for project work to be done by students in a laboratory setting, particularly for pick and place operations. The bottom level is flush with the floor, giving a good location for securely fastening the table to the ground.

7.2 Electrical System Design

Some of the key aspects of the electrical system design and implementation were supplying power to the whole system, the wiring of the communications network, and design and construction of an external interface board. There were a number of concerns that were tackled during the development of the electrical system, such as ensuring that the communication protocol could have the speed necessary to control a highly dynamic system. The project accomplished many goals, these include creating an open-architecture platform that allows the users to interact with the robot in a number of ways, either through a GUI and jog functions or through an interface board where the user can input signals and pull out feedback.

7.2.1 Electrical System Requirements

Some of the requirements of the whole system include an open-architecture, movement at 100G, and the ability to be smoothly integrated into the Industrial Robotics curriculum. These requirements were then analyzed to ensure the electrical system fulfilled these requirements while still ensuring it helped the system as a whole satisfy these requirements. The first system requirement was that the robot have an open-architecture, this was done by giving the students two options on how to interface with the robot. A student can choose to interact with the robot

through the GUI or though the external interface board. By incorporating the interface board, users are allowed to get digital feedback from the system as well as input analog and PWM signals for more hands-on control over motor position.

The second system requirement which was central to the project was designing and implementing a system which moves at 100G. The electrical component requirement was to set up a communication network which would allow for information to be sent fast enough to keep an extremely dynamic system moving synchronously at such a high acceleration. Lastly, an essential requirement of the system is that it must assimilate into the Industrial Robotics class. This was done by designing and creating the external interface board. The Industrial Robotics class utilizes the Fanuc 200iB robotics arm which allows users to add sensors and get signal feedback by using a National Instruments Data Acquisition Module (NI-DAQ), by creating the external interface board this feature of the Fanuc robot was emulated making it possible for easy incorporation into the curriculum.

7.2.2 Electrical System Layout

Within the electrical portion of the project, there were many large accomplishments; these include the wiring and powering of the whole system as well as the design and creation of the external interface board. Figure 19 provides a schematic of the whole electrical system.

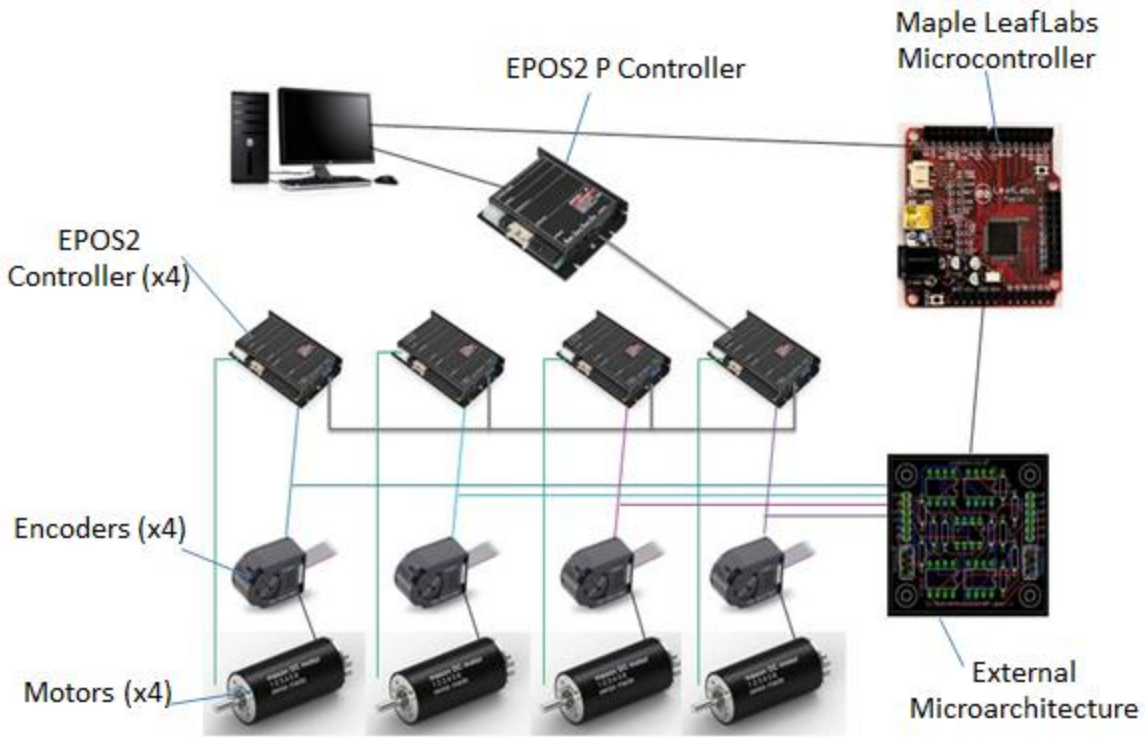


Figure 19: Electrical Schematic of Entire System

When wiring the system, sending power to the motors and controllers is the first concern. This is done by using a 24V power supply connected to a barrier strip, where all four motor controllers were also connected. By connection the system in this manner, the power supply could be placed anywhere around the robot's area and still supply each motor controller with 24V. One issue that arose was how to power the EPOS2 P which would be destroyed if run on 24V. This was solved using a computer power strip, by cutting the end where the cord usually connects to a computer and soldering a power connector to the cord, a power cord supplying 17V was able to be used to power the EPOS2 P. The next largest issues were how to wire the communications cables and how to include the external interface board in the system's schematic.

The EPOS2 P and four EPOS2 controllers all communicate through CAN and are connected in daisy chain fashion. This causes commands to be sent to one controller after the

other, nonetheless by using the EPOS2 P coordinated motion was still achieved using this protocol. The EPOS2 P connects directly to the PC though USB and functions as a master device while the other controllers function as slaves. After the computer, motor controllers, and motors were connected and communicating the external interface board had to be integrated into the schematic.

7.2.3 Embedded Microcontroller Selection

The embedded systems portion of the project included selecting an embedded platform that was extensible and comprehensive, designing an interface board that would shield electronics, and developing software libraries that would be used on the embedded platform.

The selection of the embedded platform included reviewing and understanding what was available as well as taking into consideration the differences of the embedded system vendors. The Arduino microarchitecture was selected preliminarily because of its open-source architecture and extensibility. Arduino is an open-source electronics platform that is focused on flexibility and ease of use. It also has a large community that contributes to both the hardware and software libraries. The community continually adds and supplements hardware technology and the associated software to the collection of sensors and displays that are in the market. For instance, there are GPS, camera, Ethernet, and infrared sensors that can be easily added to the Arduino based family of boards. Some of the extensions are often housed in daughter boards also referred to as Arduino Shields.

Reviewing the extensive set of hardware and software libraries that the Arduino community has developed was only part of the analysis. The capability of the embedded microarchitecture was also an important factor that was researched. This analysis was primarily focused on the performance, the size of the data bus, and the availability of peripherals such as

UARTS and DAC. For instance, the Arduino community is predominately focused on 8-bit microcontrollers, while other microcontrollers can use 16 or even 32-bit operators. For this reason it was felt that an Arduino board may not have the performance necessary for the system; however, finding a community and embedded vendor that could offer the same open architecture as Arduino was still an important goal.

The 32-bit microcontroller embedded boards are not as common when compared to the 8-bit microcontroller space, but many vendors have embraced making Arduino compatible boards. Some of the analysis that was performed during the selection of an embedded board vendor centered on the following capabilities and functionalities:

- 8 bit versus 32 bit
- Availability of hardware boards and shields
- Clock speed
- Number of I/O ports
- Number of timer interrupts
- Number and size of the counters
- Serial communication
- ADCs and DACs
- Software compatibility
- Arduino compatibility

Using the list of capabilities and functionalities, a comparison table was devised on the boards that available within the budget of the project.

Table 1 - Comparison of ATMEGA and LeafLabs board

Functionality	Arduino ATMEGA	Maple LeafLabs
Performance	16 MHz	72 MHz
Core	8-bit	32-bit
RAM	8KB	128KB
Serial Communication	USARTs, limited USB	USB and USARTs
Timers	8/16-bit Timers	16-bit Timers
PWM	8-bit PWM	16-bit PWM
ADC	10-bit	12-bit
Digital I/O	14	9
Analog Input Pins	8	15

After reviewing the capabilities, the Maple LeafLabs board was chosen. The Maple board is Arduino compatible, but it relies on the STM 32 family of microcontrollers based on the 32-bit ARM Cortex-M3 architecture rather than 8-bit Atmel AVR chip. The vendor claimed to be software compatible, but there is always a difference due to the difference in chipsets. The Maple board was also compatible with the Arduino Shields on the market which was another benefit since the team was targeting the fabrication of an extensible robotic platform. The selected Maple LeafLabs board is displayed below.

Figure 20 - The Maple LeafLabs Board based on ARM Cortex-M3 Architecture

The Maple board would interface with a host PC. This board can interface with a variety of operating systems such as Windows, Linux and Mac OS X. The lab PC has the Windows 7 64-bit operating system. Software would be compiled on the host PC and loaded onto the Maple board via a USB connection. A USB-to-mini USB cable is used for host PC to Maple communications and even power. Maple provides an integrated development environment (IDE), called the Maple-IDE [11]. This package contains a compiler, upload utility, software library, text editor, example code, and a serial monitor utility. The bundle has been tested on Windows XP, Linus Ubuntu 10.04, and Mac OS X. There was additional installation work to ensure that it would function on the Windows 7 64-bit operating system, which is the currently supported operating system at Worcester Polytechnic Institute.

The programming language used for the Maple LeafLabs board is Wiring which is the same language used for the Arduino boards. The syntax is like C/C++; and in fact, Wiring is simply a wrapper for C++. Unlike C/C++ where every program has a main() function, with Wiring, setup() and loop() functions are required. The function, setup, is only run once and is used to initialize variables, configure general purpose I/O as in the case of the pin mode

instruction, set up interrupt handlers, and access libraries. The loop function gets called repeatedly, thereby allowing the program to respond and react. Projects in Wiring are called sketches.

One key capability that LeafLabs board, namely the ARM architecture offered is the ability to read PWM signals and determine the period and duty cycle of the submitted signal. This functionality is a result of the extensive timer capabilities. The board provides 8 timers with timer 1 and timer 8 being advanced timers and the others providing general time functionality. To leverage this capability the team developed a program to configure the board and calculate the period and duty cycles of submitted signals. The ARM Cortex based microcontroller that the Maple LeafLabs board uses is the STM32F103RB microcontroller developed by STM Electronics. To develop the program, the reference and data sheets were extensively consulted to ensure that the microcontroller was properly configured

To measure the period and duty cycle of a PWM signal, the timer must be configured as PWM input mode. This is a special case of the input capture mode. The input capture mode refers to latching the counter value when a transition is detected in the input. The PWM input mode will latch both transitions: the one from low to high and the other from high to low. In order to latch both types of transitions, the timer needs to be an advanced timer capable of performing this and it needs to have a slave counter as well. Because of these requirements, the timer that was chosen was timer 1, which offers both capabilities.

The Maple LeafLabs libraries did not specifically support the hardware timer functions that were needed. What is meant by support refers to the Maple IDE providing functions, settings and libraries that perform the tasks needed. For instance, they support OUTPUT_COMPARE but not INPUT_COMPARE. The support library function,

OUTPUT_COMPARE generates a pulse when a certain value is reached. Even though there was no configuration or library setting within the Maple IDE, the ARM chip could be programmed with the necessary settings. This means that bit masking and setting had to be done directly versus using a canned library to configure the timer. The hardware timer functions were clearly flagged on the website as being in a state of change and being incomplete.

To configure the Maple LeafLabs board so that microcontroller would be in the PWM input mode, the following steps were taken to ensure that the period count was in the first capture register and the counts associated with the duty cycle are placed in the second capture register:

- Select the Timer 1 Input in the Capture/Compare Mode Register linked to the counter of period
- Establish the polarity for the signal that triggers the capture register associated with the period count. Polarity for this capture register is rising edge.
- Select the Timer 1 Input in the Capture/Compare Mode Register associated with the counter of duty cycle
- Establish the polarity for the signal that triggers duty cycle count with the falling edge of Timer 1 Input
- Select the proper filtered trigger in the Slave Mode Controller Register
- Configure the Slave Mode Controller Register in reset mode
- Enable the capture registers – one for period and another for duty cycle
- Enable an interrupt per capture register

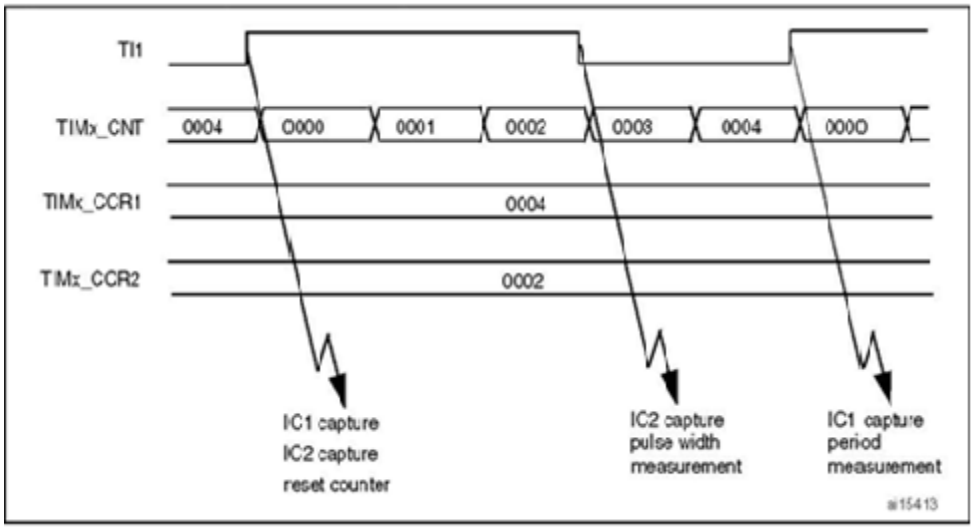


Figure 21 - Timing Diagram of the PWM Input Mode

The timing diagram depicts how the Timer 1 Input (TI1) is the input and how Capture/Compare Registers 1 and 2 contains the counts associated with the period and duty cycle. The counts also rely on the pre-scalar and clock frequency of the board. The diagram also shows how interrupts are fired with transitions of the timer 1 input.

The above procedure refers to configuring the Maple LeafLabs board to measure duty cycle and the period of a PWM signal, the pre-scalar and the timer clock frequency were configured and overflow was accounted. The chip's operating clock functions at 72MHz. A pre-scaled factor can be defined to divide the clock frequency into more desirable operable timer counter cycles. An overflow refers to where the limit of the counter will reach before overflowing. An interrupt was also set up in case of an overflow scenario. Overflow is not expected in this case. The upper limit of $2^{16} - 1$ was set up as the overflow.

To validate the embedded application, a test PWM signal was developed. A test signal, consisting of a PWM signal having a frequency of 100 Hz and a 50% duty cycle was used to prove the hardware. A pre-scalar of 288 was used against 72MHz clock, which resulted in a 250

kHz clock signal. The test signal was generated on Pin 7 and fed to Pin 6 via a jumper cable. Pin 6 is the input associated with the timer 1 input. The software would display the counts associated with the period and duty cycle using the Serial Monitor application provided by the Maple IDE. The SerialUSB function was used to pipe the counts out to the Serial Monitor application. The counts that the application displayed for the given test PWM were 2510 for the period and 1255 for the duty cycle counts. These values correspond to a 100 Hz PWM with 50% duty cycle using a 250 kHz clock.

7.2.4 Interface Board

The external interface board connects to the computer through USB on the Maple LeafLabs board. The external interface board was designed as a shield or daughter board for the Maple board; through this board, users can input PWM signals to control the position of the motors. After the PWM signal is imputed into the interface board, it is sent to the Maple, and then sent over USB to the computer, which then sends position information to the motor controllers based on the input signal. Moreover, the interface board connects to the encoders by connecting to a ribbon cable which is pressed onto the encoder wire that is attached the encoders. By pressing the ribbon cable extender onto the encoder cable, encoder feedback can be pulled out and used as available feedback through the interface board.

Although the external interface board was integrated in to the larger electrical infrastructure of the system, the board's design and implementation were also a major accomplishment of the project.

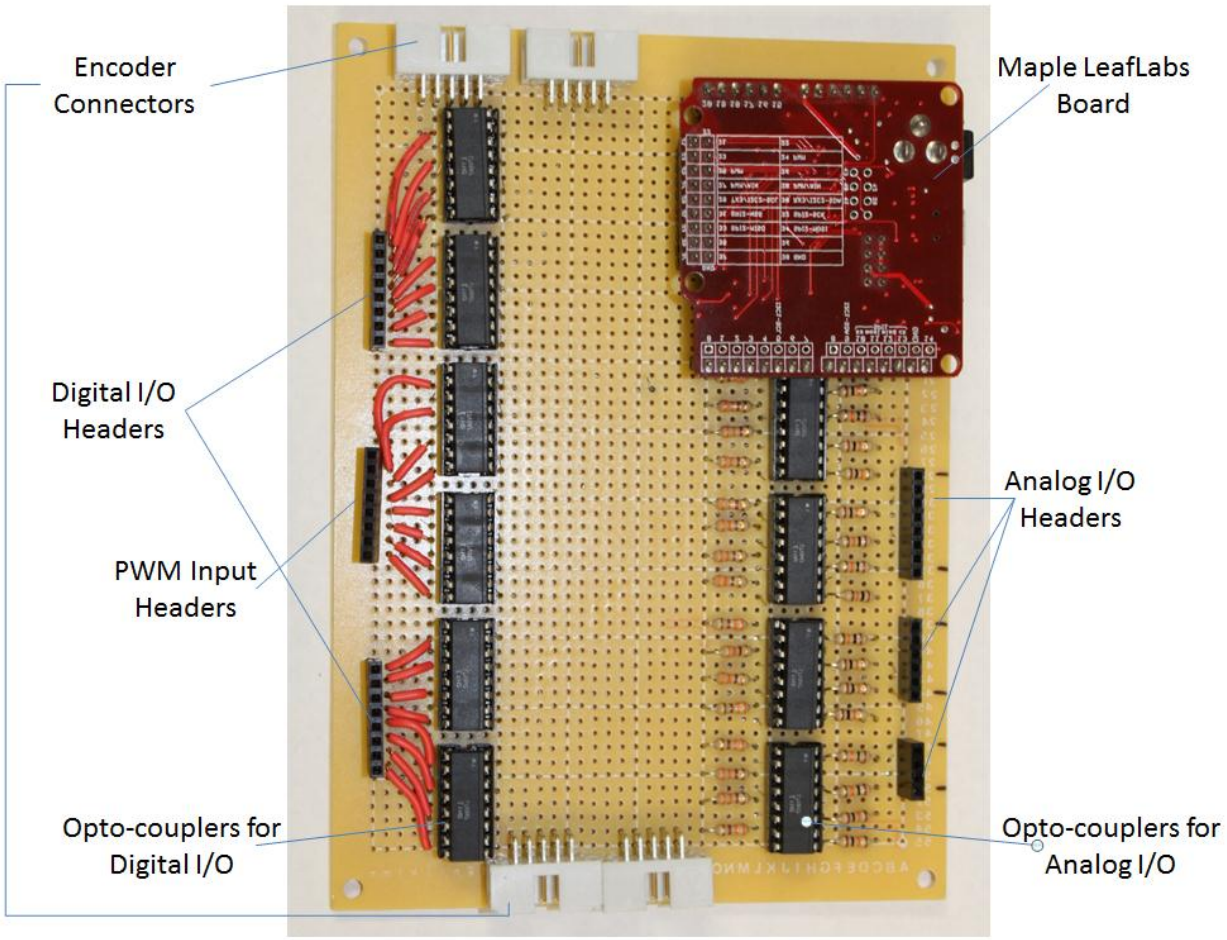


Figure 22: External Interface Board Layout

As can be seen in Figure 22, one of the major components of the board is the opto-couplers used to isolate the signals being inputted and outputted to the system. These allowed for no electrical connection to be made between the headers available to the user and the wires going to the embedded board. Using the opto-couplers was one key manner in which the electronic components were shielded from harmful signals injected into the system. As well, when looking at the top of the interface board only the back of the Maple LeafLabs embedded system can be seen, this is due to how the interface board connects to the Maple board. The interface board has pins which correspond to certain headers in the Maple board, what is seen in Figure 22 is actually the Maple fitting perfectly with the external interface board. Next, what can be seen is

the headers on either side of the board. The board was designed with all the headers available to the user located towards the outside of the board and all the wires to the embedded board running up through the middle. This allowed for all the opto-couplers integrated circuits (ICs) to be oriented in the same direction which is a standard electronics design practice. The right side of the board is where all the digital I/O is available to the user, while the analog I/O is all located on the left side of the board. This allowed for easy streamline design of the board, which would permit for ease-of-use for any user utilizing the external interface board. By completing the external interface board, a second mode of interaction with the robot was made available to the users; this is a significant task that needed to be accomplished to allow for easy integration into the Industrial Robotics class.

Since the robot is intended for classroom use, it is necessary to protect the system from harmful circuitry that could be connected to the system. To ensure that the custom electronics created for the robot were project, much research was done to find a method to completely isolate the signals being input into the system. It was then decided to utilize opto-couplers also known as opto-isolators. Opto-couplers are used in electronics to isolate circuitry from potentially harmful outside circuitry. The isolation is accomplished by the use of a light source and a photoresistor. The voltage applied to one side of the circuit is converted to a beam of light thereby preventing any high voltages such as voltage spikes from one side of the circuit from harming the other side of the circuit.

A typical opto-coupler consists of a power source, a LED (light emitting device), a closed channel, and a device sensitive to light, such as photosensor. Typical examples of photosensors are photodiodes or phototransistors. There is no electrical connection between the LED and the photosensor other than light. The input voltage is the opto-coupler's power source and is utilized

to generate a beam of light via the LED. This beam travels down the closed channel until it reaches the photosensor. The photosensor will convert the light back to electricity. This electricity powers the second side of the circuit.

Opto-couplers can be utilized to isolate analog or digital signals. The problem is that opto-couplers tend to be non-linear devices so they are better suited for digital signals. Although they are not as well suited for analog signals, they are still an inexpensive way of isolating signals. In this case, the key focus is isolating signals such as an analog signal being sent to drive one of the motors, which students may attempt to inject as well as removing electrical interferences such as power surges.

In this project, the LTV-847 opto-coupler was selected. It provides four channels per device meaning it will isolate four different circuitry pairs. It also provides a high degree of voltage isolation, namely 5,000 V_{rms}. The device also offers a cutoff frequency, f_c, of 80 kHz with rise, t_r, and fall, t_f, times of 4 and 3 µs. These characteristics are associated with an R_L of 100Ω. R_L is the resistor that serves as a pull-up resistor for the output voltage. In practice, there is a tendency to use a higher value for the pull-up resistor values. After setting up the circuitry and working with higher pull up resistor values that would still yield a suitable cutoff frequency, a pull-up resistance of 330Ω was selected. Typically, pull-up resistors for transistor circuits tend to be 1000-5000 Ω, but that would result in an inadequate cutoff frequency as denoted by the frequency response diagram, Figure 23 - Frequency Response. The 330Ω provided both an adequate cutoff frequency and rise and fall times as detailed by Figure 23 - Frequency Response and Figure 24 - Response Time vs. Load Resistance.

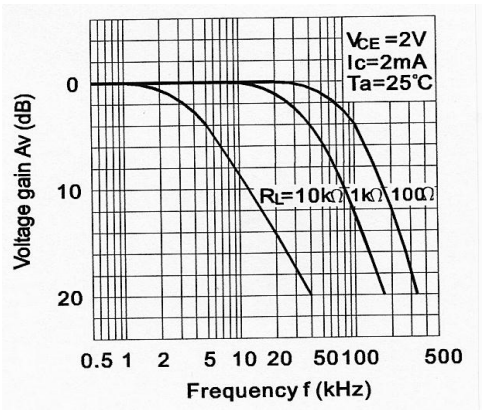


Figure 23 - Frequency Response

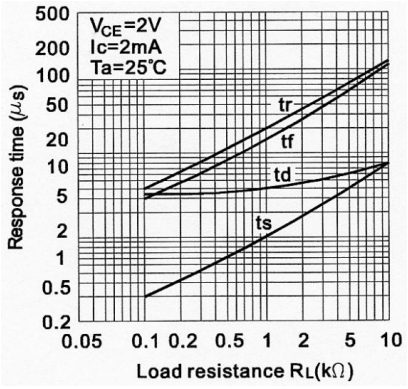


Figure 24 - Response Time vs. Load Resistance

The LTV 847 datasheets were used in the design of these circuits and were validated using the test circuits shown in Figure 23.

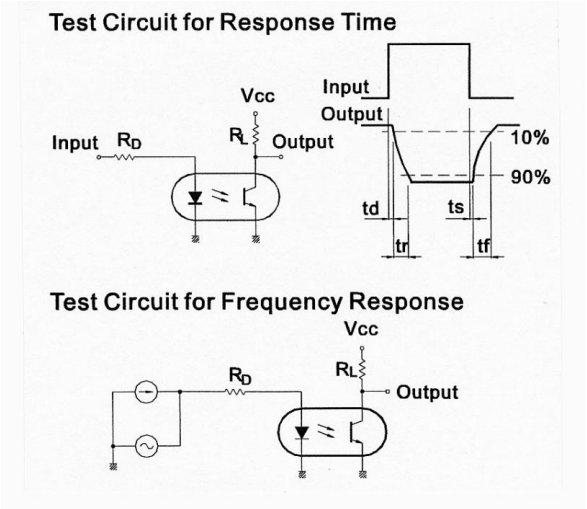


Figure 25 - Test Circuits for the Opto-Coupler

8 Software Architecture

The goal of the project from a computer programming perspective was to create an open software architecture capable of operating in multiple modes, depending on user preference. Achieving this goal required the creation of an extensive application programming interface (API) and touchscreen graphical/natural user interface (GUI or NUI) for the robot, which would provide the functionality most desired by WPI robotics students and faculty in a natural and accessible way.

8.1 Computer

The robot control program is stored on and run from a PC, which communicates by USB with the main EPOS2 P controller. This controller in turn communicates with 4 subordinate EPOS2 controllers, each of which is in charge of managing the motion of a single motor. A touchscreen monitor allows users to interact more naturally with the control interface, but a mouse and keyboard can also be used.

8.2 Modes of Operation

In keeping with the project's design philosophy of open architecture, multiple ways of interfacing with the robot are provided for users. In addition to the GUI, the interface board allows users to communicate directly with the robot using their own PC or controller.

8.2.1 Graphical User Interface

The GUI was developed using NetBeans GUI Builder. The right side of the interface has information relevant to the current position and orientation of the robot. The left side contains movement controls; the user can use tabs to switch between jog motion controls and coordinate input controls. Additionally, the coordinate control tab contains text fields to allow the entry of an alternate origin, and buttons to allow users to string multiple movement commands together

into a single program. The interface was developed in parallel with our API, tying new functionality to its corresponding buttons/menus/etc. as it was created. Both views of the GUI can be seen below Figure 26 and Figure 27

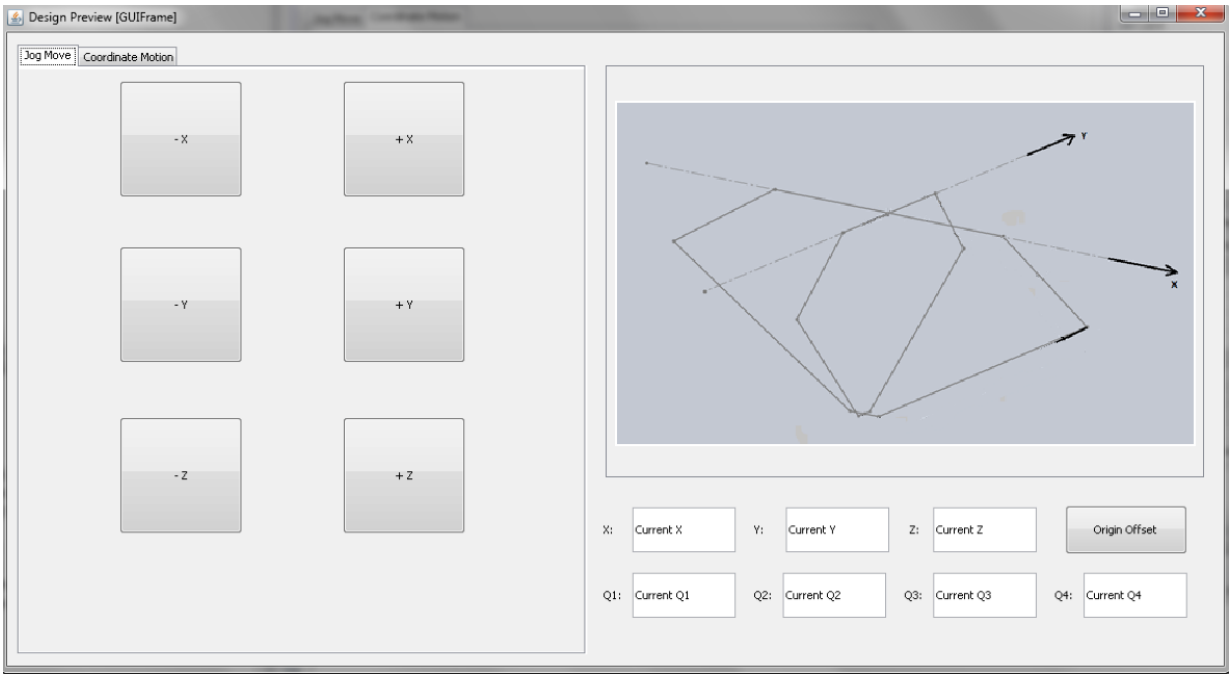


Figure 26: Jog Motion GUI

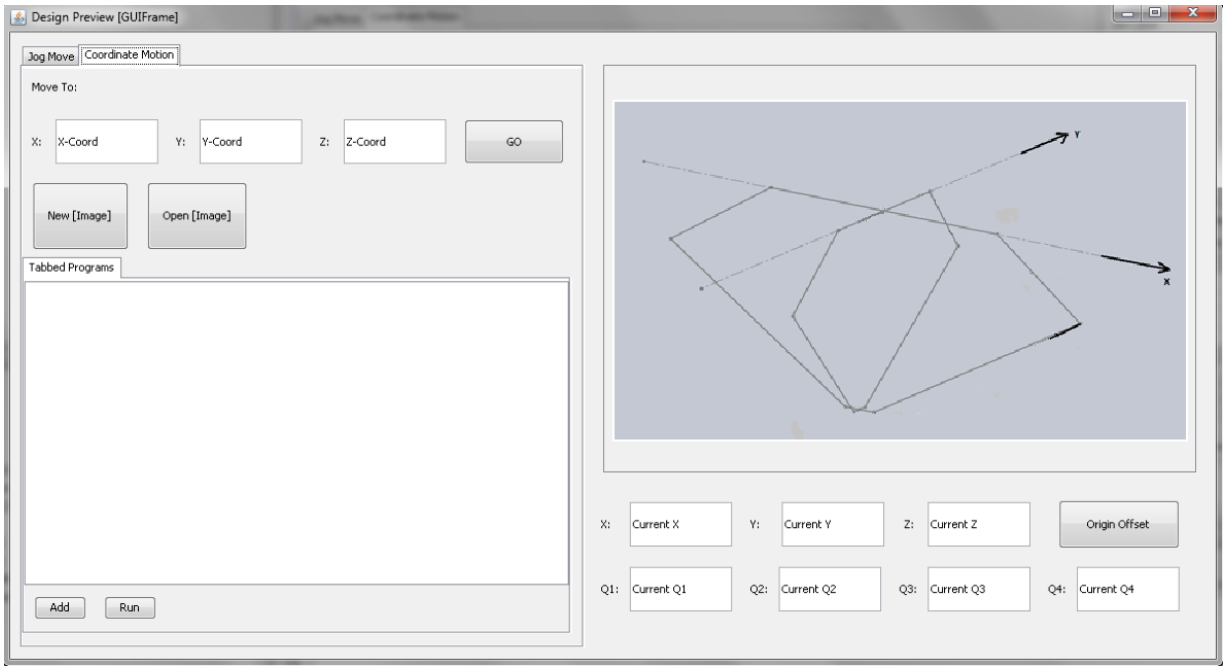


Figure 27 - Coordinate Motion GUI

8.2.1.1 GUI Design considerations

Members of the WPI Computer Science faculty were asked for advice on how to create an effective and robust user interface. This feedback was used to generate use cases and class and state diagrams as helpful planning tools, and the information from the professors stressed the importance of an iterative design process to ensure we accomplished as many of our goals as possible as completely as possible.

In an effort to develop an effective set of use cases, we spoke with professors and students in the WPI Robotics Engineering and Mechanical Engineering departments to determine what functionality they would find most valuable in a parallel kinematic robot. A number of common requests were identified, from the simple (movement commands, end effector feedback and control, alternative coordinate systems) to the complex (writing programs; setting/getting velocity, position and time between movements; writing logic). The list that was created helped outline specific programming milestones that were most desirable while still being achievable in the time frame available. The final list of important milestones, ranked in order of importance is:

1. Implementing kinematics equations derived during mechanical system analysis (including limits of the work area)
2. Jog movement
3. Movement speed regulation (as a percentage of maximum system speed)
4. Movement to specified XYZ coordinates
5. Position and orientation feedback from the end effector
6. Ability to string several sets of movement coordinates into a single program
7. 3D representation of our robot in our GUI
8. Feedback from encoders, including joint angles

9. Set and get velocity, position and time between movements

10. Ability to write higher level logic for the robot to obey

Additionally, common testing procedures were researched to identify those that would be most useful in determining how well the goals that were set would be met, and how well the system performed in a more general sense. The three testing types that were found to be most useful were scalability testing (testing large or very complex data inputs), soak testing (working the system for an extended period of time) and stress or load testing (essentially, trying to "break" the system by issuing multiple simultaneous or conflicting commands). Specific testing procedures were further outlined for a number of the milestones.

8.2.2 Touch Screen Considerations

Since the API was developed primarily in Java, Swing was used to develop the GUI. Swing has no "touchscreen library," per se, so a number of freely available mouse gesture libraries were researched, mainly iGesture and Smardec's Mouse Gestures. Since, on the touchscreen purchased for the project, a "touch" is equivalent to a mouse click, these libraries would allow us to incorporate more complex motions into the GUI for more advanced control of the system.

The major consideration that differentiates touchscreen interfaces from traditional mouse and keyboard GUIs is that the former requires larger controls and greater tolerances. Mice are far more precise pointing devices than fingers, and any touchscreen must take this into account. The GUI developed for this project uses oversized buttons and text fields to account for this difference.

8.2.3 External control

Other than the GUI, the user can control the robot's motion and position using the external interface board. The board has the capabilities to take in PWM signals, send them to the computer, and move the motors to a desired position based on the input PWN signal. To do this, code which accepted a PWM signal input and read the signal's period and duty cycle had to be written; the period and duty cycle of the signal correspond to a certain motor position. Once the Maple LeafLabs board reads the period and duty cycle, it sends this information to the PC. The robot has an allowable workspace, which translates to a collection of allowable end effector positions, using inverse kinematics the corresponding motor positions are found for all accepted end effector positions. The motor position calculated from the input PWM signal is then compared with the collection of permissible motor positions within the workspace. If the desired motor position is within the allowable workspace the motors move to the new positions; however, if the desired motor position is not within the workspace the PC does not allow the motors to move to the new position.

8.3 Open-Architecture API

The first step in developing the code to run our robot was programming on our prototypes. This was very simple, proof-of-concept programming in Java, consisting of a simple sequence of move commands. While very basic, it provided valuable insight for future work, most usefully in the intricacies of getting multiple motors to work cooperatively. This was part of the reason the four-motor design was selected over the three-motor design– it was felt that it would be easier to program two sets of opposing motors working more or less in tandem than three completely independent motors.

After the prototyping work had given a rudimentary idea of the size and nature of the goal, extensive research of API, GUI and software design best practices as they applied to the project. It was decided to work primarily in Java and C, as these were the languages that provided the most functionality for both the GUI and the motor controllers

The actual writing of code for the PKM was a complicated task. The program needed to perform complex inverse kinematic calculations to determine motor position for each point the robot moved to. Not only that, all four motors had to perform their motion commands simultaneously to ensure smooth movement and prevent damage to the system.

A basic class diagram of the program can be seen in Figure 28. All of the kinematics calculations are done in the abstract AbsKinematics class with the calculateAngles method; provided a desired set of (x, y, z) coordinates, it calculates the angle below horizontal that each upper arm needs to be set at to place the end effector at those coordinates. The subclass RealRobot can then take the angular value produced and convert it into a position value that the motor encoders can interpret. The initial plan was to create a VirtualRobot class that would also extend the AbsKinematics class to create a virtual representation of the robot in our GUI. Unfortunately, due to time constraints and general lack of experience working in 3-D modeling languages, this goal was forced to be abandoned for this iteration of the project.

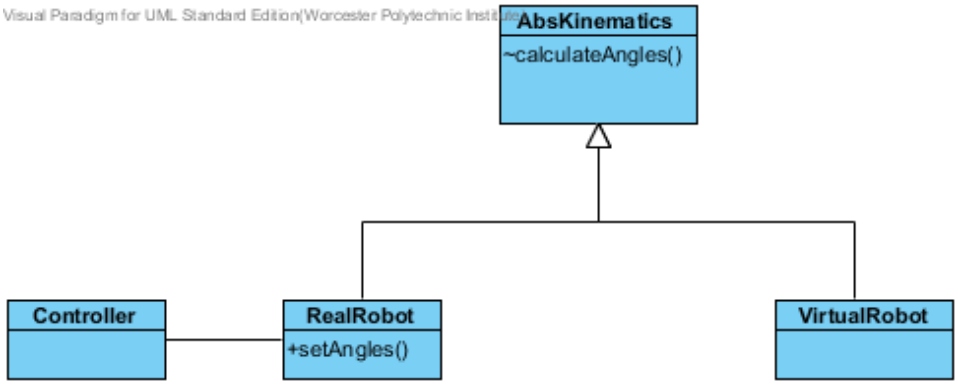


Figure 28 - Class Diagram

One problem that was encountered after the controllers had been selected was that the EPOS controllers are not capable of handling commands in Java, only C++ (for which Maxon has created an extensive command library). As such, intermediate, "wrapper" classes had to be created in the Java framework, which would allow for C++ commands to be called.

The synchronization aspect of the robot's motion was the most difficult functionality to implement. Extensive work was done with EPOS Studio, proprietary software provided by Maxon for programming the EPOS controllers in order to understand the EPOS communication protocols and how to best approach synchronization. A program was adapted from a sample program provided by Maxon, to automatically coordinate motion between the motors.

9 Project Summary

The following section provide the accomplishments of the project and other considerations including economic, health/safety, and reliability along with social impacts and use of standards as observed in the design and realization of the system.

9.1 Project Accomplishments

The mechanical portions of the robot have been completely manufactured and assembled. A steel frame was welded together, but will not serve as an acceptable long-term mounting platform for the robot. The steel used to construct the frame is too thin to properly support the robot during operation without significant vibration. There is some backlash concern in the motors, which allows the end effector to wiggle. Some method of biasing the gearboxes to one side of the gear teeth, possibly with springs, would need to be implemented to solve this problem. The project's limited budget did not allow for re-machining of parts, so any imperfections that occurred during machining simply became a part of the final project.

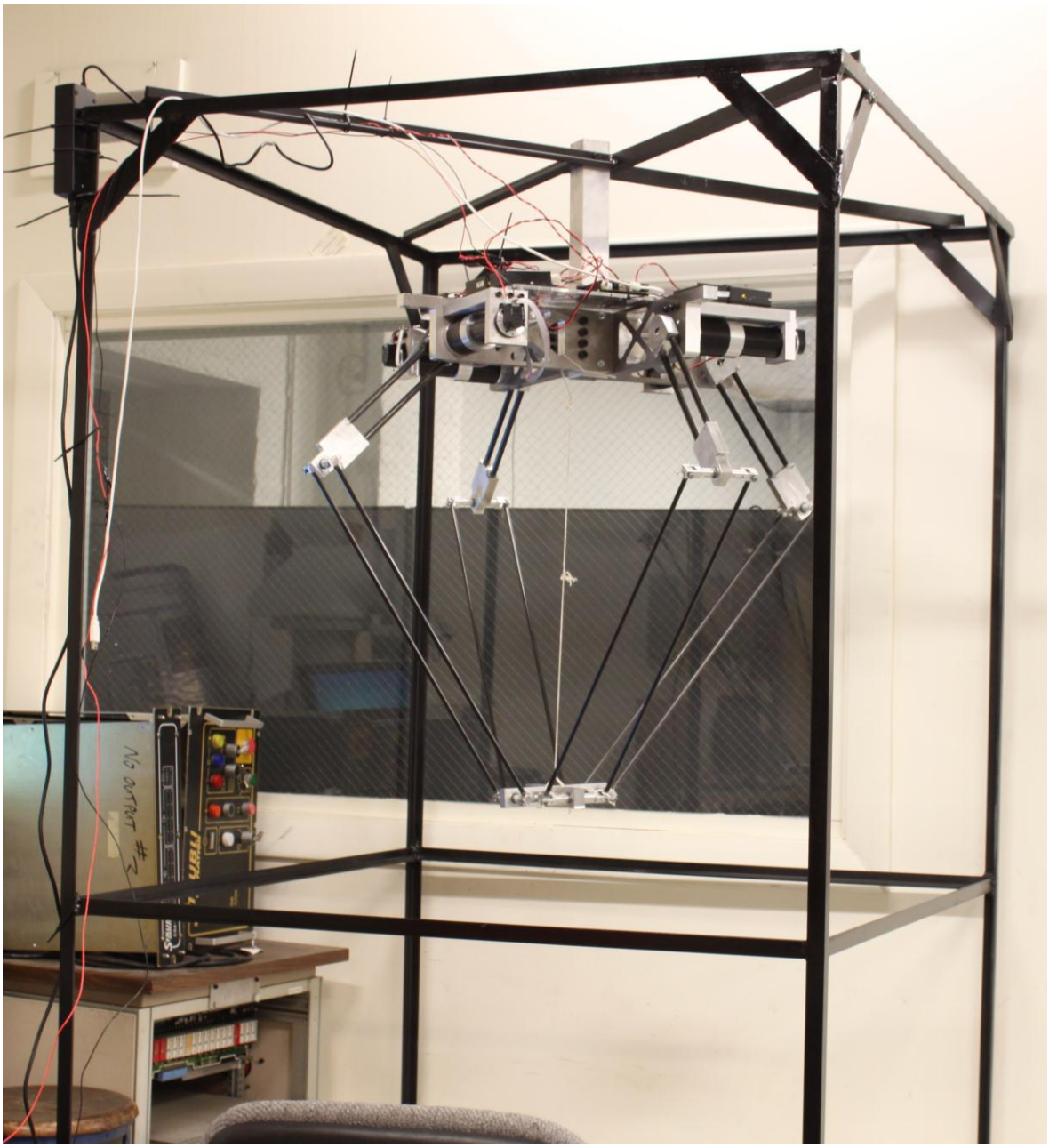


Figure 29 - Final Completed Assembly

The robot is currently operated using a program written with Maxon's proprietary EPOS Studio software. While an API and GUI have been created for the robot, they did not see use during the course of the project. The GUI needs to be made to communicate with the API, and C++ wrapper classes need to be written to allow the API to control the robot.

The electrical system has had a majority of the external interface board completed, and only requires several more connections to be made between the encoder headers and the opto-isolators as well as between the opto-isolators and the microcontroller. All of the wiring on the robot itself has been essentially completed, and only requires some neatening up into proper wiring harnesses.

9.2 Economic Considerations

One of the biggest challenges when implementing this system was the economic impact of both building it and keeping it in operation. Since this robot will be used as a learning tool, it was necessary to keep costs to a minimum while still create a robust, effective system. For this reason many of the parts were custom machined out of stock or scrap metal. The most expensive components are the motors and motor controllers which were donated. Although the robot was successfully built for approximately $1400, the cost of operation and maintenance is still a significant concern. One of the largest costs that still must be incurred is the cost of making it comply with OSHA standards. Lastly, the costs of operation and maintenance will be costs that the will be incurred by the robot over time for the entire time it is in use, these costs include electrical costs to power and use the robot as well as repair costs for any damaged components. Although this robot poses economic concerns, it has proven to be a lower cost solution compared to the cost of purchasing a proprietary unit.

9.3 Health/Safety Considerations

When building an industrial robot the health and safety conditions are of principal concern. The first issue that must be tackled is making the system compliant with OSHA standards; these include adding emergency stops and light shields to the robot. Some of the safety concerns presented by the system are the pinch points in all the universal joints as well as

the high operational speed of the robot. Also, if the robot moved outside of the work envelope it could cause damage to its area or injury to anyone who is operating it. Although this robot will be an excellent addition to the Industrial Robotics lab, it must be made safe for students to use before it can be integrated into the curriculum.

9.4 Reliability Considerations

There are several important concerns to consider regarding the reliability of the system. While the system is designed to be mechanically very robust, it is intended to be used in a classroom/laboratory setting. This setting exposes it to physical danger from inexperienced or undertrained operators. This can result in breakages of the carbon fiber arms, the end effector platform or the motors and gearboxes. The electrical system can suffer from issues due to the use of brushed motors to power the robots motion. These brushes suffer from unavoidable wear as the motors are used. The motor controllers are also limited in the amount of power that they can provide to the motors, which provides an upper bound to the speed and acceleration of the end effector.

9.5 Social Impact

The aesthetic of the work performed was very important, especially with regard to the user interface components. It is important to consider the ways in which the user might want to interact with the system, and attempt to provide a framework which would allow them the freedom to utilize the PKM in whatever manner they need. With this in mind, the Computer Science Faculty at Worcester Polytechnic Institute were consulted for what they felt made an effective user interface, and their recommendations were implemented into the design of the user interface.

9.6 Use of Standards

During the construction of the robot, the electrical system was the location where the most commercial off the shelf components were used. The EPOS2 and EPOS2 P are available for sale from Maxon to provide power to the motors and collect feedback from the attached sensors. The Maxon RE-Max motors and accessories are also available as off the shelf components, including the attached gearboxes and encoders for feedback to the control system. All of the individual components of the opto-isolator board, including the Maple microcontroller are also available off the shelf, and were purchased from a variety of vendors, including Digi-Key and Radio Shack for this project.

10 Future Work

The following sections provide areas of potential system enhancements as consideration for future work.

10.1 Mechanical Recommendations

While the mechanical system is ready for operation, there are several improvements that can be implemented to increase the capability of the system. In particular, a new frame would contribute to the overall stability of the system. Purchasing larger extrusions of steel to ballast the system would be beneficial, as it would prevent the robot from vibrating the table at the resonance frequency of the supports. Further modifications also should be included to the table, including integration of a light curtain system to remove the possibility of unintended activation while users are in danger.

A further improvement to the mechanical system would be the placement of the robot in a semi-permanent location, which would allow it to be solidly attached to the floor. This system should also be easy to fasten and unfasten to allow for reconfigurations of the manufacturing laboratories, especially during machine tool exchanges with Haas.

Further machine guarding can be implemented using light curtain systems available in the manufacturing laboratories. These will be an important addition to the mechanism, especially while the robot is being utilized by students for a variety of projects in the Industrial Robotics curriculum.

10.2 Electrical Recommendations

Throughout this project significant progress was made in the design and implementation of the electrical infrastructure. The completed electrical system included integration of the power electronics, such as power supplies, motors, and controllers, as well as embedded electronics,

such as the Maple LeafLabs board and the opto-coupler integrated circuits. Although this portion was successful in its completion there are a number of recommendations that can be made to improve the system.

One of the issues encountered throughout the electronics design was the non-linearity of the opto-couplers. The opto-couplers currently on the board work perfectly for the digital input and output isolation; however, when implementing the analog input ports using the opto-couplers on the board became more challenging. Thus one of the recommendations suggested is the use of linear opto-couplers when isolating the analog signals.

The second most important recommendation is to create a printed circuit board for the electronics interface board. This would make the system more robust, as well it would allow for much easier replacement and transfer of parts if issues arise from a user frying or damaging the board. Lastly, due to the issues that arose during design and implementation of the electronics a large amount of testing was not done. Some testing was done on the Maple LeafLabs board to ensure it could take in a PWM and read its period and duty cycle; nevertheless, it is testing of the whole implemented board was not done. Any testing would increase the performance of the system as well as ensuring any difficulties found could be corrected and improved.

10.3 Software Recommendations

The API and GUI currently exist as separate Java archive files; these need to be integrated with each other if further work is going to be performed on this project. A C++ framework to communicate with the controllers must be developed as well, complete with wrapper classes in the current API to allow the Java code to properly control the robot. This code will then need to be rigorously tested with the robot to ensure safe and proper performance. This testing will need to ensure that the robot cannot move outside of its set workspace, and

ensure that position values sent to the motors are accurate to prevent damage to robot components. Additionally, it may be necessary to tweak the resolution of motion commands to ensure that the robot moves at appropriate speeds and along the smoothest possible path.

As of the completion of this project, the API is capable of both jog and coordinate motion commands, speed control as a percentage of max system speed, and origin offset. This is obviously a much shorter list than the full set of use cases generated early in the project. Further study of student and classroom requirements could be useful in refining, clarifying and expanding this set of use cases, and future programming work should be geared towards adding this missing functionality.

11 Conclusions

The goal of this project was to develop an open-architecture, reliable parallel kinematic manipulator for use in the Industrial Robotics course at WPI. The result of this project has been the creation of a capable platform that is expandable, reliable and open source for use in Industrial Robotics curriculum at Worcester Polytechnic Institute. As an operating system, this robot has several economic considerations, including the cost of replacement parts, such as motors, controllers and carbon fiber. This robot, being used as a teaching tool, is subjected to the potential for accidents that could result in breaking one or more delicate components. These components can be very valuable both in monetary cost and in the time required to manufacture replacements.

One of the most important considerations of the completed robot is the potential for operator injury during use of the robot. There are many potential pinch points in the mechanism, and while these are rather easy to see, the potential for injuries always exists and this is a danger. Another consideration is that during high speed operations, any small flaws in the composite materials run a great risk of becoming catastrophic failures.

The parallel kinematic manipulator is designed to be reliable in its positioning of the end effector. The parallel kinematic chains that support the manipulator provide a high amount of stiffness to the entire system, and this interdependence reduces the backlash present in the motors and gearboxes. One important factor regarding the reliability of the completed system is the potential for failures in either the electric or computer systems. As it has been pointed out earlier in this report, extensive testing is required before this system can be proven to be reliable.

Extensive kinematic and mathematical analysis of three- and four-arm parallel robots granted an understanding of the pros and cons of each, and of the effect on system performance

of the manipulation of various robot parameters and relative lengths. Once the four-arm design was decided upon, and the ideal robot configuration to achieve project goals was determined, SolidWorks models were created. These models provided a blueprint for the machining of parts for mechanical assembly, almost entirely from carbon fiber and aluminum to minimize weight. Motors and controllers, generously provided at a very steep discount by Maxon Motors, were the final components necessary to complete the assembly. An electrical infrastructure necessary to power the system, carry out the communication between the controllers, and an external interface board which allows students more low level control of the robot was realized. An application programming interface (API) and graphical user interface (GUI) which are run from a dedicated PC with touchscreen monitor were developed to allow control of the robot.

At the conclusion of the project, all mechanical components were completely manufactured, assembled, and integrated with electronic components. The electrical system was created that successfully powers the system and allows users to pull out digital feedback and input PWM signals for controlling the motors. The programming architecture was not fully implemented, but it should be possible for the current API to control the robot with some additional coding work to allow proper communication between the PC and the robot.

The conclusion of this project was successful, although there are a number of improvements that need to be made for it to be ready to use in a classroom setting. This project will be a great addition to the Industrial Robotics class and will be extremely helpful to students learning inverse kinematics for the very first time. The robot was built for a fraction of the cost of a proprietary unit and had many of the features stated by Mechanical Engineering and Robotics Engineering students as most important in the development of a new robot for the class.

These were two of the essential design requirements of the system and the robot far surpassed those basic requirements.

Bibliography

[1] Anonymous. “Delta robots are designed for PACKAGING.” Packaging Digest. Volume 43. Issue 8. (Aug 2006): P3-P4.

[2] Bilstein, Miriam. “Optimizing Delta Pick-and-Place Robot.” *Design News Supplemental Packaging*. Volume 62. Issue 12. (Sept 3, 2007): S20.

[3] Brogárdh, T., S. Hanssen and G. Hovland. Application-Oriented Development of Parallel Kinematic Manipulators with Large Workspaces. 2005.

[4] Carricato, Marco. “Fully Isotropic Four-Degrees-of-Freedom Parallel Mechanisms for Schoenflies Motion.” The International Journal of Robotics Research. Issue 24 (2005): 397-414.

[5] Gosselin, Clément. *Kinematic Analysis, Optimization and Programming of Parallel Kinematic Robotic Manipulators*. (Doctoral Thesis) Retrieved from Dissertations and Theses database, McGill University. (1988).

[6] Müller, A. and P. Maißer. “Kinematic and Dynamic Properties of Parallel Kinematic Manipulators.” Multibody System Dynamics. Issue 5. (2001): 223–249.

[7] Ng, C. C., S. K. Ong, A. Y. C. Nee. “Design and Development of 3-DOF Modular Micro Parallel Kinematic Manipulator.” International Journal of Advanced Manufacturing Technology. Issue 31. (2006): 188-200.

[8] Shi, Jinbo Zexiang Li, Yuanquing Wu. “A New Quantitative Performance Index for Low Mobility Parallel Kinematic Manipulators’ Accuracy.” 2011 IEEE Conference on Robotics and Automation. May 9-13, 2011. 2745-2750.

[9] Staicu, Stefan. “Dynamics of the Spherical 3-UPS/S Parallel Mechanism with Prismatic Actuators.” Multibody System Dynamics. Issue 22. (2009): 115-132.

[10] Zhang, Dan. *Parallel Robotic Machine Tools*. Springer Science and Business Media. New York. 2010

[11] The Maple IDE. http://leaflabs.com/docs/maple-ide-install.html#download.

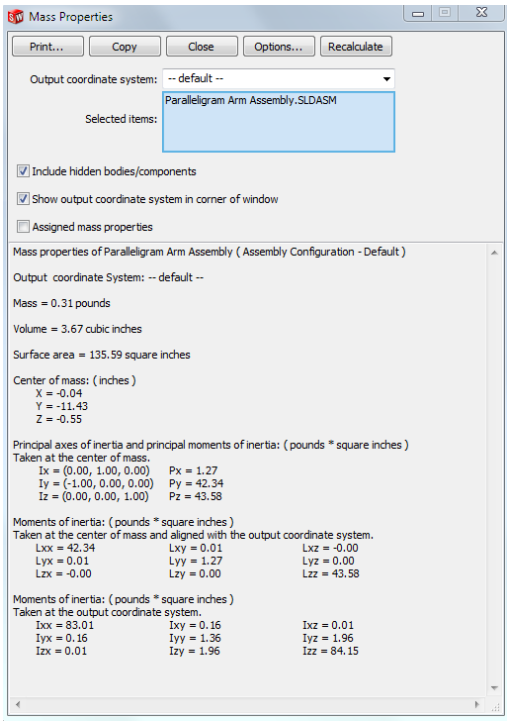
[12] STM32F103RB datasheet

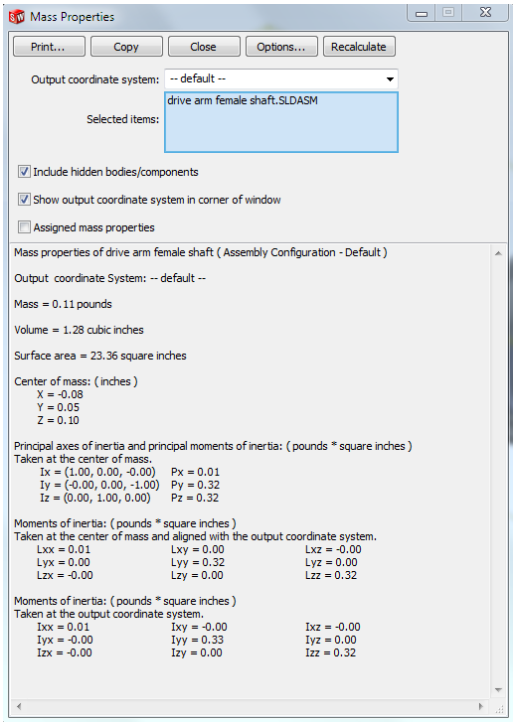
http://www.st.com/internet/mcu/product/164487.jsp#DATASHEET

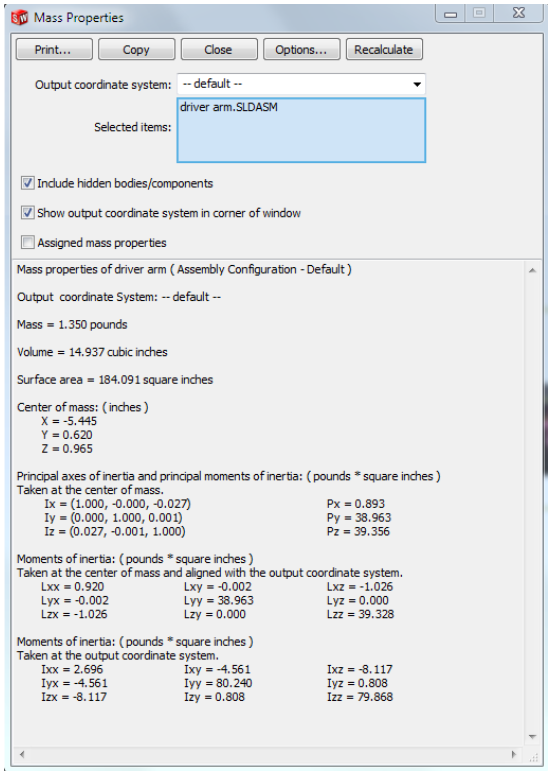
[13] STM32F103RB datasheet

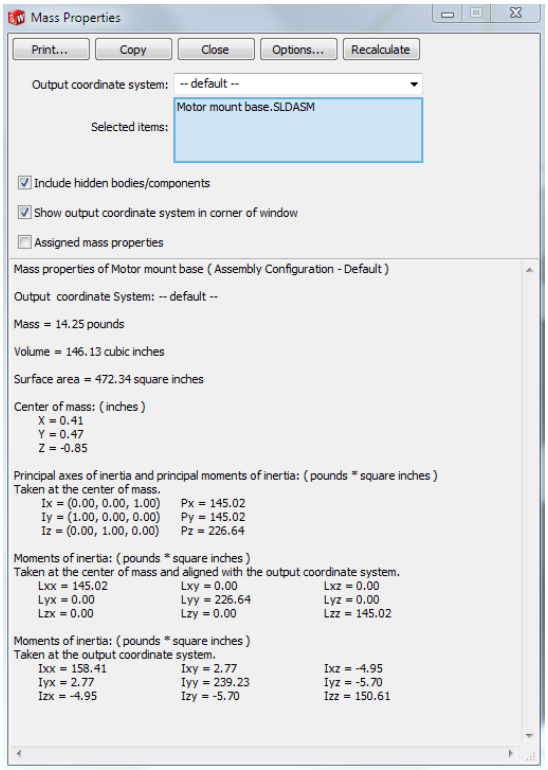
http://www.st.com/internet/mcu/product/164487.jsp#REFERENCE_MANUALS.

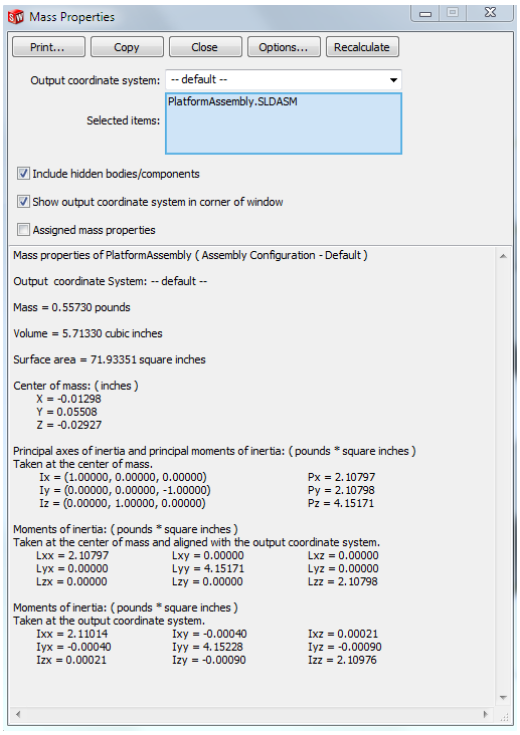
Appendix A – Weight Analysis of Subassemblies






Mass Properties
Print...
Copy
Close
Options...
Recalculate
Output coordinate system:
-- default --
Selected items:
driver arm.SLDASM
Include hidden bodies/components
Show output coordinate system in corner of window
Assigned mass properties
Mass properties of driver arm (Assembly Configuration - Default)
Output coordinate System: -- default --
Mass = 1.350 pounds
Volume = 14.937 cubic inches
Surface area = 184.091 square inches
Center of mass: (inches)
X = -5.445
Y = 0.620
Z = 0.965
Principal axes of inertia and principal moments of inertia: (pounds * square inches)
Taken at the center of mass.
Ix = (1.000, -0.000, -0.027) Px = 0.893
Iy = (0.000, 1.000, 0.001) Py = 38.963
Iz = (0.027, -0.001, 1.000) Pz = 39.356
Moments of inertia: (pounds * square inches)
Taken at the center of mass and aligned with the output coordinate system.
Lxx = 0.920 Lxy = -0.002 Lxz = -1.026
Lyx = -0.002 Lyy = 38.963 Lyz = 0.000
Lzx = -1.026 Lzy = 0.000 Lzz = 39.328
Moments of inertia: (pounds * square inches)
Taken at the output coordinate system.
Ixx = 2.696 Ixy = -4.561 Ixz = -8.117
Iyx = -4.561 Iyy = 80.240 Iyz = 0.808
Izx = -8.117 Izy = 0.808 Izz = 79.868


Mass Properties
Print...
Copy
Close
Options...
Recalculate
Output coordinate system:
-- default --
Selected items:
Motor mount base.SLDASM
Include hidden bodies/components
Show output coordinate system in corner of window
Assigned mass properties
Mass properties of Motor mount base (Assembly Configuration - Default)
Output coordinate System: -- default --
Mass = 14.25 pounds
Volume = 146.13 cubic inches
Surface area = 472.34 square inches
Center of mass: (inches)
X = 0.41
Y = 0.47
Z = -0.85
Principal axes of inertia and principal moments of inertia: (pounds * square inches)
Taken at the center of mass.
Ix = (0.00, 0.00, 1.00) Px = 145.02
Iy = (1.00, 0.00, 0.00) Py = 145.02
Iz = (0.00, 1.00, 0.00) Pz = 226.64
Moments of inertia: (pounds * square inches)
Taken at the center of mass and aligned with the output coordinate system.
Lxx = 145.02 Lxy = 0.00 Lxz = 0.00
Lyx = 0.00 Lyy = 226.64 Lyz = 0.00
Lzx = 0.00 Lzy = 0.00 Lzz = 145.02
Moments of inertia: (pounds * square inches)
Taken at the output coordinate system.
Ixx = 158.41 Ixy = 2.77 Ixz = -4.95
Iyx = 2.77 Iyy = 239.23 Iyz = -5.70
Izx = -4.95 Izy = -5.70 Izz = 150.61


Mass Properties
Print...
Copy
Close
Options...
Recalculate
Output coordinate system:
-- default --
Selected items:
PlatformAssembly.SLDASM
Include hidden bodies/components
Show output coordinate system in corner of window
Assigned mass properties
Mass properties of PlatformAssembly (Assembly Configuration - Default)
Output coordinate System: -- default --
Mass = 0.55730 pounds
Volume = 5.71330 cubic inches
Surface area = 71.93351 square inches
Center of mass: (inches)
X = -0.01298
Y = 0.05508
Z = -0.02927
Principal axes of inertia and principal moments of inertia: (pounds * square inches)
Taken at the center of mass.
Ix = (1.00000, 0.00000, 0.00000) Px = 2.10797
Iy = (0.00000, 0.00000, -1.00000) Py = 2.10798
Iz = (0.00000, 1.00000, 0.00000) Pz = 4.15171
Moments of inertia: (pounds * square inches)
Taken at the center of mass and aligned with the output coordinate system.
Lxx = 2.10797 Lxy = 0.00000 Lxz = 0.00000
Lyx = 0.00000 Lyy = 4.15171 Lyz = 0.00000
Lzx = 0.00000 Lzy = 0.00000 Lzz = 2.10798
Moments of inertia: (pounds * square inches)
Taken at the output coordinate system.
Ixx = 2.11014 Ixy = -0.00040 Ixz = 0.00021
Iyx = -0.00040 Iyy = 4.15228 Iyz = -0.00090
Izx = 0.00021 Izy = -0.00090 Izz = 2.10976

Appendix B – Four-arm Inverse Kinematics

Vector-Based inverse kinematic analysis of the 4 arm Parallel Kinematic Manipulator

Given Constants

desired position of the End Effector

$x := 0$

$y := 0$

$z := -5$

Distance from center of End effector to the connection A_i

$A := 1$

Distance of Motor from origin

$P := 2$

Length of upper arm

$l := 2$

Length of lower arm

$L := 4$

angular position of motor 1-4

$\theta_1 := 0$

$\theta_2 := \frac{\pi}{2}$

$\theta_3 := \pi$

$\theta_4 := \frac{3\pi}{2}$

position of A_1

$A_{x1} := x + A$

$A_{y1} := y$

$A_{z1} := z$

Position of A_2

$Ax_2 := x$

$Ay_2 := y + A$

$Az_2 := z$

Position of A_3

$A_{x3} := x - A$

$A_{y3} := y$

$A_{z3} := z$

Position of A_4

$A_{x4} := x$

$A_{y4} := y - A$

$A_{z4} := z$

Angular position of motor 1

Position of B_1

$B_{x1}(q_1) := (P \cdot \cos(\theta_1) + l \cdot \cos(q_1) \cdot \cos(\theta_1))$

$B_{y1}(q_1) := P \cdot \sin(\theta_1) + l \cdot \cos(q_1) \cdot \sin(\theta_1)$

$B_{z1}(q_1) := l \cdot \sin(q_1)$

Given

$$\left(B_{x1}(q_1)^2 - A_{x1}\right)^2 + \left(B_{y1}(q_1)^2 - A_{y1}\right)^2 + \left(B_{z1}(q_1)^2 - A_{z1}\right)^2 = L^2$$

$\text{Find}(q_1) \rightarrow$

Angular Position of motor 2

Position of B_2

$$B_{x2}(q_2) := \left(P \cdot \cos(\ 2) + l \cdot \cos(q_2) \cdot \cos(\ 2)\right)$$

$$B_{y2}(q_2) := P \cdot \sin(\ 2) + l \cdot \cos(q_2) \cdot \sin(\ 2)$$

$$B_{z2}(q_2) := l \cdot \sin(q_2)$$

Given

$$\left(B_{x2}(q_2)^2 - A_{x2}\right)^2 + \left(B_{y2}(q_2)^2 - A_{y2}\right)^2 + \left(B_{z2}(q_2)^2 - A_{z2}\right)^2 = L^2$$

$\text{Find}(q_2) \rightarrow$

Angular Position of motor 3

Position of B_3

$$B_{x3}(q_3) := \left(P \cdot \cos(\ 3) + l \cdot \cos(q_3) \cdot \cos(\ 3)\right)$$

$$B_{y3}(q_3) := P \cdot \sin(\ 3) + l \cdot \cos(q_3) \cdot \sin(\ 3)$$

$$B_{z3}(q_3) := l \cdot \sin(q_3)$$

Given

$$\left(B_{x3}(q_3)^2 - A_{x3}\right)^2 + \left(B_{y3}(q_3)^2 - A_{y3}\right)^2 + \left(B_{z3}(q_3)^2 - A_{z3}\right)^2 = L^2$$

$\text{Find}(q_3) \rightarrow$

Angular Position of motor 4

Position of B_4

$$B_{x4}(q_4) := \left(P \cdot \cos(\ _4) + l \cdot \cos(q_4) \cdot \cos(\ _4)\right)$$

$$B_{y4}(q_4) := P \cdot \sin(\ _4) + l \cdot \cos(q_4) \cdot \sin(\ _4)$$

$$B_{z4}(q_4) := l \cdot \sin(q_4)$$

Given

$$\left(B_{x4}(q_4)^2 - A_{x4}\right)^2 + \left(B_{y4}(q_4)^2 - A_{y4}\right)^2 + \left(B_{z4}(q_4)^2 - A_{z4}\right)^2 = L^2$$

$\text{Find}(q_4) =$

Appendix C – Embedded Application Code

```
/*
* *
* This code is based on publicly available code available at the following sources:
* http://pastebin.com/NQtbVCFh
* http://forums.leaflabs.com/topic.php?id=1038
*
* The above sources provided period measurement guidance for generic timer
* The following uses advanced timer capabilities such as timer with slave and PWM input mode to
measure period and duty cycle
*/

#include <timer.h>

// initialize global variables
int periodvalue = 0;
int dutycyclevalue = 0;
int count = 0;
int overflow_flag = 0;

HardwareTimer timer1 = HardwareTimer(1);

//quick function to return if there's anyone on the other end of the serialusb link yet.
boolean isConnected(){
  return (SerialUSB.isConnected() && (SerialUSB.getDTR() || SerialUSB.getRTS()));
}

void setup()
{

  SerialUSB.begin();
  while(!isConnected()); //wait till console attaches.
  SerialUSB.println("Hello!");

  //Setup Timer1 Channel 1 (pin 6) as an input
  pinMode(6, INPUT);

 //Configure pin 7 as an output (100Hz 50% duty cycle will be generated and outputted pin 7.  //A jumper
from pin 7 to pin 6 will be applied.
  pinMode(7, OUTPUT);

  // set up advanced timer capabilities
  setup_timer();

}

void __measurePulse_irq()
{

  //For some reason, the interrupt is triggered as soon as
  //the device is turned on. This gets around that bug.
  // came with sample code - probably used because pulse may be in unknown state
  if( count == 0 )
```

```
    return;
  else {

    if(~overflow_flag) {

      periodvalue = TIMER1_BASE->CCR1;
//read the captured value - period defined from pulse going up to next cycle where pulse goes //high.
CCR1 contains period count

  }

    else {
      periodvalue = 0;
      overflow_flag = 0;
    }


    timer1.setCount(0);  //Zero timer
  }
}

void __measureDutyCycle_irq()
{

  //For some reason, the interrupt is triggered as soon as
  //the device is turned on. This gets around that bug.
  if( count == 0 )
    return;
  else {

    if(~overflow_flag) {
      dutycyclevalue = TIMER1_BASE->CCR2;    //read the captured value – dutycycle count defined from
pulse going up to pulse going down.  CCR2 contains duty cycle count

  }

    else {
      dutycyclevalue = 0;
      overflow_flag = 0;
    }

  }
}

//overflow irq accounts for when the counter overflows.  Overflow is  defined in this case as //65535 (2^16-
1). Should not be getting overflow.
void handle_overflow() {

  overflow_flag = 1;
}

void loop() {

  //Display the counter values (period and duty cycle) very 500 mS
  if( count == 100 ) {
    //delay(500);
```

```
    SerialUSB.print("Period: ");
    SerialUSB.println(periodvalue);
    SerialUSB.print("Duty Cycle: ");
    SerialUSB.println(dutycyclevalue); //value = 0;
    count = 0;
  }


 //Pulse pin 7 at a rate of 100Hz @ 50% DC - this is the PWM signal with 100Hz and 50% duty //cycle.
  digitalWrite(7, HIGH);
  delay(5);
  digitalWrite(7, LOW);
  delay(5);

  count++;

}


//Halt processor and flash LED - used for debugging - code that calls this function is currently //not used -
can be removed.
void HaltAll(void) {

  //Timer1.pause();
  timer1.pause();
  pinMode(BOARD_LED_PIN, OUTPUT);

  while(1) {

     toggleLED();
     delay(1000);

  }

}


// configure timer1 which is an advanced timer with a slave to measure period and duty cycle.
void setup_timer(void) {

  //Create a pointer to the timer (why?)
  timer_dev *t = TIMER1;

  //Initialize timer
  timer1.pause();

  /*
   * A prescaler of 288 means that at 72MHz, I get the following:
   * 100 Hz = 2510
   * 10 Hz = 25084
  */
  timer1.setPrescaleFactor(288);

  timer1.setOverflow(65535);
  timer1.setCount(0);
```

```
//New code to set overflow timer
timer1.setMode(3, TIMER_OUTPUT_COMPARE); //Set this channel to compare
timer1.setCompare(3, 65534); //Set the compare for the channel to the overflow value
timer1.attachInterrupt(3, handle_overflow); //Assign the interrupt handler

timer1.refresh();

//Create a pointer to the timer registers (why?)
timer_reg_map r = t->regs;


//capture compare regs TIMx_CCRx used to hold val after a transition on corresponding ICx

//when cap occurs, flag CCXIF (TIMx_SR register) is set,
//and interrupt, or dma req can be sent if they are enabled.

//if cap occurs while flag is already high CCxOF (overcapture) flag is set..

//CCIX can be cleared by writing 0, or by reading the capped data from TIMx_CCRx
//CCxOF is cleared by writing 0 to it.

//Capture/Compare 1 Selection
// set CC1S bits to 01 in the capture compare mode register.
// 01 selects TI1 as the input to use. (page 336 stm32 reference)
// (assuming here that TI1 is D6, according to maple master pin map)
// CC1S bits are bits 0,1
bitSet(r.adv->CCMR1, 0);
bitClear(r.adv->CCMR1, 1);


//Input Capture 1 Filter.
// need to set IC1F bits according to a table saying how long
// we should wait for a signal to be 'stable' to validate a transition
// on the input.
// (page 336 stm32 reference)
// IC1F bits are bits 7,6,5,4
bitClear(r.adv->CCMR1, 7);
bitClear(r.adv->CCMR1, 6);
bitSet(r.adv->CCMR1, 5);
bitSet(r.adv->CCMR1, 4);

//sort out the input capture prescaler..
//00 no prescaler… capture is done as soon as edge is detected
bitClear(r.adv->CCMR1, 3);
bitClear(r.adv->CCMR1, 2);


//select the edge for the transition on TI1 channel using CC1P in CCER
//CC1P is bit 1 of CCER (page 339)
// 0 = rising
// 1 = falling
bitClear(r.adv->CCER,1);

//Capture/Compare 1 Selection of down part of the pulse on TI1
// set CC2S bits to 10 in the capture compare mode register.
// 01 selects TI1 as the input to use. (page 336 stm32 reference)
```

```
  // (assuming here that TI1 is D6, according to maple master pin map)
  //CC2S bits are bits 8,9
  bitSet(r.adv->CCMR1, 9);
  bitClear(r.adv->CCMR1, 8);

  //select the edge for the transition on TI1 channel using CC2P in CCER
  //CC2P is bit 5 of CCER (page 339)
  // 0 = active high
  // 1 = active low
  bitSet(r.adv->CCER,5);


  //select filtered trigger option:  TS bits are set 101 in SMCR
  //selects the trigger input to be used to synchronize counter
  //in this case a filtered TI1 (page 328)
  bitSet(r.adv->SMCR,6);
  bitClear(r.adv->SMCR,5);
  bitSet(r.adv->SMCR,4);

  //select slave mode selection:  SMS bits are set 100 in SMCR
  //configure slave mode in reset mode (page 329)
  bitSet(r.adv->SMCR,2);
  bitClear(r.adv->SMCR,1);
  bitClear(r.adv->SMCR,0);


  //set the CC1E bit to enable capture from the counter.
  //CCE1 is bit 0 of CCER (page 339)
  //Enable capture
  bitSet(r.adv->CCER,0);


  //set the CC2E bit to enable capture from the counter.
  //CCE2 is bit 4 of CCER (page 339)
  bitSet(r.adv->CCER,4);


 //attach interrupts for when pulse goes high for period measurement and pulse goes low for //duty cycle
measurement
  timer1.attachInterrupt(1, __measurePulse_irq);
  timer1.attachInterrupt(2, __measureDutyCycle_irq);

  //timer_resume(TIMER4);
  timer1.resume();

}
```

Appendix D – Instruction on How to Install Maple IDE

The following instructions were used to install the Maple IDE. First, extract all the files from zip file that was downloaded. Regardless of the OS and its edition, drivers needed to be downloaded and installed prior to using the IDE.

Here is a list of actions that were performed to ensure the proper installation of the IDE:

- Install Serial Drivers
- Install (Device Firmware Upgrade) DFU Drivers
- Compile the sample blink application
- Place in the board in boot load operations mode
- Upload the blink application
- Verify the board LED is blinking to validate blink application
- Reset board
- Compile the HelloWorld application
- Select Serial Port from Tools menu
- Place in the board in boot load operations mode
- Upload the HelloWorld application
- Open SerialMonitor application from Tools menu
- Verify that “Hello World” phrase is displayed on the Serial Monitor application

Although the instructions on the Maple site list the installation of the DFU drivers before the serial drivers, it was noticed that doing so impacted the serial driver functionalities and installation, especially when installing the IDE on a 64-bit Windows OS. The site also mentions that the OS will prompt for the drivers. That was not the case, especially with Windows 7. To solve this problem, the drivers were manually installed and this issue was reported to the

community forum. Although the community simply lists installing the DFU drivers and the serial drivers and trying the blink applications, it was decided to do a more thorough test of the IDE installation.

To install the serial drivers, plug the Maple board to the host PC, and go to the "Device Manager" that is located in the "Control Panel". The board may appear under "Other Devices" or "Ports (COM & LPT)". Highlight the board name and select "Update Driver Software" from the Action menu. Browse for the serial drivers that were part of the unzipped bundle. The serial drivers should be located in /drivers/MapleDrv/serial. There may be a prompt indicating the driver is not signed. The installation of the unsigned driver should be done regardless of the prompt.

There are DFU drivers in the /drivers/MapleDrv/dfu directory, but they are suitable for Windows XP 32-bit and not for Windows 7 64-bit. The proper DFU drivers can be obtained from http://sourceforge.net/projects/libusb-win32/files/. The libUSB bundle needs to be extracted. The Maple board needs to be connected to the host PC, and the "Device Manager" in the "Control Panel" needs to be opened. The Maple board should be put into the continuous Boot loader mode by pressing the "Reset" button until the LED blinks fast and then the blinking slows down. The "Reset" button needs to be pressed again and the button "BUT" should be hit and held until the blinking slows again. The "Device Manager" will display a USB device with a reference to Maple as broken. This should not be confused with the COM device. There is a wizard in the /bin/x86 bundle libUSB labeled in-wizard.exe. This wizard application should be executed and the dialogs should be reviewed. The team discovered that the IDE looks for DFU at 0x0003 but the wizard tends to use 0x0004 for an ID. This ID for the driver needs to be changed to 0x0003 so that it will work with the IDE. A file with the file type of .inf is generated

and the driver can now be installed by going to the computer's "Device Manager" and selecting "Update Driver Software" menu option. If the device does not appear, a device can be added and the newly created .inf file is supplied. There may be a prompt validating if the user wishes to install an unsigned driver; regardless of the prompt the driver should be installed.

Once the drivers have been installed, the test applications such as blink and HelloWorld can be compiled and uploaded. There are times that the uploaded application was so time-consuming that the "Reset" button needs to be pressed more than once or the board needs to be put in boot load mode.