Robot Design Project

Engineering 1282.01H

Spring, 2016

Team B5: Leeroy Jenkins

Paul Harshbarger

Benjamin Higgins

Jonathan Liew

Logan Meyer

E. Helber/R. Freuler MWF 8:00 AM

Start Date: 01/29/16

Date of Submission: 04/25/16

Team B5 – Executive Summary

The Fundamentals of Engineering for Honors Space Administration required a fully autonomous robot to complete missile launch preparations at the spaceport. It was important that this robot was created as the safety zone around the launch site contained materials hazardous to human personnel. FEHSA commissioned the Ohio State University Research and Engineering Development team to select a prototype presented by our company and competitors.

A team of four engineers was brought together to design several potential prototypes that could successfully navigate OSURED's scale model spaceport. After evaluating three potential designs, the team settled upon a robot that would utilize omnidirectional wheels for holonomic drive. On top of the chassis were two mechanical arms for toggling the communications, a servo motor arm for manipulating supplies, and two offset trunnions for pressing the fuel buttons. Utilizing the omnidirectional drive, the robot would be able to more efficiently move around the course, and would reliably align with its tasks. With almost entirely mechanical means of accomplishing objectives, the robot's design was kept simple and low-cost.

Exhaustive testing has proven the design to be reliable and efficient. The robot is capable to consistently accomplishing all objectives set by FEHSA and within the constraints set by the agency. In official FEHSA testing, the robot scored one prefect run out of three in individual competitions, and seven perfect runs out of seven in head to head competitions.

Future changes could be made to the robot to increase efficiency for the full scale robot. Larger diameter wheels could be pursued to allow the robot to go up the main ramp. This would cut a significant amount of time from preparations. Secondly, more durable materials could be pursued. Prototype materials such as paper and rubber bands were used to great effect in the scale model; however, these materials would not do for full scale production.

Contents

List of Tables

List of Figures

1. Introduction

Autonomous machines allow for safer, more precise, and faster task completion than human systems. Their motors, sensors, and programs play a major role in attaining these results consistently. By integrating these systems into an adaptable code that focuses on minimizing error, maximum efficiency and success can be achieved. Currently, these systems need to be applied to create an autonomous robot to complete the tasks for the Fundamentals of Engineering for Honors Space Administration's, or FEHSA's, rocket launch site [1]. The robot needs to complete preparation tasks for rocket launch to ensure the safety of human personnel near the launch zone. The purpose of this project was to create a scaled prototype of the full robot to secure a working contract with FEHSA.

1.1 Problem Statement – General Overview

The section serves to more clearly define the specific tasks that FEHSA required for their rocket launch preparation. The robot needed to transfer air purification system supplies to the space shuttle from the lower level. Also, it needed to supply liquid hydrogen and oxygen to the rocket and initialize the communications systems for the launch sequence, weather, and shuttle electronics stations. Lastly, the robot had to signal the crew once all of the necessary launch preparation tasks were completed [1].

1.2 The Team and Project Dates

Team B5: Leeroy Jenkins consisted of four members. The members were Paul Harshbarger, an electrical engineering pre-major; Benjamin Higgins, a materials science pre-major; Jonathan Liew, a biomedical engineering pre-major; and Logan Meyer, a mechanical engineering pre-major.

The project was time sensitive and needed to meet certain timelines set out by FEHSA. Before February 26, 2016, the robot needed be able to react to a start light and make it to the upper surface of the launch preparation site. By March 4, 2016, the robot was to toggle the communications switches forward or backward based on crew input. Prior to March 11, 2016, the robot should have been able to transport the air purification supplies from the bottom bin to the top receptacle. By March 25, 2016, the robot needed to press and hold the correct fuel button to transfer either liquid hydrogen or oxygen to the rocket [2]. On April 1st, the robot competed in an individual competition viewed by FEHSA for initial design impressions. On April 9, 2016, the robot competed against other companies to determine who would be awarded the contract with FEHSA [1].

1.3 Roadmap for Remainder of Report

The following section, Preliminary Concepts, highlights the methodology and results of the team's initial robot design brainstorming process. The Analysis, Testing, and Refinement section discusses all of the calculations, tests, and analyses that led to alterations in the physical design, strategy, and code. The next section, entitled Individual Competition, discusses the strategy going into the individual competition, the outcomes of the competition, and the changes made following the competition. The Final Design section explains the full details of the final design including the final budget, SolidWorks model of the design, code, and electrical systems documentation. The following section, named Final Competition, describes the strategy for the competition and the outcomes of the competition. The final section, Summary and Conclusions, summarizes all of the previous information and analyzes all of the information to make recommendations for what changes would be made in future, similar projects. Lastly, the Appendices contain all the other information needed to fully understand the project including figures, tables, budgets, and more.

2. Preliminary Concepts

The purpose of this section is to provide detail on the specific requirements and constraints for the project. Additionally, it will outline the brainstorming process used to translate individual work into an agreed upon preliminary concept. It will also give detailed sketches, images, flowcharts, and pseudocodes needed to fully understand the initial design and programming plan.

2.1 Project Requirements and Constraints

In order to best complete the tasks required by the Fundamentals of Engineering Honors Space Administration, FEHSA placed specific constraints on the robot to ensure that the robot was economically and physically pursuable. A full list of these constraints are given in Table 1 below.

Table 1: Constraints for Robot Design Project [1].

Constraint	Purpose
$160 Budget	Ensure that the robot is economically reasonable for FEHSA
Only FEH Store Approved Parts	Ensure quality control and maintenance assistance with parts
12” Max Height, 9” Max Length, 9” Max Width	Ensure that robot will fit in the rocket launch preparation site
Self-Propelled Robot	Ensure safety for humans by removing them from launch site
Place QR Code 9” Above Ground	Ensure that robot can receive Robot Positioning System (RPS) data
Performance Tests	Deadlines to ensure that robot will be able to complete all the tasks
No Adhesives on Proteus	FEH property must be protected
2 Minute Course Time Limit	Rocket preparation is time sensitive

In addition to these constraints, the robot needed to complete a specific set of tasks to earn points for FEHSA’s prototype testing scoring system. Also, penalties existed to reduce scores for

performing tasks that would negatively affect rocket launch preparation. A brief summary of the points scoring system used is given in Table A1 in Appendix A. The following subsections will contain a more detailed explanation of all of the tasks in the table. A top view of the course can be found in Figure A1 in Appendix A for reference in understanding the course and tasks.

2.1.1 Start Light

Before the robot attempted to complete any of the tasks, it first needed to be given a red light start signal on a 12” by 12” platform [1]. An image of the starting area is given below in Figure 1. The start light is the white rectangle at the center of the yellow area, while the start area is the entire area encompassed by the blue outline. In order to achieve any points, the robot needed to start on the start light.

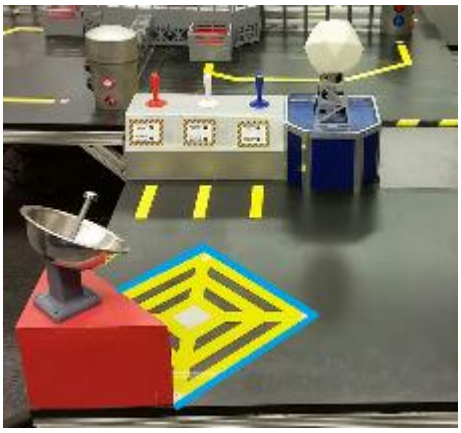


Figure 1: Starting Area for Robot Course [3].

2.1.2 Air Purification System Supplies

One of the tasks that the robot needed to perform was to transport the air purification system supplies from cargo storage bin on the bottom platform to the receptacle on the top platform. The supplies weighed 77 ± 1 grams, and the storage bin was approximately two inches lower than the receptacle bin on the upper surface [1]. As shown in Table A1, the supplies needed to be removed

from the bin to receive the primary points, and correctly placing the supplies in the receptacle bin allowed for the receipt of additional secondary points [1]. Images of the storage and receptacle bins are given in Figure 2 and Figure 3 below. See Figure A1 to see the bins in relation to each other on the top course view. The storage bin appears in the bottom right portion of the course, while the receptacle is in the top left portion of the image.

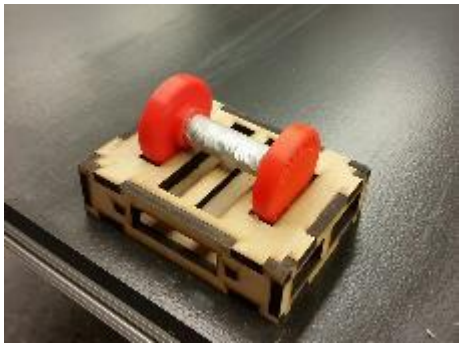


Figure 2: Storage Bin for Air Purification System [3].

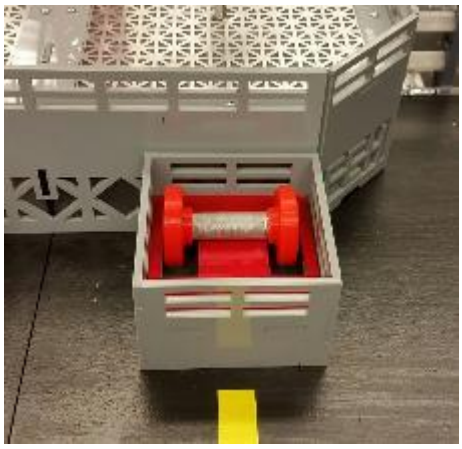


Figure 3: Receptacle Bin for Air Purification System [3].

2.1.3 Launch Switch Sequence

The robot needed to flip the switches for the communications systems in the correct direction to establish communication systems for the launch sequence, weather, and shuttle electronics

stations. To accomplish this task, the robot needed to receive information from the Robot Positioning System, or RPS, on whether to push the switches forward or pull them backwards [1]. The switches were accessible from both the top and bottom platform of the preparation site. However, there was a height difference between the ground and switches depending on which side the switches were approached from, measured to be 3.0 inches. As shown in Table A1, primary points were awarded for toggling one switch in the correct direction, while full secondary points were awarded for moving all three switches into the correct positions [1]. An image of these switches on the course appears below in Figure 4.

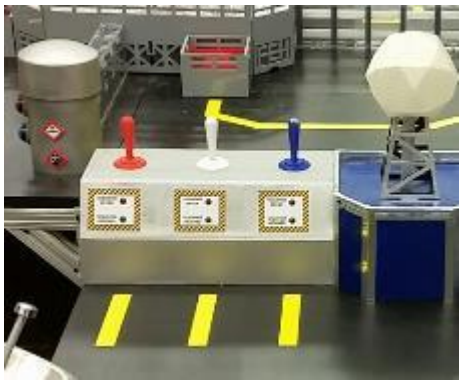


Figure 4: Launch Sequence Activation Switches [3].

2.1.4 Fuel Delivery

A major task was to deliver either liquid hydrogen or liquid oxygen fuel to the rocket for five seconds depending on what the FEHSA crew needed. To signal for liquid hydrogen, a red light would shine in front of the buttons used to deliver fuel. To signal for liquid oxygen, a blue light would shine in front of the buttons used to deliver fuel [3]. As shown in Table A1, primary points were awarded for pressing the correct fuel button, while additional secondary points were awarded for holding the button for five seconds [1]. An image of the fuel delivery system is given on the following page in Figure 5.

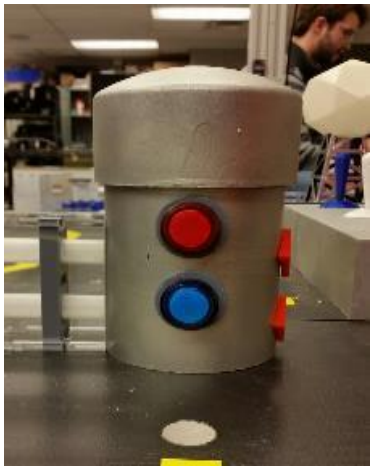


Figure 5: Fuel Delivery [3].

2.1.5 Final Launch Button

Lastly, the robot needed to be able to press the final launch button, located on the start platform, to inform the crew that the rocket was ready for launch. In order for the rocket to actually launch, all of the primary and secondary tasks needed to be completed. As shown in Table A1 there were only primary points associated with pressing the final button [1]. An image of the final button is given below in Figure 6.

Figure 6: Final Launch Button [3].

2.1.6 Ramp Navigation

In order to navigate between the upper and lower portions of the course, the robot could use one of two available ramps. The first possible ramp option was a short, steep ramp with a construction bump on the top of it. The bump's diameter was measured to be one inch. The other possible ramp option was a temporary access ramp that was longer, less steep, and contained two right angle turns. However, the ramp did not have RPS established because it was only temporary [1]. Images of the main ramp and the temporary access ramp are given in Figure A2 in Appendix A and Figure A3 in Appendix A.

2.1.7 Penalties

Lastly, penalties had the ability to impact a robot's score for the competitions. The punishable offenses included knocking the weather ball off of the tower, interfering with a competitor's robot or objects, and failing to control own robot [1]. Knocking the weather ball off of its location or interfering with a competitor's objects resulted in a ten point deduction. An image of the weather ball is given in Figure A4 in Appendix A. Failing to keep the robot within its current course's bounds and/or impeding another competitor robot's task execution resulted in automatic disqualification.

2.2 Brainstorming Process

This section contains all of the information needed to understand the individual and team portions of the brainstorming process that were used to generate a preliminary concept. First, members individually viewed the course to generate ideas for the completion of tasks. Next, everyone derived and created sketches for their own ideas for the robot chassis, drivetrain,

mechanisms for completing all of the tasks, and course path strategy. Then, the team worked as a group to eliminate poor ideas and generate new ones to arrive at a preliminary concept using scoring systems. After, physical and SolidWorks mockups of the robot were made. Finally, pseudocodes and flowcharts were written to plan out the task execution in terms of programming.

2.2.1 Individual Ideas

This section contains all of the individual ideas generated from the brainstorming process. This information is presented in text with supporting SolidWorks models and hand-drawn sketches.

2.2.1.1 Chassis and Drivetrain Ideas

One of the brainstorming ideas for the chassis involved a square base with rectangular cuts on the corners for four wheels. The robot was to be rear driven by two motors, with the wheels on each side connected by a chain and sprockets system. Additionally, a wireframe was to extend from the chassis base to allow for the mounting of arms and sensors higher up on the robot. An image of this design is given in Figure B1 in Appendix B.

Another brainstorming idea involved a laser cut chassis design with four Mecanum wheels driven by four individual motors. This chassis and drivetrain system would allow for the robot to move in any direction based on motor powers. An image of this design in given in Figure B2 in Appendix B.

A third idea utilized a rectangular chassis being driven by tread system. There would be three wheels on each side used to maintain tension in the treads. The top wheels on either side of the robot would be used as the drive wheels for the entire robot. This idea was proposed by multiple group members, and an image of this design is given in Figure B3 in Appendix B.

A final brainstorming idea for the chassis and drivetrain revolved around a rear-driven two wheel drivetrain with two front skids. Another idea utilized four omnidirectional wheels driven by four separate motors. This drivetrain was to be implemented onto a stationary rectangular chassis with the wheels and motors mounted below the chassis. Other ideas involved similar concepts to these, with rear wheel drive systems and skids or a different geometry of omnidirectional wheels, all on a rectangular chassis.

2.2.1.2 Air Purification System Ideas

One idea developed in the brainstorming process for the handling of the supplies involved a forklift-like design. The forklift would be mounted at a height equal to that of the supplies storage area. Once the forklift was beneath the supplies, a servo motor would be used to tilt the supplies in towards the body and secure them. It would then uncurl the forklift to deposit them. An image of this design is given in Figure B4 in Appendix B.

A different design featured a fixed arm would be extended out and rotate upward once the Lexan wall had been hit. Then, the arm would rotate downward to release the supplies once the robot ran into the storage area. A sketch of this design is given in Figure B5 in Appendix B.

A vice grip option was proposed where the two sides of the dumbbell would be grabbed and carried to the supplies receptacle. A servo motor would be used to clamp onto the sides of the dumbbell and release when necessary. A sketch of this design is given in Figure B6 in Appendix B.

Other ideas generated included a retractable pole with mounted magnets inside of a vertical PVC pipe. To lift up the supplies, a servo motor would be used to lower the apparatus into the storage bin, while the retracting the pole fully would cause the supplies to fall off into the

receptacle bin. A clamp design was discussed where the robot would grab the supplies from the top and bottom of the center beam to carry them the receptacle. Additionally, a horizontal PVC design with magnets was proposed. Magnets would line the inside of the PVC pipe, and the pipe would be raised or lowered to drop off or pick up the supplies.

2.2.1.3 Launch Switch Mechanisms

The first brainstorming idea for toggling the switches involved a T-shaped hook that was placed at a fixed height. This design would allow the switches to be pushed or pulled from either the upper or lower platform, preventing the need to change platforms. A sketch of this design is given in Figure B7 in Appendix B.

Instead of a T-shaped hook at the end of the stationary arm, a slightly different design was proposed where there would be a double-sided hook that would allow for pushing or pulling. A servo motor would be used to rotate the head of the arm in the necessary direction to push or pull the switch. A sketch of this design in Figure B8 in Appendix B.

Another idea that was considered was an arm that moved up and down on a motor via gear and pinion. The robot would then drive forward as needed to push the launch switches forward from either the top or bottom portion of the course. An image of this design appears below in Figure B9 in Appendix B.

Additionally, a boxing glove design with two motors was considered. The first motor would raise and lower the arm, while the second motor would extend and retract the arm. The switches would only be pushed, not pulled from the top and the bottom. Another idea utilized a clamp mounted at a height equal to that of the switches. Once the switches are clamped, the robot would either drive forward or backward to push or pull the switch.

2.2.1.4 Fuel Button Ideas

One of the ideas proposed for fuel button management was a square that matched the geometry of the fuel buttons. Depending on the color of the light, a slot the size of the buttons would be retracted into the square by a motor. Then, the robot would just drive forward and the correct button would be hit. An image of this apparatus is given in Figure B10 in Appendix B.

A seesaw lever controlled by a servo motor was proposed. Depending on the color of the fuel light, the lever would rotate about its center to angle in towards the desired button. Then, the robot would just drive forward and press the desired button. An image of this design is given in Figure B11 in Appendix B.

The next idea used a flat panel design that would be raised or lowered by a motor. Depending on the color of the light, the panel would be raised or lowered, and the robot would drive forward to press the desired button. An image of this design is given in Figure B12 in Appendix B.

Other ideas considered included a checkerboard design. Based on the light color, the robot would align itself with the fuel buttons such that the extended part of the checker board would hit the desired button. Another idea considered was placing two fixed rods on the front and back of the robot matching the heights of the buttons. Based on the color of the light, the robot would rotate to align the correct side with the fuel buttons and drive into them.

2.2.1.5 Final Button Mechanism

Essentially, everyone on the team came up with the idea of using any previous mechanism to press the final button or having a bumper on one side of the robot to prevent damages when running into the final button. One such example of bumper is given in Figure B13 in Appendix B. All the bumper ideas were roughly the same as this. Additional options that were considered included

launching a catapult projectile at the final button from a distance and using a gear rail to extend out an arm to press the final button.

2.2.2 Group Brainstorming

This section explains the group portion of the brainstorming process. First, criteria for drivetrain, chassis, and mechanism success were established. Then, multiple screening matrices were implemented in order to weed out the weaker concepts and bring the best ideas forward. The screening matrices functioned by first establishing a reference design. The reference design was supposed to be neutral in terms of all of the criteria, making it clear which designs were above average or below average. Positive signs were awarded to designs better than the reference design in a criterion, negative signs were awarded to designs worse than the reference, and zeros were awarded to designs comparable to the reference design. These were tallied up to determine the best few designs that would be further considered.

Next, three full robot layouts were created from these concepts based on what seemed to have a high success probability. This was a judgement call made collectively by the team based on the original scoring. A final scoring matrix was then implemented to choose the best possible full design from these three layouts, using newly established criteria. Additionally, physical and SolidWorks models of the winning robot from the scoring matrix are provided. Finally, flowcharts and pseudocode are provided for the robot's completion of the tasks on the course.

2.2.2.1 Drivetrain Screening Matrix

The screening matrix for the drivetrain focused on a number of key criteria for determining the viability of idea proposals. These criteria were: balance, minimal blockage, center of gravity,

durability, cost, structure, traction, weight, and turning. The balance, tractions, weight, and center of gravity criteria existed to prevent the robot from tipping over on the ramps or drifting to one side or the other due to uneven weight distribution. The minimal blockage criteria meant that the drivetrain system should not interfere with the mounting of other sensors and mechanisms. The durability and structure criteria were to make sure that the design would be able to withstand rigorous testing and produce consistent results. Turning was an important criteria because the robot would be useless if it could not turn. Lastly, cost needed to be included because there was a $160 budget.

All designs were compared to the reference design, which was a two wheel powered drivetrain with four wheels. After tallying the positive and negative signs, the top designs were a two wheel drivetrain with front skids, an omnidirectional wheel system on two wheels, and four omnidirectional wheels on three motors. All of the criteria and the results of this screening process are given in Table C1 in Appendix C. It should be noted that the "M" in parentheses in the matrix indicates the number of motors needed for the design. For example, 3M means three drive motors needed.

2.2.2.2 Chassis Screening Matrix

The screening matrix for the chassis involved some similar and different criteria than that for the drivetrain. The criteria were balance, minimal blockage, center of gravity, durability, cost, structure, and weight. The descriptions of these criteria are the same as that for the drivetrain matrix, so further explanation will not be provided.

The reference design that the ideas were compared to was a rectangular chassis made out of acrylic with the wheels mounted on the exterior of the chassis. The top chassis designs were a

rectangular chassis made out of acrylic with the wheels enclosed and the same design made out of PVC. The full criteria and results of this screening matrix are provided in Table C2 in Appendix C.

2.2.2.3 Switch Toggling Screening Matrix

The screening matrix for the switch toggling included balance, minimal blockage, weight, maintenance, cost, height, extendibility, and motors need as criteria. The new criteria introduced were height, extendibility, and motors needed. Height and extendibility were important considerations because whether or not the apparatus could reach the switches would strongly affect the design's performance. Motors needed was considered because more motors meant more moving parts and more that could go wrong. Reducing the amount of motors meted would help with both consistency and cost. Additionally, maintenance considerations were made for the design because increased time fixing something old meant less time developing something new.

The five designs in question were compared to a reference design of an arm on a motor that goes up and down, using the robot's driving to push the switches from either side. The top designs were a T-shaped hook at a fixed height and a general fixed stationary arm that each pushed and pulled the switches from the same side. The full criteria and results of this screening matrix appear in Table C3 in Appendix C.

2.2.2.4 Fuel Delivery Screening Matrix

The criteria for the fuel delivery screening matrix were minimal blockage, durability, cost, structure, accuracy, weight, maintenance, and motors. The only new criteria here was accuracy,

which is very important in that secondary points were lost for accidentally pressing the wrong button at any point.

The five designs in question were compared to the reference design of flat panel on a motor that moves up and down to press the appropriate button. The top designs were a checkerboard design with a geometry that allows the correct button to be pressed based on robot positioning, pressing the buttons from the front and back, and a seesaw lever on a motor that tilts in the needed direction. The full criteria and results of this screening matrix appear in Table C4 in Appendix C.

2.2.2.5 Final Button Screening Matrix

The screening matrix for the final button press included consistency, minimal blockage, durability, cost, structure, and weight as criteria. The consistency and cost of the design were very important because if the robot cannot press the final button, then the rocket cannot launch. Also, this was a good place to try and reallocate cost to other parts of the robot, so minimizing the design price was a goal.

The reference design was an extending arm to press the final button. The top designs involved a bumper system that just ran into the final button while protecting the rest of the robot. The full criteria and results of this screening matrix appear in Table C5 in Appendix C.

2.2.2.6 Air Filtration Supplies Screening Matrix

The criteria for the air filtration matrix were balance, minimal blockage, drop potential, maintenance, cost, structure, accuracy, and weight. Drop potential was a very important criterion because if the robot dropped the dumbbell at an undesired location on the course, then full points would be lost, and it would obstruct the robot's future motion. Additionally, accuracy was an

important consideration in that the robot must be able to pick up the dumbbell in small space and place it accurately in the final bin.

The reference design that the potential designs were compared to was a forklift running on one motor. The top designs from the matrix were a design that mounted magnets on a vertical rod and a horizontal PVC pipe with magnets on the interior surface. The full criteria and results of this screening matrix appear in Table C6 in Appendix C.

2.3 Generating a Preliminary Design

Using the results from the screening matrices, three potential ideas for a full robot layout were created and modeled. These designs were later compared in a scoring matrix with different criteria than the screening matrices to determine the team's agreed upon preliminary concept.

2.3.1 The Three Ideas

The first full layout design featured four omnidirectional wheels in an X configuration, or a Killough drive. Two opposite wheels were to be connected by an axle and driven by one motor using a beveled gear system. The motors were to be mounted to the bottom of the base of the robot, which was a PVC sheet using laser cut acrylic. The Proteus microcontroller was attached to the PVC sheet on top. A single servo motor was used to drive a mechanism that raised up and down on a gear and pinion. At the back of the robot, a rod with a magnet on the end slid through a PVC pipe to pick up the supplies. When raised to maximum height, the supplies lost contact with the magnet and dropped into the receptacle bin. In the front of the robot, a flat panel was raised by the same gear and pinion mechanism to press the correct fuel button. On the left and right sides, an arm pivoted on the side of the robot when pressed against a wall. This was used to hit the toggles

once the robot runs into the correct wall. A sketch of this design is given in Figure 7 below. The proposed course path for this design is given in Figure A5 in Appendix A.

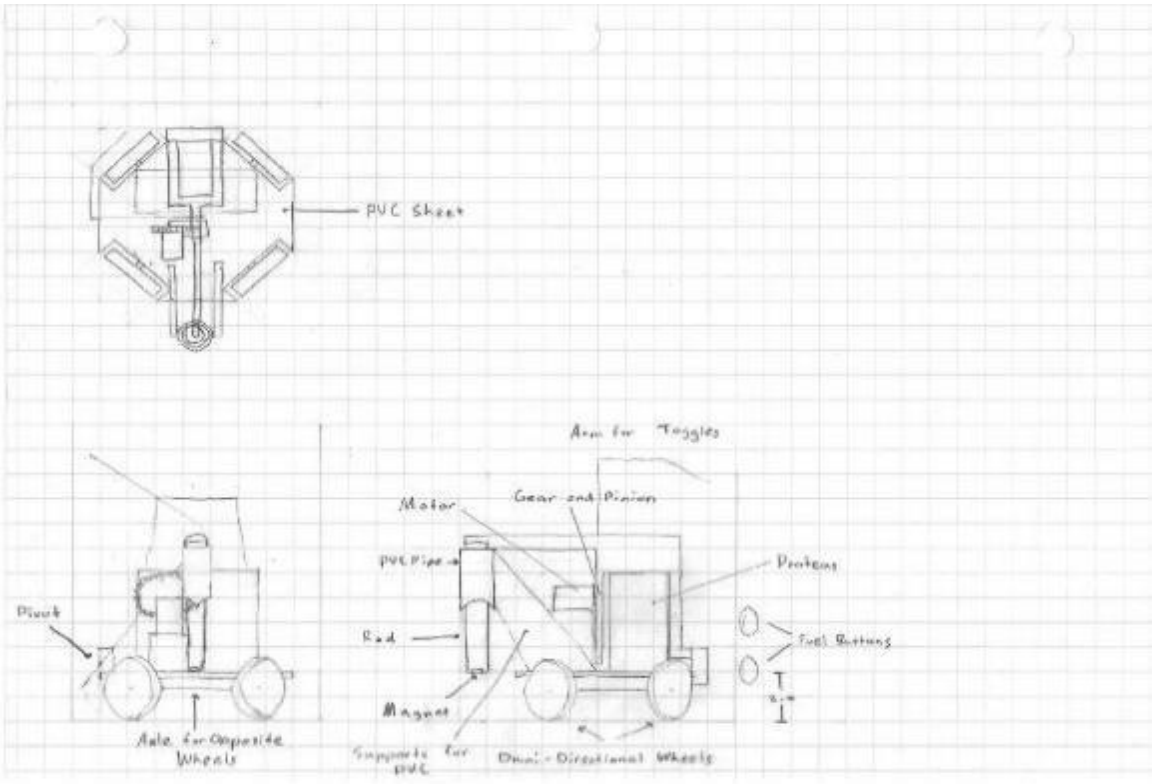


Figure 7: Orthographic Projections of First Design.

The second full layout design featured a two wheel drive system run on two motors with two skids on the front. The chassis was a rectangular sheet that enclosed the rear wheels and extended past the skids. On the front of the robot, a checkerboard design was implemented to handle the fuel delivery buttons. The robot would line up according to the light color and drive forward to press the desired button. A T-shaped hook was placed on the top of the robot to push and pull the communications switches. A horizontal PVC pipe with magnets attached was implemented to pick up and release the dumbbells. A servo motor would be used to lift the apparatus vertically, and a rod through the pipe would be used to release the dumbbell at a desired height into the receptacle.

A SolidWorks model of this design is given below in Figure 8. The course path is given in Figure A6 in Appendix A.

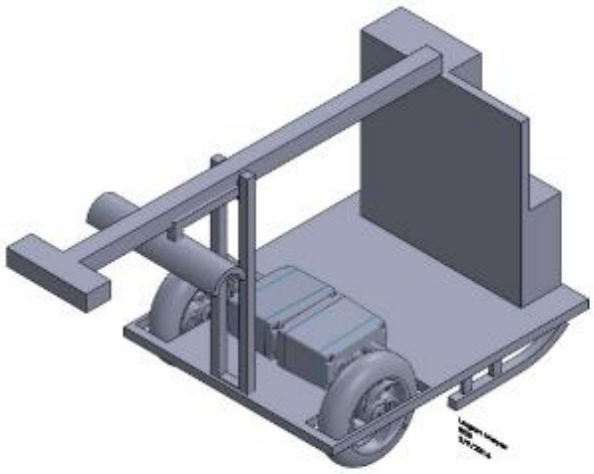


Figure 8: Second Potential Preliminary Design.

The third considered design was a tread-based system mounted on a rectangular PVC base. The treads were powered by a drive wheel placed at the top of each tread, while the bottom wheels would allow for driving and maintaining necessary tread tension. A vertical lever would be used to press the correct fuel button. A servo motor would rotate the lever to face the correct button, and the robot would drive forward to press it. A forklift design would be used to lift the dumbbell and carry it to the receptacle bin. An arm would be raised vertically through a tube and fall onto the switches to toggle them. Once the switches were toggled, the arm would retract back into the tube. A sketch of this design is given on the following page in Figure 9. The proposed course path for this design is given in Figure A7 in Appendix A.

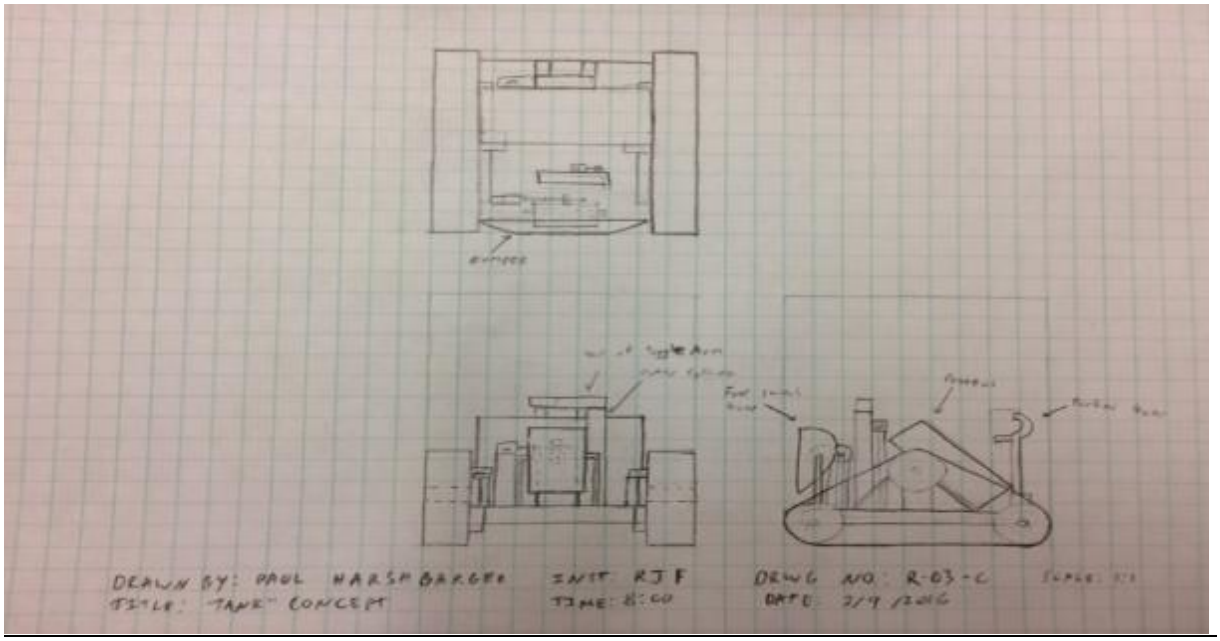


Figure 9: Third Potential Design for Preliminary Concept.

2.3.2 The Scoring Matrix

Once the final preliminary ideas were created, the best choice needed to be selected so that the build process could begin. To do this, a scoring matrix was created to rank the robot layouts. Weight was assigned to each criteria depending on how essential it was viewed to be to the success of the design. Some of the highest weighted criteria included cost, maintenance, mobility, and minimal blockage. Each design was ranked on a scale from one to five, with five being the highest, and multiplied by the weight percentage as a decimal. The tiebreaker, should one be needed, was the total number of points earned, separate from weight. The winning design was the first design, Design A, in the scoring matrix. A summary of the designs and the results of the scoring matrix are given in Table C7 in Appendix C.

2.3.3 Mockup Creation

Once the preliminary concept was chosen, mockups were created for the design, both physically and in SolidWorks. The purpose of these mockups was to ensure that the design was geometrically, logically, and economically reasonable. For example, this would address the question of if the motors could all be mounted in a certain amount of space. An image of the physical mockup is given below in Figure 10. Additional images of the physical and SolidWorks mockups can be found in Figure D1, Figure D2, and Figure D3 in Appendix D.

Figure 10: Completed Physical Mockup.

Within one day of the completion of these mockups, it became apparent that revisions needed to be made. First, discussion amongst the team and with the instructional staff indicated that there were better more reliable options for the fuel button than the front panel on a motor. Additionally, the linear actuation desired from a gear and pinion system for the supplies was shown to be outside of the budget possibilities at an approximate cost of $75.00 [5]. Finally, the staff indicated that implementing a beveled gear system would be incredibly difficult under the budget and time constraints.

For these reasons, a few revisions were made to the mockup to make it more feasible. First, the three motor system was replaced with a four motor, direct-drive system to prevent the need for beveled gears. In addition, a checkerboard design replaced the front panel on a motor idea to reduce cost. Lastly, a forklift design using dual-locking positions with a rotating wheel replaced the gear and pinion system in order to fit into budget constraints. An image of the new chassis design is given in Figure 11 below.

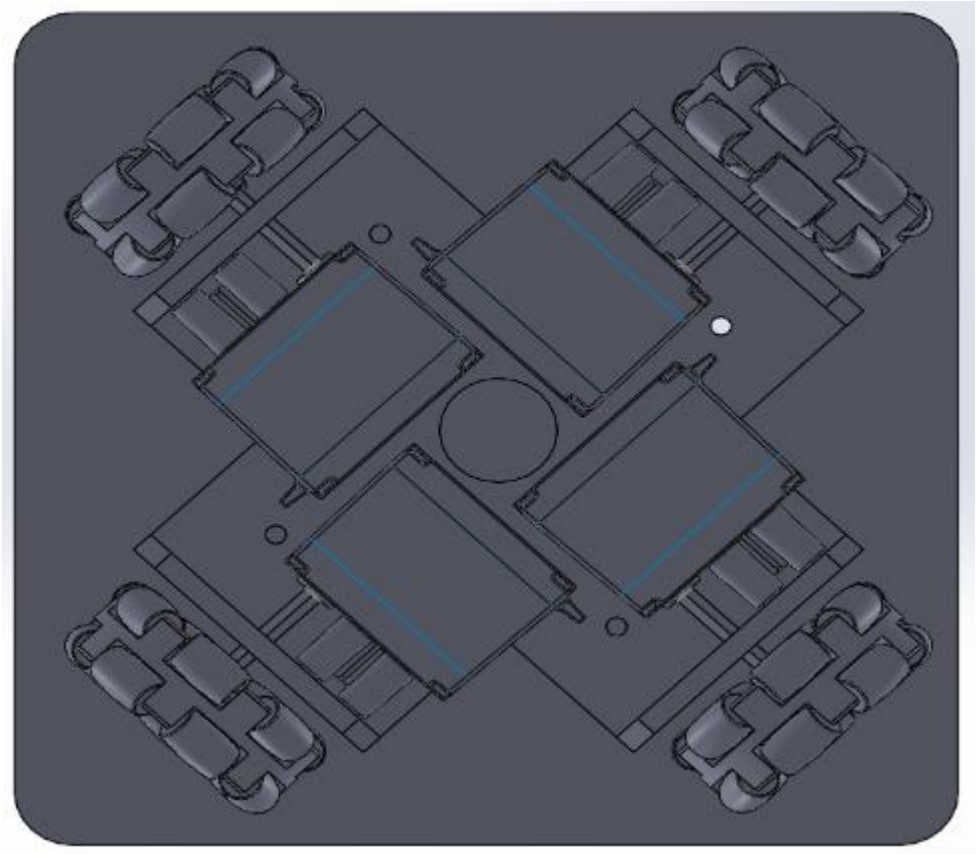


Figure 11: Direct Drive, Four Motor Design.

2.3.4 Flowcharts and Pseudocode

Once a preliminary concept for the design was fully agreed upon, a preliminary concept for the task execution needed to be established in terms of flowcharts and pseudocode. The

programming was centered on a few key functions and header files. By placing all of the functions inside of separate headers, the main code was planned to be very modular, so that it would be easier to see which headers and functions contained errors. The overall plan was to get the crucial functions working, then tweak which ones to call in order to most efficiently complete the tasks. Table 2 below outlines all of the major functions and their purpose in the pseudocode. It was planned for each function to accept multiple arguments so that the desired control could be achieved. For example, a driveUntilTime function would accept heading of travel, power of motors, and time of travel as three arguments. Other functions would require additional arguments. Figure B14 in Appendix B provides a flowchart containing the general flow of the program, placing these functions in the order in which they would be executed.

Table 2: Pseudocode Header Files.

Header File	Functions Associated
Drive	Drive until: bump switch, encoder counts, line reached, time
Turn	Encoder counts, RPS heading angle, time
Ramp	Use drive and turn functions to navigate up and down the temporary access ramp
Microswitch	Detects what switches on which sides have been pressed
Optosensor	Detects line
CdS (Cadmium Sulfide) Cell	Detects red, blue, and no light
Constants	Keep all constants here to organize
Start	Initialize everything, wait for start light with CdS, initial turning and driving setup to reach first location
Toggles	First Toggles = push all switches forward Second Toggles = optosensor to get on line, drive forward to press switch based on RPS
Supplies	Pick up = align with supplies and pick up with servo motor Drop off = line up with receptacle and lower servo
Fuel	Align with fuel light, read value with CdS, drive to side according to data, run into buttons with checkerboard
End	Line up with final button and run into it.

3. Analysis, Testing, and Refinements

The purpose of this section is to provide detail regarding the refinement process that was underwent to move from the preliminary concept towards a final design. To demonstrate this process, the section will give a chronological account of the tests and analyses leading up to the performance tests and individual competition. It will then describe how these analyses led to alterations in the final product, in terms of the code and physical robot.

3.1 Performance Test 1 Refinements

For the first performance test, the robot needed to be able begin motion on a red start light, then leave the start platform. Then, it needed to navigate to the upper surface of the course using either the temporary access ramp or main ramp that was currently under construction. Finally, the stretch exercise for the robot was to correctly read and display the color of the light associated with the fuel delivery system. The two options were red or blue.

In preparation for this performance test, the drivetrain and chassis of the robot were constructed, initially, in accordance to the preliminary concept. Initial testing conducted on February 22 indicated that the robot was able to drive straight in the cardinal and ordinal directions, drive up the ramp, and turn counterclockwise and clockwise on a point. This indicated that the general design of the chassis and drivetrain would likely work for competition purposes. However, a few corrections needed to be made to improve the design. First, shaft lock collars were added to the axles on both of sides of the wheel to prevent the wheel from slipping or falling off. Next, it was noticed that the acrylic was integral to the mounting system of the robot, so it should be structurally supported even further than the acrylic cement. The cost of the acrylic parts was $9.57

in total. Erector supports were hot-glued to the acrylic joints to increase strength and prevent any potential damage. See Table E1 in Appendix E for a full description of the budget.

Next, the Cadmium Sulfide, or CdS Cell, needed to be wired, tested, and implemented onto the robot to read the start light and fuel light color. Once soldered on, the CdS cell values were recorded on February 24. By analyzing this data, thresholds were determined for what the robot would consider a red light or blue light. The results of this testing and analysis are given in below in Table 3.

Table 3: Key CdS Readings and Thresholds.

Light Color	CdS Reading (Volts)	Established Robot Threshold (Volts)
Red	0.36	0-0.99
Blue	1.26	1-1.8
General Surface of Course (No Light)	2.5	Everything Else

Based on the robot testing over the course of several days, it was determined that the robot code would need to be modified to achieve greater consistency. First, the robot relied on time for a lot of movement and turning functions, which led to inconsistent navigation. Adding in the use of the Robot Positioning System, or RPS, and relying more on microswitches hitting the walls would help improve consistency. Also, it was noticed that the robot did not drive completely straight at all times, which was a problem when it was supposed to ride along the acrylic wall leading up to the temporary access ramp, keeping the microswitches pressed. To fix this issue, the travel heading of the function was modified to drive the robot slightly into the wall while still moving forward.

Once these issues were sorted out, the robot attempted to complete the performance test. It failed three times, but a few key observations and refinements came out of the tests. First, the sharp edges of the PVC base needed to be rounded because the edges consistently caught on various objects around the course. Second, the microswitches used to navigate the course were not flush with the edge of the PVC. This caused them to be pressed when the robot was not actually to a wall yet, which made riding along walls quite difficult. This issue was resolved by screwing the switches to the chassis, ensuring that they were always flush. These revisions led to the robot's successful completion of the performance test, with the stretch.

In order to complete the performance test, the robot followed a specific path. First, the robot left the starting block at signal of the red light and ran directly into the wall to the right. Then, it rode along the wall until its bump switches were no longer activated. Afterwards, it navigated the access ramp using bump switches. Finally, it left the access ramp, rode along the wall up to the fuel delivery light, and read the color of the light. A diagram of the path taken is given in Figure A8 in Appendix A.

Finally, it was noticed that there was only $64.33 remaining in the budget after the completion of the first performance test on February 26. This was concerning in that 62.5 percent of the available budget had been spent, but none of the mechanisms, shaft encoders, or the RPS mount had been installed. Realizing this caused a small shift in strategy. All of the mechanisms would be the same as in the planning phase; they would just need to be done with cheaper materials such as paper or already owned materials such as PVC. A line chart depiction of the budget over time is given in Figure E1 in Appendix E.

3.2 Performance Test 2 Refinements

For the second performance test, the robot needed to push one switch toward the upper level, push one different switch toward the lower level, and drive to the upper level at some point during the test. The stretch objective was to press one or both of the fuel buttons. The color of the light did not matter for this performance test.

An initial concern came up during the beginning of testing. The drive surfaces of the robot occasionally became uneven coming off of the start platform. This caused the robot to drive uncontrollably and unpredictably while trying to leave the platform. In the long term, RPS heading and position checks would need to be included in the code to make sure that the robot was where it needed to be at the start of a run. For now, the motor power was just increased to ensure that the robot had enough momentum to leave the platform properly.

Next, it was discovered that the current arm mechanism that flings forward works, but it does not have the power to push all of the toggle switches forward while riding along the wall. To account for this, a strategy revision must occur in that the robot was to drive into the wall every time to press the switch forward. Additionally, the paper mount for the switch arm was replaced with bent Erector to provide a more consistent press. Lastly, the method for aligning with the switches is currently time. This produced several failed practice runs and several good ones. It was determined that adding an optosensor or RPS check into the design would help ensure proper alignment.

Once these refinements were made or noted for future implementation, the robot was able to complete the performance test with the stretch bonus on its first attempt. Although the robot was successful, the reliance on time for almost everything was still quite concerning. For this reason,

rotary mechanical shaft encoders were added to the design idea to help with more fine-tuned alignment.

The following path was taken by the robot to complete the performance test. First, the robot left the start platform on the signal of the red light. Then, the robot used the walls to navigate to the leftmost switch, pressing it forward. Then, the robot navigated along the switch wall and eventually to the temporary access ramp. Next, the robot navigated up this ramp. Then, the robot traveled over to the toggle switches from at the top of the course, pressing the rightmost switches forward. Finally, the robot traveled over to the fuel buttons, running into both the red and blue buttons. A full image of this path is given in Figure A9 in Appendix A.

3.3 Performance Test 3 Refinements

For the third performance test, the robot needed to be able to first leave the start platform on the red start light. Next, it needed to travel to the cargo storage area and pick up the supplies. Then, it needed to carry the supplies to the upper surface and release them into the receptacle. Finally, the stretch activity was for the robot to drive back down to the lower surface.

In preparation for this performance test, much work needed to be done in testing and refining the functions regarding turning and moving until RPS headings or positions. This meant that the robot should turn until the mounted QR Code was in line with the desired alignment. First, it was discovered that the current RPS turning algorithm did not properly account for turning through the 0° mark, where it often oscillated at this mark. Next, moving to an X- or Y- position on the course posed difficulties in terms of speed. If the robot traveled too fast when entering the desired range, with a motor power greater than 35 percent, its momentum would carry it past the region. However, if the robot traveled too slow, around 20 percent, it would start to rotate based on whichever wheels

had the best traction. To account for this, the code was modified such that the robot always realigned itself using a wall once it drove slowly to a position. This helped ensure both robot angle and position were correct.

The next class of refinements needed to occur based on the mechanical locking arm design. The purely mechanical design was not going to work in that the bottom wheel necessary for the robot to lift and lower the dumbbell to the necessary heights protruded away from the robot's base and got caught on other objects. To fix this, the team changed the design to a servo-powered arm that picks up the dumbbell using magnets. This design was much easier to implement. PVC supports were added to help the magnets pick up and secure the dumbbell, as it fell off at higher angles during testing. Once the servo angles and RPS positions were found for pick up and release, the new arm performed consistently in testing.

One final strategy refinement was made towards the end of the pre-performance test testing phase. The tests showed the robot took approximately 8 seconds to navigate up and down the acrylic ramp, and this would not be sufficient for the robot to most efficiently complete the course. For this reason, the team decided to send the robot down the construction ramp, using the motors as a sled to get over the construction bump. This saved a helpful amount of time in testing, and it prevented the robot from losing RPS data.

With all these revisions, the robot successfully passed the performance test on its first attempt. First, the robot left the start platform on the red light. Next, it traveled towards the supplies and aligned itself properly using RPS data and microswitches with the walls. Then, it traveled to the upper surface on the temporary access ramp and dropped off the supplies. Finally, the robot aligned itself with the main ramp and sledded down it. A full course diagram of this performance test run is given in Figure A10 in Appendix A.

Immediately following the completion of this performance test, a budget analysis was conducted. It was determined that there was $17.41 remaining in the budget as of March 9. However, a VEX 393 motor, of which the robot was using four, costed $15.00 in the FEH Store [6]. Because the team wanted to leave spare budget for this motor in the case of a failure, shaft encoder and optosensor plans were scrapped from the design unless absolutely necessary. See the budget in Table E1 in Appendix E for the state of the budget at this point.

3.4 Performance Test 4 Refinements

For the fourth performance test, the robot needed to be able to first start on the start light. Also, the correct fuel button was to be pressed and held for five seconds, as indicated by the light on the ground. Finally, the robot had to drive down one of the ramps and press the final button. Additionally, bonus was awarded for controlling the supplies by the end of the run. This meant that the supplies could not be in the starting bin.

In preparation for this performance test, the biggest thing that needed to be fixed was turning until a specific RPS value was reached. This was determined to be critical in reducing error and lining up correctly with the objectives. The code for this was changed to focus on which way the robot was going to turn, rather than actually getting to a particular angle. Once the robot was able to determine this, coding it to stop once at the correct angle was much easier.

Using a lot of RPS checks while running the motors at about 25 percent takes much more time than driving at 80 percent power for a time. For this reason, driving until a time was combined with driving until a desired RPS value was reached. The robot would drive or turn for a time that was expected to achieve the desired value, then the location would be tweaked using RPS. The same idea was applied to driving into a wall. The robot would drive towards the wall at higher

power for a time, then decrease the power once near the wall to reduce the force of impact with the wall.

Due to the strategy change to be more reliant on RPS values for navigation, key locations on the course needed to be recorded. A diagram giving the desired locations in order is given in Figure A11 in Appendix A. A full list of these values and location descriptions is given in Table A2 in Appendix A. These values include those relevant to the performance test and others needed for the individual competition such as the fuel light area. These checks helped ensure that the robot was in the right relative location before moving on throughout the course in an effort to reduce error.

The last major issues that needed to be corrected prior to the performance test dealt with the electrical systems. One of the motor wires was not tightly screwed into the motor terminal, causing a poor connection to be established. This meant that the motor would receive current only on occasion, causing it to stop working occasionally. This issue informed the team that all electrical connections need to be tested intensely before the performance test and competitions in order to ensure that a hardware issue did not cause poor performance.

Following the testing and refinement process, the robot was able to successfully complete the fourth performance test on its first attempt. First, the robot started on the start light and drove over to pick up the supplies. Then, it drove up the temporary access ramp and stopped on the fuel light, holding the appropriate button for five seconds. Finally, the robot drove down the main ramp and ended the run by pressing the final button. A full diagram of this run is given in Figure A12 in Appendix A.

In order to complete this performance test, no additional money was spent. This was a good indication because this meant that enough money was still available for a VEX 393 motor in case of damage or failure before the individual or final competitions. Additionally, each of the

performance tests were completed with full credit, so it was unlikely that much more spending needed to occur in order to achieve success.

3.5 Additional Refinements before Individual Competition

For the individual competition, the robot needed to complete the entire course as described in the first two sections of the report. As a reminder, Table A1 contains a full description of the primary and secondary tasks that needed to be completed in order to achieve points. The remainder of this section contains the refinements made before the individual competition, while the following section describes the individual competition procedure, strategy, and results.

In preparation for the individual competition, the switch toggling arms needed to be revisited. First, the shorter switch arm was moved to the side of the robot with the checkerboard in order to prevent the need for turning after releasing the supplies into the receptacle. By eliminating this turn, a turn in a section without RPS was removed, and time was saved. Additionally, the longer switch toggling arm was changed in that the paper on the end of it was replaced by an Erector set piece with manually cut PVC strips on it. This was done to improve the strength and reach of the arm, allowing it to press both the red and white switches in one motion. Lastly, it was noted in testing that the robot consistently hit the red switch on the top of the course, even when undesired. To account for this issue, the code was changed to have the robot continue driving until it reached a specific RPS value.

The next major set of changes dealt largely with the use of the Robot Positioning System, or RPS. The first major issue appeared in testing on March 28th, where the robot started to drive the incorrect direction on the temporary access ramp. This occurred because the robot received a value of -1 from the RPS here. To account for this, the drive until RPS functions were changed such that

the robot will continue driving in its current direction if it is receiving this value. Next, it was discovered that the RPS values differ between each course and the day that it is tested on. In order to account for this inconsistency, code was added to the start function to log values that overwrite the default values given earlier in Table A2. In order to ensure that the robot could be ready to run within one minute, the number of values logged was reduced to the three most crucial positions: the dumbbell pickup, fuel light, and main ramp. By logging these values, the robot was able to perform more consistently in these precision-heavy locations.

There were also other refinements made to fix other problems that arose during testing. First, a function was created that allowed the robot to read CdS values while driving. In doing this while driving up to the fuel light, the robot was able to determine if the light was red and act accordingly. If not, the robot just performed its usual light check. The inclusion of this code produced improved accuracy in reading the light and reduced time pressing the red fuel button. Also, a 30 second timeout was added in case the robot did not read the start light so that points could still be obtained. Lastly, it was noted that poorly insulated wires caused the Proteus microcontroller to short out and restart. To prevent this from being an issue during the competition, all poor wiring jobs were redone and reinsulated.

From a budget standpoint, one purchase needed to be made after this testing. A microswitch broke because it got caught on the weather ball station. This switch was replaced by spending $1.20, which although unfortunate, kept the team above the $15 needed to replace a VEX 393 motor in case of failure or damage. The full description of the budget is shown in Table E1 in Appendix E.

4. Individual Competition

The individual competition for Team B5: Leeroy Jenkins took place in Hitchcock Hall, Room 208 on Friday, April 1, 2016 from 8:00 a.m. to 10:05 a.m. The competition occurred between only the teams of students in Dr. Richard J. Freuler's 8:00 Fundamentals of Engineering for Honors class. Each team was awarded a grade ranging from 0 to 75 points in accordance with the primary tasks outlined by FEH Staff [1]. Additionally, the secondary tasks and time needed to complete the course factored into the seeding of the final competition and 15 grade points associated with being the best performing team in a particular section [1]. As described by the staff, the three most important criteria were best score, best average score, and best time in that order. The order in which the teams would run was determined randomly.

Each team was given three runs on the course. The first run was a completely randomized run with the switch order, fuel light color, and course being unknown to the competitor prior to arrival. For the second run, the instructor got to choose whichever of these options they desired prior to the run. For the third and final run, the teams were able to choose all of the run criteria before setting their robots on the course. Each team had two minutes to report to their assigned course once their team number was called. Additionally, each team was given one minute to complete all set up tasks necessary for their robot to complete their run [1]. Finally, the robots needed to complete the course within two minutes of the start light signal [1].

4.1 Strategy

Before the competition on April 1st, a strategy needed to be established to ensure full preparation for any potential or probable errors or issues. For this reason, the strategy used for the

individual competition was split into four major sections: on-course, pre-run, active run, and post-run. These sections are described in greater detail below.

4.1.1 On-Course Strategy

In terms of the on-course strategy, the team agreed upon an order in which the robot should complete all of the tasks. First, the robot would drive off of the start platform towards the switches from the bottom. The robot would align itself with the wall and push the red and white switches away, then push the blue switch away by driving into the wall twice. Second, the robot would navigate to the dumbbell storage bin with the servo motor side facing it. The robot would pick up the supplies with its magnetic arm and navigate to the temporary access ramp. Then, bump switches would be used to navigate to the top of the access ramp. Next, the robot would navigate to the fuel light and press the appropriate button for five seconds. Afterwards, the robot would drive to the supplies receptacle and release the supplies from the magnetic arm. From here, the switches were pushed forwards as needed to match the desired switch order. Finally, the robot would navigate down the main ramp and press the final button. A full diagram of the course path is given in Figure 12 on the following page.

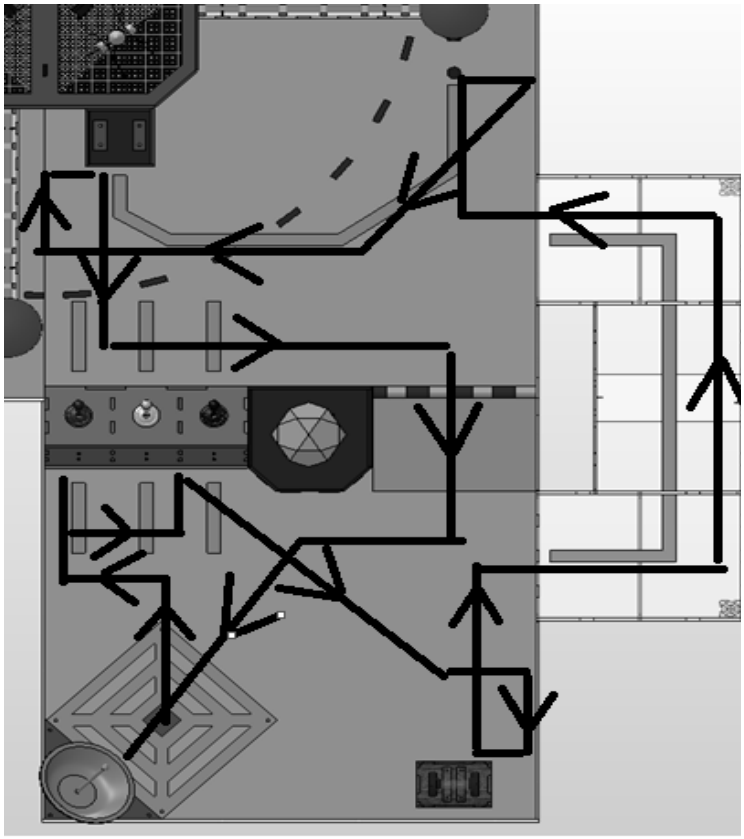


Figure 12: Course Strategy for Individual Competition.

4.1.2 Pre-Run Preparations

To reduce error during each run, a number of preparatory measures need to be taken both on and off of the course. The off-course preparations were charging the Proteus to approximately 12.0 Volts and checking the tightness of the bolts and rubber bands on the switch arms. On course preparations included logging the key locations on the course described in Section 3, checking that the bend angle of the long Erector arm was sufficient to toggle the switches, placing the robot in the proper starting orientation on the start platform, and pressing the left button on the button board once the course master instructed everyone to press their final button. The start orientation was

such that the corner between the checkerboard and long switch arm side was facing the switches with the CdS cell over the start light.

4.1.3 In-Run Strategy

It was also important to assign roles to everyone when the robot was actually on the course in order to cover everything that needed to be done. The roles of each team member during the run are given in Table 4 below.

Table 4: Member Roles during Individual Competition.

Person	Role
Ben	Take notes on the course number, time of run, run time, and issues or success with the run in order to help make changes for subsequent runs on the day.
Jonathan	Take video of the run for documentation and error correcting purposes.
Logan	Kill the robot's run if one of the described scenarios arises.
Paul	Make additional observations on run in order to help with potential improvements.

Additionally, scenarios to kill the robot's run needed to be established because time of run was important in acting as a tiebreaker between runs that scored equally. First, the robot was to be killed if the logged RPS values from the pre-run were incorrect, causing the robot to drive to random locations on the course. Second, the run was to end if the robot got stuck on the weather ball station for five seconds on the way down the main ramp. Third, the run would be killed if the robot was clearly incapable of completing the course, agreed upon by the team on the spot.

4.1.4 Between Runs

Between runs, it was determined that only minor changes to the hardware and software would be made based on observations. Changes to the order in which the robot would complete tasks or changes to major apparatuses would not be made. This decision was made in order to prevent rushing through changes that would only make the robot perform worse.

4.2 Performance

The performance of the robot was based on the score and completion time during the three runs. Overall, the robot placed first in Dr. Freuler's 8:00 a.m. section, receiving the 15 bonus points for grade associated with this result. Additionally, the robot was awarded the 5th overall seed for the final competition. Unfortunately, the robot did not perform as well as expected, as it did not complete three perfect runs. The details of these runs are given in the three subsections below.

4.2.1 First Run – Instructor's Choice

For the first run, the course conditions were determined by the instructional staff. Course B was used, the red fuel light was showing, and the switch order was red forward, white backward, and blue forward. The robot completed the course with a perfect run of 100 points in 1 minute and 26 seconds. The only major concern was that the robot lost contact with the wall at the top of the temporary access ramp early. This meant that the robot entered into its failsafe code of driving slower, adding time to the run. Although it was good to know that the failsafe code worked, entering into it did not help with the run time.

4.2.2 Second Run – Random Assignment

For the second run, the course conditions were determined at random. Course E was used, the red fuel light was showing, and the switch order was red forward, white backward, and blue backward. The robot was not able to complete the course this time, as it got caught on the weather ball station on the way down the main ramp. The robot achieved a score of 92 points, but the time was not recorded. The run was killed in accordance with the previously established kill conditions for the weather ball stand. Other concerns included the robot pivoting on the top of the access ramp and the fact that the robot was close to hitting the dumbbell receptacle.

4.2.3 Third Run – Team Choice

For the third run, the course conditions were determined by the team. The course conditions were identical to the previous run, except that course B was used. The robot was able to complete the course; however the blue switch was not toggled from the bottom for an unknown reason. The end result was 90 points in 1 minute and 28 seconds. Other concerns included the top of the access ramp once again and the line up with the main ramp being very close to ending the run again.

4.3 Analysis of Reasons for Success

There were a few major reasons for the success of the robot during the individual competition. First, the team committed to early and frequent testing from the beginning of the project. The performance tests were consistently completed two days prior to their due date, allowing for reduced stress and earlier preparations for the following performance tests. Additionally, the lack of last second changes to the robot meant that everything was thoroughly tested and most potential errors had been addressed in the code.

There was also a lot of failsafe programming built into the competition code. For example, the only reason that the robot's run did not end on the top of the access ramp was due to the inclusion of an infinite loop for it navigating while the RPS value was less than zero. Second, the adjustment of RPS values between courses helped account for the differences between each course to ensure consistency. Lastly, using frequent RPS X, Y, and heading checks prevented the robot from continuing through the course when in a non-ideal location.

Success can also be attributed to the simplicity of the mechanism used to complete the tasks. There were only two non-mechanical motions made by the mechanisms during an entire run, picking up and releasing the supplies. All the task completions besides handling the supplies were dependent upon the consistent drivetrain of the robot. The robot could consistently drive or turn in any desired way, so driving left or right to press the fuel buttons or switches was less of a concern.

The final reason for success was the preparation taken before the competition. For example, practicing the timing in logging the RPS values before the run helped reduce stress during the competition. Also, having established kill scenarios helped reduce the time wasted in making a decision ending the second run. This helped improve the robot's average time.

4.4 Analysis of Reasons for Failure

There were also a few reasons that the robot and team failed during the individual competition. First, the team did not stick to the strategy of not changing too much between runs. Following the first perfect run, the orientation of the checkerboard was changed in an effort to prevent it from pushing the robot off of the wall. This actually resulted in a near run end with the robot getting stuck here for a few seconds. In general, the team did not have an answer for the issues with the top of the ramp, which was a major problem. The biggest cause for failure, the one that killed the

second run, was that the code was not actually modified to account for the offset at the top of the main ramp. This meant that the changes here did not actually do anything. Overall, the code needed to be thoroughly re-read through to make sure that everything desired was present.

4.5 Potential Improvements

Based on the results of the competition, there were a few potential improvements that were considered. First, the RPS offset needed to be applied to the top of the main ramp which would likely prevent the robot from catching on the weather ball station. As an added precaution, the robot should be programmed to drift away from the station. Next, the issues with the top of the temporary access ramp could be reduced by having the robot drive for a time instead of until the microswitches released. This would prevent the robot from catching on the wall and pivoting or getting stuck.

With these issues addressed, attention should be turned to the switches, which prevented a perfect third run. For this reason, the most testing should be done on these in preparation for the final competition. Tests should be done to ensure that all of the switches can be consistently pressed from both the top and bottom. Also, experimentation needs to be done with pushing the blue switch from the top as the weather ball station and switch platform are not perfectly flush. This could cause the robot to contact the wall a non-square angle. Additionally, reducing the distance for the robot when driving horizontally to the supplies receptacle would reduce the risk that the robot drifts too far and hits it. A final suggestion included changing the start orientation of the robot to prevent it from getting stuck on the start platform.

In general, the reduction of time was prioritized. Although the robot was the fastest in the class, robots in other classes had been recorded with faster times, and teams would continue improving.

Time reduction could be achieved by increasing motor powers at certain areas, not stopping at switches that do not need to be pressed, decreasing the timeout on RPS checks, and increasing the acceptable range on areas of the course where the robot's positioning does not need to be as precise. Additionally, the inclusion of a new drive while turning function would allow the robot to drive and turn in one motion, which would prevent time waste from doing both separately. On a separate note, Velcro was added to further secure the Proteus to the PVC base.

5. Final Design

The final design of the robot was determined through hours of testing, analyzing success and failure during performance tests, and applying the lessons learned from the individual competition to the finished product. This section describes the chassis, drivetrain, mechanisms, and electrical systems final state using SolidWorks models and images of the robot. Additionally, it provides information regarding the final budget, time, and reasons that resources were applied to particular areas. An image of the final robot, The Black Baron, can be found on the following page in Figure 13. Additionally, the completed SolidWorks model of the robot can be found in Figure F1 in Appendix F. The full working drawing set is provided in Appendix G.

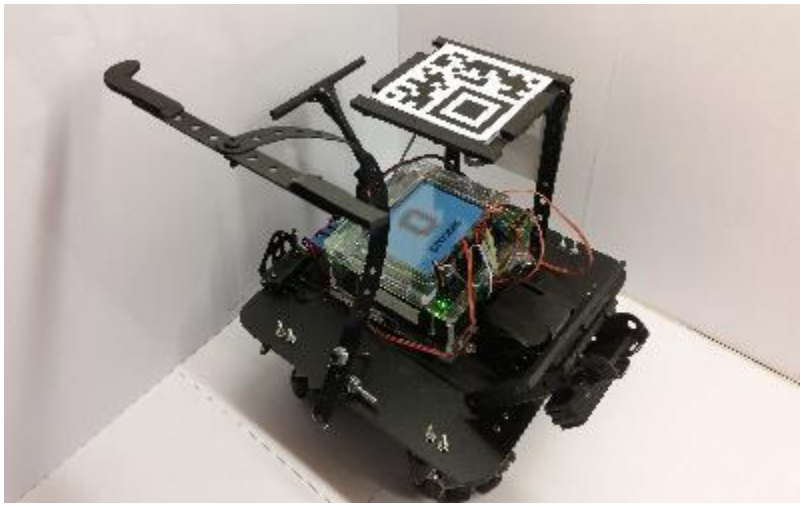


Figure 13: Final Image of the Black Baron.

5.1 Budget

Overall, the robot was able to be built within the $160 budget imposed by the Fundamentals of Engineering for Honors Space Administration with $15.56 remaining. The amount of money remaining was in line with the original goal of leaving at least $15 left in case an additional motor needed to be purchased. In general, most of the spending occurred at the beginning of the project and went into the chassis and drivetrain, with very little spending towards the end of the project.

5.1.1 Budget over Time

As stated in the previous section, most of the spending occurred at the beginning of the project, with 62.5% of the budget being spent by the first performance test on February 26, 2016. This was expected as the entire chassis and drivetrain of the robot needed to be assembled at this point. From here, very little spending occurred until the third performance test preparations began. Here, a Futaba servo motor needed to be purchased, and the arm that attached to it needed to be built. Once this was built, nothing else needed to be purchased except for the replacement of a broken

microswitch and Velcro. A full description of the budget over time is given in Table E1, and the information is displayed graphically in Figure E1.

5.1.2 Budget Breakdown by Part Type

As discussed in the Preliminary Concepts section of this report, the chassis and drivetrain of the robot were the most essential components of the robot. They were intended to simplify the task completion mechanisms of the robot and make navigation as simple as possible. As a result, they accounted for nearly 75% of the robot's total cost. See Figure 14 below for a visual depiction of the cost breakdown by part type. Additionally, see Table E2 in Appendix E for a tabular description of this data.

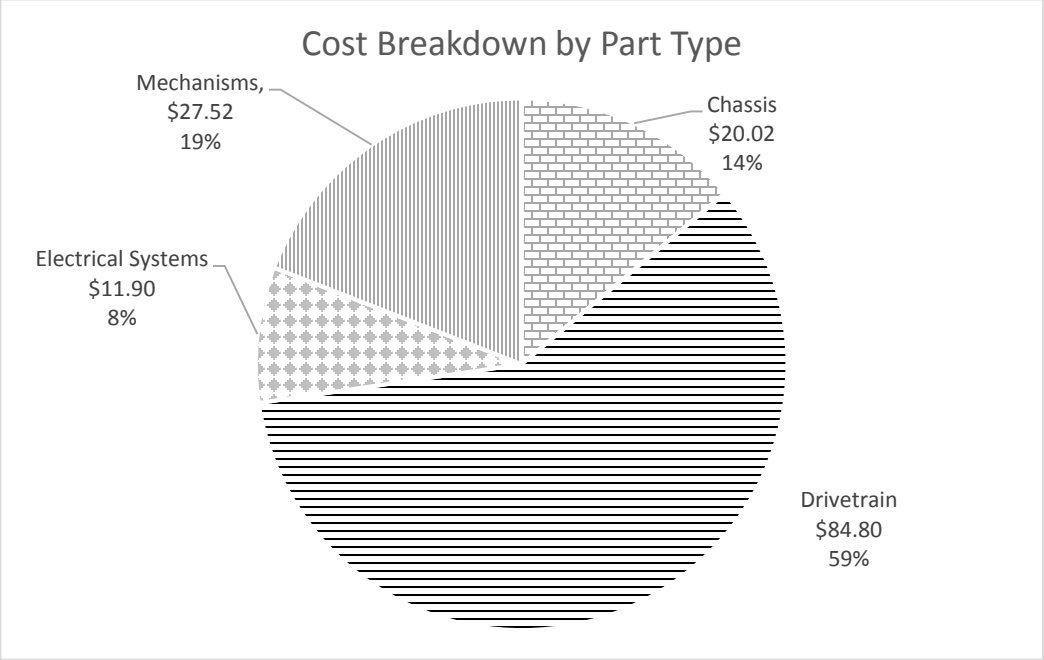


Figure 14: Cost Breakdown of Budget by Part Type.

5.2 Final Design of Chassis and Drivetrain

The robot was supported by an acrylic base. The base was made of five laser cut pieces of acrylic mounted together using acrylic cement. This laser cut chassis contained a one inch diameter hole in the middle for wiring purposes, holes for mounting the motors and PVC base, and holes for the wheel axles to go through. A picture of the acrylic chassis broken down into its components can be seen in Figure F2 in Appendix F. At the points of connection between the four fins of the chassis, and the large base piece, there are Erector set right angle brackets. These brackets help support the connection between the acrylic. One side of the bracket is attached to the fins with hot glue, and the other side is bolted to the large acrylic base.

The robot used four Vex omnidirectional wheels. An image of these wheels can be seen in Appendix F, Figure F3. Each of these wheels was driven by a VEX 2 Wire 393 DC motor. These motors were mounted directly to the fins on the acrylic chassis with Vex motor mounting screws. A Vex axle, placed through a hole in the acrylic connected the omnidirectional wheels to the Vex motors. These axles were held in place with two shaft locking collars. One collar was placed between the acrylic and the Vex motor; the other was placed outside of the wheel. Also, a 1/8 inch spacer was used to separate the wheel and the acrylic to prevent rubbing. An image and an exploded view of the motor and wheel connection can be seen in Figure F4 in Appendix F.

A Polyvinyl Chloride (PVC) sheet was placed on top of this acrylic chassis. It was connected with the same bolts that connected the angle brackets to the acrylic base. This base was 8 by 8 inches with each corner rounded by a radius of one inch. The PVC sheet was mounted so that each corner was over top of one of the four wheels. In order to gain clearance from the wheels, two ¼ inch spacers were used between the acrylic base and PVC sheet, elevating the PVC ½ inch from the acrylic base.

5.2.1 Navigation and Driving

The biggest advantage of this drivetrain design was the ability to drive at all headings, meaning that this robot could drive in the direction of North, East, South, West, and everywhere in between. This was made possible by the omnidirectional wheels designed for that function. When a certain heading was desired, vector calculations were done inside the code which powered the four motors in such a way as to drive at the desired heading. See the Final Code section of this report for more details about these functions and calculations.

With the Vex motors mounted underneath the acrylic chassis, there was not enough clearance to get up the main access ramp, so the temporary access ramp had to be used. Although the robot could not make it up the main access ramp, it could make it back down. The Vex motors were used, like a sled, to slide over the bump on the main ramp. This decreased the time it took the robot to complete the course because going down the access ramp required the robot to travel less distance.

5.2.2 Cost

This drivetrain and chassis system contained 5 laser cut acrylic pieces, one PVC sheet, eight ¼ inch spacers, four 1/8 inch spacers, four Vex motors, eight shaft locking collars, four Vex axles, four 1.5 inch number 6 screws, and four Vex omnidirectional wheels. This drivetrain and chassis system consumed 65 percent of the $160 available. The pie chart shown on the following page in Figure 15, gives the breakdown of the $104 cost.

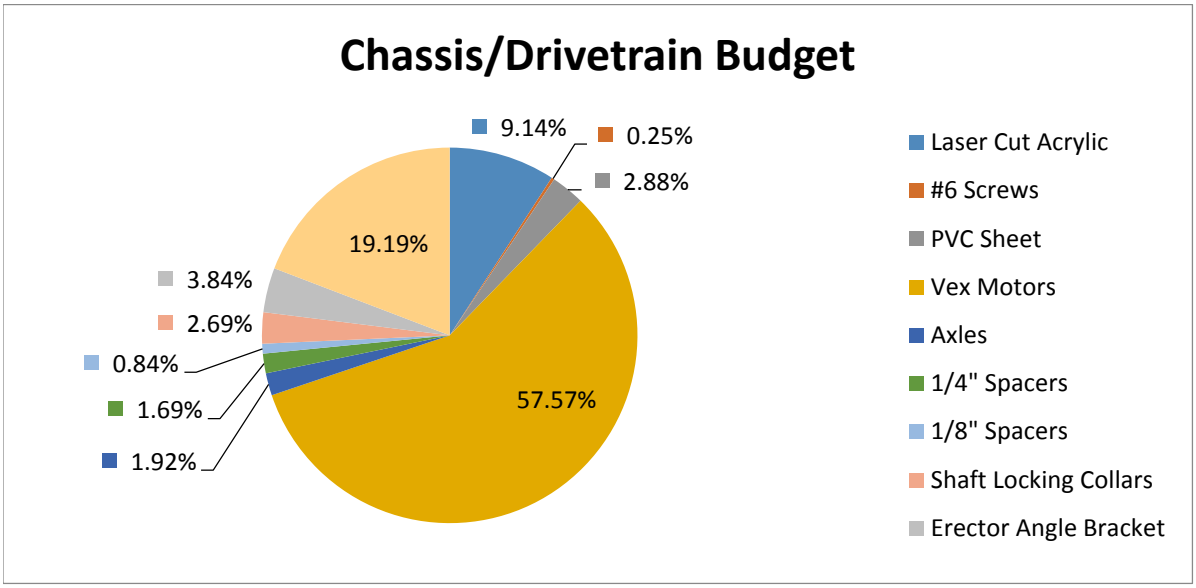


Figure 15: Detailed Breakdown of Budget Allocated to Chassis and Drivetrain.

5.3 Final Design of Electrical Systems

The electrical system of the robot was split into three main categories: general-purpose input/output, or GPIO, drive motors, and servo motors. The GPIO category of the electrical system was made up of eight roller blade microswitches, one CdS cell, and an FEH Button board. Two bump switches were mounted on each of the four sides of the robot, and were used for bump navigation. These microswitches were named microswitch 1 through 8, going clockwise starting from the northwest corner of the robot. The CdS cell was mounted onto the bottom front of the robot to sense the starting light and both colors of the fuel light. The wires for six of the microswitches and CdS cell were brought up through a hole in the center of the chassis and connected to the Proteus. Two of the microswitches, mounted to the top side of the chassis, were connected directly to the Proteus.

The drive motor category was made up of four VEX 2 Wire 393 DC motors. These motors were mounted onto the acrylic drivetrain of the robot, and directly drove the wheels. Starting from

the northwest corner and going clockwise in a manner similar to the microswitches, these motors were named Motor1 through 4, and were wired into motor ports zero through three on the Proteus. The wires of these motors were brought up through a hole in the center of the drivetrain and chassis and connected to the Proteus.

The third category of the electrical system, servo motors, consisted of only a single Futaba servo motor. This motor was secured to the top on the robot chassis using PVC, hot glue, and tape, and it was used to control the dumbbell supply arm. The servo was wired directly to the Proteus. A complete wiring diagram for all three categories can be seen in Figure F11 in Appendix F. Additionally, tables describing each of the GPIOs and motors in more detail can be found in Table F1 and Table F2 in Appendix F.

5.4 Final Design of Mechanisms

The goal with all of the mechanism designs was to keep them as simple, cheap, and mechanical as possible. For this reason, materials such as paper, left over bolts, and already purchased PVC were used. The major mechanisms were the long and short arms used to push the toggle switches forward as needed, the magnetic arm used to control the supplies, and the checkerboard design used to press the fuel buttons. The short switch arm and offset trunnions were placed on the front side of the robot with the CdS cell. The long switch arm was placed on the left side of the robot relative to the front side, and the magnetic supplies arm was placed on the rear side of the robot. These designs, which were largely contingent on the drivetrain of the robot, are described in greater detail in the subsequent subsections.

5.4.1 Switch Toggling Arms

To toggle the switches, it was determined that it would be easier and more consistent to push the switches from both sides of the course instead of pulling them. One arm pushed all of the switches from the bottom of the course in as few motions as possible, while the other arm pushed switches back from the top as needed to match the RPS data.

The first arm used, the long arm, was made out of Erector, nuts, bolts, PVC, and rubber bands. The main body of the arm was an Erector piece manually curved in a way that allows it to reach the switches from the bottom and stay within FEHSA's size constraints. A horizontal Erector strip was attached to the end of this curved piece. Additionally, hand-machined PVC strips were glued to the ends of this horizontal Erector to add extra length and consistency to the apparatus. The arm pivoted about a horizontal screw mounted on the top of the PVC base. Finally, a rubber band was used to pull the arm back so that it did not protrude out and interfere with other parts of the course. A SolidWorks model of this arm can be found in Figure F5 in Appendix F.

The short arm was constructed very similarly to the long arm, but it was made using paper instead of Erector. This decision was made because this arm did not extend as from its pivot point, so it did not require as much strength or flexibility as the Erector model. Additionally, using paper only costed $0.01 while another Erector strip would have costed $0.89. An image of the short arm's final design is shown in Figure F6 in Appendix F.

5.4.2 Air Filtration Supplies

To transport the air filtration supplies from the bottom storage bin to the top receptacle bin, a mechanical arm controlled by a servo motor was designed and built. The arm featured two magnets attached to the end of angle girder Erector pieces with hot glue. The purpose of these magnets was

to secure the dumbbell, which contained metal washers inside of it and release it into the receptacle by scraping it off when driving away. However, the magnets were not strong enough on their own, as noted in the Analysis, Testing, and Refinements section of the report, so PVC strips were cut to support the dumbbell at its highest point. This arm was controlled by a Futaba servo motor, with different angles used to pick up, hold, and scrape off the supplies. An image of this design can be seen in Appendix F, Figure F7.

The major strategy for this design was to minimize the number of electrical motions that needed to occur to improve consistency and reduce wear on the servo motor. This was done by using the robot's maneuverability to get the arm into the exact location needed to just raise and lower. From a materials standpoint, the design was kept to Erector, PVC, and magnets, which helped keep the cost within the $160 budget.

5.4.3 Fuel Delivery Apparatus

The final design and strategy of the fuel delivery apparatus relied almost entirely on the drivetrain of the robot. The strategy was to read the fuel light and drive either drive straight or to the right depending on the color of the light. If the light was blue, the robot would drive straight to press the blue button using the trunnion mounted on the bottom of the PVC base. If the light was red, then the robot would align itself with the wall and drive forward to press the red button with the trunnion mounted on the top of the base. The trunnions were spaced out far enough to allow for one half inch of error in pressing the buttons. An image of this design can be found in Figure F8 in Appendix F.

From a time saving and strategy standpoint, it was determined that it would be advantageous to try and read the fuel light while driving up to it. If the CdS detected a red light, then the robot

would stop, run into the wall to the right, and press the red fuel button. If the CdS cell detected a blue light or did not detect a light, the robot would go through its usual progression of using RPS to align with the light and act accordingly.

5.5 Final Design of QR Code Mount

The QR Code mount was essential to the final design of the robot because the robot relied very heavily on RPS because it did not have shaft encoding navigation available. The QR mount was placed a little under 9 inches above the ground surface of the course in order to obtain the best data [1]. The mount was made out of two Erector pieces bent at 90° angles at the top and bottom and bolted into the PVC base. PVC pieces were cut and glued to the top of the Erector mount to firmly secure the QR Code in place so that it would not change angle or location during the run. The design was intended to be as cheap and simple as possible in order to account for the large cost spent on the chassis and drivetrain parts of the robot. An image of the mount can be found in Figure F9 in Appendix F.

5.6 Final Code

The final robot was autonomously controlled using C++ code run on a custom Proteus microcontroller. The software package was modular, utilizing 37 separate source and header files. The use of several source and header files offered many advantages. The code was easy to maintain and was organized. Separate source and header files could be copied and reused without modifying them. In addition, the compilation time was decreased. The final robot code is included in Appendix H.

5.6.1 Software Structure

The software was broken into five tiers. The first and second tier included the main program and the touch screen menu. The menu accessed the sub-programs, which were part of the third tier. A few sub-programs used during development included a Proteus test, a world state program, a competition program, or a test program. The competition code accessed the task completion functions, which were part of the fourth tier. These functions controlled the robot on the course to complete the course objectives. The code for the robot course tasks were broken up into six sections: start, toggles, supplies, ramp, fuel button, and launch button. Each section contained the functions to complete each task. The final tier was comprised of the basic robot control functions. These offered definitions for motors and sensors, drive functions, a world state logging function, and more. A structure of the software can be seen below in Figure 16.

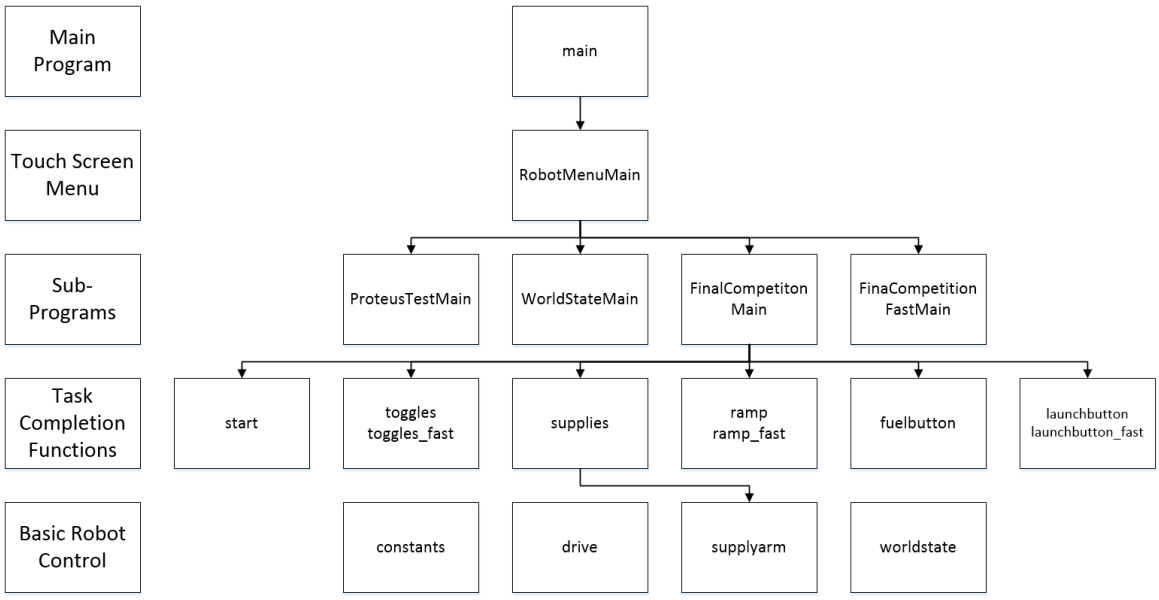


Figure 16: The Structure of the Software was Modular and Tiered.

5.6.2 Code Functions

The robot task completion functions were designed to divide the course into different sections. These functions executed other functions that controlled the different parts of the robot. Table F4, located in Appendix F, outlines where each function was located and what it did. The functions are further described in the commented code.

The drive code included 11 defined drive functions. These functions accepted many arguments for easy modification. Some of the common arguments would be the heading, power, and time out. The heading declared the direction the robot should drive which was calculated using a motor ratios function using trigonometry. A plot of the motor percentages against the heading is shown below in Figure 17. The specified power was multiplied by the motor ratios to alter the speed of the robot. The time out allowed for the function to stop prematurely in the case of the unexpected.

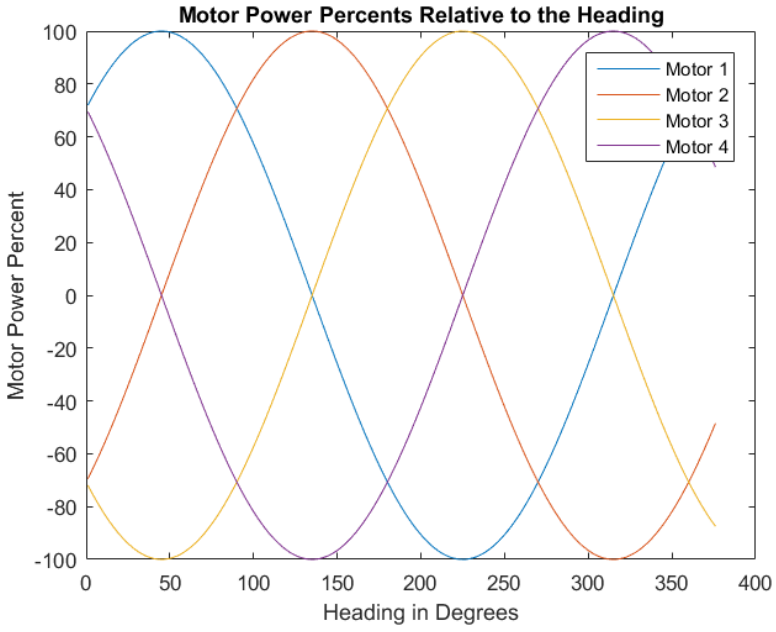


Figure 17: Sinusoidal Curves of the Motor Percentages as the Heading Changes.

5.6.3 Version Control

The code was developed and maintained using version control. A Git repository, hosted by Bitbucket, was used to backup and track changes in the code, which can be found at https://bitbucket.org/jonliew/fehrobotcode. The repository provides detailed changes during the development of the code. Additionally, the code was automatically saved to a Box folder, which provided additional versioning and backup. This allowed for the entire team to access the code and restore the desired versions.

5.7 Final Schedule and Time

Overall, the team underestimated the total number of hours that the project would take and the total number of hours that would be spent on each major area. It was estimated that the project would take 301.5 total man hours to complete, while the actual time logged was 439.3 man hours. This underestimation of 137.8 hours occurred largely due to the unexpected amount of time that went into the planning process of the design, which differed by 57.01% from the expected value. Additionally, it was interesting to note what parts of the project took the greatest amount of time. Documentation, testing, and planning took the most time while building took the least amount of time by 13 hours. A breakdown of the time predicted and spent on each category is given in Table 5 on the following page. A pie chart break down of hours by category is given in Figure F10 in Appendix F.

Table 5: Breakdown of Hours Spent by Category.

Type of Work	Estimated Hours Spent	Actual Hours Spent	Percent Difference
Planning	39.5	71	57.01 %
Documentation	104	161	43.02 %
Building	44	50.5	13.76 %
Programming	54	63.5	16.17 %
Testing	60	93.30	43.44 %
Total	301.50	439.30	37.30 %

In terms of the design schedule, the team was consistently behind on the estimated start dates, but the due dates were all achieved reliably. It was determined that the original dates were at times early in that other parts of the robot or documentation were not yet completed that needed to be. However, setting early start dates for everything helped the team stay ahead on performance tests and assignments for the most part. The final design schedule can be found in Table F3 in Appendix F. Due to size constraints, the schedule is restricted to merely deliverables, desired dates, and actual dates.

6. Final Competition

The final competition for Team B5: Leeroy Jenkins occurred at The Ohio State University's Recreation and Physical Activities Center on Saturday, April 9, 2016. The team arrived at 10:00 a.m., and the awards ceremony was completed at 6:00 p.m. The competition occurred between over 70 teams from the Fundamentals of Engineering for Honors course at Ohio State. In the same way as the individual competition, each team was awarded a grade ranging from 0 to 75 points based on the level of completion of the primary objectives [1]. Additionally, there were 25 secondary points, not associated with the team's grade, used for scoring. A full description of the scoring can be found in Table A1 in Appendix A.

The format for the competition was as follows: Each team competed in three round robin matches which were used to determine their grade for the competition. Then, based on the seeding from the individual competition, the head-to-head competition began. In each match, four robots competed, and the top team advanced out of each stage. In total, there was one play-in round and three single elimination rounds. As with the individual competition, each team had one minute to setup their robot for the run and two minutes to actually complete the run [1].

6.1 Strategy

For this competition, the team employed a strategy nearly identical to the individual competition. The on-course strategy and order of task completion remained identical. Additionally, the in-run strategy and team member roles remained the same. The only differences in strategy were to the pre-run strategy and what changes were allowed between runs. As discussed in the Individual Competition Section, the robot's start orientation was changed. Additionally, issues were encountered as the team made changes to the robot between runs. As a result, it was determined that no changes to the code or hardware would be made upon arrival at the RPAC.

6.2 Performance

Overall, the robot performed quite well during the seven runs that it had during the final competition. The robot achieved a perfect score of 100 points in each run with an average run time of 1 minute and 12 seconds. In the pool play rounds, the robot received second place overall for being the second most consistent robot. During the knockout stage of the head-to-head

competition, the robot made it into the final four round, placing second overall. A full description of the run results is given in Table 6 below.

Table 6: Description of Runs at Final Competition.

Run Description	Course Designation	Time of Run	Score	Time (min:sec)	Result
Practice Run	B	11:30 a.m.	100	1:14	Robot looked ready to go
Pool Play 1	A	12:00 p.m.	100	1:12	Earned 75 grade points
Pool Play 2	D	12:30 p.m.	100	1:14	No changes needed
Pool Play 3	C	1:00 p.m.	100	1:09	No changes needed
Knockout Stage 1st Round	C	4:00 p.m.	100	1:15	Won heat and advanced
Sweet Sixteen	A	4:30 p.m.	100	1:09	Won and advanced
Final Four	D	5:05 p.m.	100	1:12	2nd Place

Overall, the robot performed well, but the robot made a few unexpected and undesired actions. For example, the robot did not read the red fuel light on its first attempt during a few of the runs. This hurt the time by about 4 seconds. Additionally, the robot frequently drove well past the desired location off of the temporary access ramp. This meant that it needed to spend approximately 5 seconds readjusting its position to the desired location on the course using RPS.

6.3 Analysis of Reasons for Success

The success of the robot during the final competition can be attributed largely to the survivability to the robot code. As noted in the Final Schedule and Time Section, over 90 hours were spent testing the robot on the course, discussing changes to be made, and implementing failsafe code in case of major error. This was crucial in areas such as having multiple ways to

check the fuel light, aligning with the main ramp, and ensuring that the robot left the temporary access ramp. Additionally, key RPS locations on the course were logged and used in run instead of preset values. This helped to make sure that the robot had the necessary precision when picking up the supplies and reading the fuel light.

The physical design of the robot also played a large role in team success. The drivetrain used allowed the robot to have good maneuverability and reliable navigation, which was a major factor in the robot's success in the competition. The robot's mechanisms were designed to be as simple as possible, which helped achieve the perfect consistency with task completion. From a strategy standpoint, the team focused on consistency over time, which allowed the Black Baron to continue advancing in the head-to-head tournament. Some robots were faster but failed to achieve a perfect score.

6.4 Analysis of Reasons for Failure

Overall, there were not too many failures for the robot at the competition as it achieved a perfect run every time. However, the biggest cause for not winning the competition was the time that it took the robot to complete the course. Although the emphasis was on consistency, improving time would have allowed the robot to win the competition. The inability of the robot to travel up the main ramp with the construction bump added approximately 10 seconds in comparison to many of the competitors. Also, VEX motors provided inferior power when compared to the Igwan motors that many teams used, which put the robot further behind in the speed category.

7. Summary and Conclusions

The purpose of this section of the report is to briefly summarize all of the information in the report up until this point and provide analyses into improvements that would be made if the team got the opportunity to restart the project from scratch.

The project was completed by Team B5: Leeroy Jenkins, which consisted of Paul Harshbarger, Benjamin Higgins, Jonathan Liew, and Logan Meyer. The purpose of the project was to build a fully functioning prototype robot to complete a set of predetermined tasks for the Fundamental of Engineering for Honors Space Administration's rocket launch preparation site. There were a few checkpoints that indicated the tasks that the robot needed to be able to achieve at certain dates. On April 9th, the robot demonstrated its ability to complete the course in a head-to-head competition with other design companies to FEHSA. Visit https://u.osu.edu/feh16b5/ for additional project information.

7.1 Summary

At the beginning of the project, the team underwent a brainstorming process that involved individual and group components. In this process, several design concepts were generated, and the best ideas were identified by comparing the concepts to a reference. From this list, three final design ideas were created and ranked. The chosen design was then modeled physically and in SolidWorks to allow the team to analyze how plausible and effective it would be on the course. After a few alterations, the final design featured an omnidirectional drivetrain, an acrylic chassis, and a PVC base on which all of the motor-free mechanisms were to be mounted.

Throughout the testing and building process, several key refinements were made to ensure success in the competitions. For example, the corners of the PVC base were rounded to prevent

the robot from catching on the corners of the acrylic temporary access ramp. Also, spacers were added between the acrylic chassis and PVC base to raise the height of the mechanisms so they could better complete tasks. A major change occurred when the mechanical locking arm design idea needed to be scrapped in favor of the final magnetic arm. In terms of time, a decision was made to go down the main ramp in order to save valuable seconds. Lastly, a function was implemented that allowed RPS values at key positions to be inserted into the code right before a run. This helped to increase consistency with fine maneuverability.

In terms of the competitions, the robot performed very well, placing first in the class during the individual competition and second overall during the final competition. The issues of catching on the weather ball station and missing the blue toggle switch were able to be corrected before the final competition. By eliminating these issues and adding in more failsafe code and RPS checks, the robot managed to achieve perfect runs in all 7 attempts.

The final design of the robot was similar to the original design idea generated from brainstorming. It featured four individually driven omnidirectional wheels and primarily mechanical mechanisms. The only motorized mechanism was the magnetic arm used to pick up the supplies, but this apparatus only made a few motions on the course. The other systems used were the trunnion checkerboard for the fuel delivery and the two Erector arms for switch toggling. Additionally, a QR code mount was constructed out of Erector to allow the robot to receive RPS data. From a coding standpoint, multiple drive functions existed which allowed the robot to drive at any heading, until a bump switch was pressed, or until the CdS recorded a certain value. Additionally, the robot could drive while turning and turn on a point. Overall, the robot was completed with $15.56 remaining from the $160 budget, with approximately 75% of the budget spent being allocated to the chassis and drivetrain. Additionally, the team underestimated the time

that the project would take by 137.8 hours in terms of building, planning, programming, documenting, and testing.

7.2 Conclusions

Overall, the final design schedule and project time utilization did not match the predictions and plan, but this worked out to help the team succeed with the robot design project. As shown in Table F3, the team consistently fell behind the projected task start dates on the design schedule. However, this was due to the fact that the dates were ambitious and set early in an effort to motivate everyone to get ahead on work. This strategy helped the team stay ahead of the schedule set out by FEHSA, completing three out of the four performance tests early and working on the competition code well before other teams. Additionally, all of the assignments were able to be completed on time.

In terms of time utilization, the team underestimated the total project time by 143.8 hours, with the biggest estimation errors coming with the planning, testing, and documentation aspects of the project. Performing more work in these areas helped the team achieve greater success. By spending 71 hours planning out the design and code of the robot, all of the key details of the robot were able to be solidified before actual construction and programming began. This helped reduce future time waste and prevent major errors. Also, committing 93 hours to testing the robot helped the team discover any potential issues that could arise during the competitions. By developing failsafe algorithms and improving the robot's hardware allowed the robot to be the only one to achieve perfect runs in every final competition run. These were major factors in obtaining success in terms of grades and placing.

As stated above, Leeroy Jenkins was pleased with the outcomes of the project, but a few changes would be considered in future work. As noted in the Analysis, Testing and Refinements

Section of the report, there was a large reliance on time-based navigation early in the process. This caused difficulties when trying to navigate to parts of the course that required precision. Pursuing RPS earlier or adding in shaft encoding as a navigation technique would have helped with these early struggles. Additionally, the inability of the robot to travel over the bump on the way up the main ramp caused it to lose valuable time in comparison to the competition. Pursuing larger diameter wheels would allow the motors to gain the clearance necessary to avoid the construction bump. Additionally, larger diameter wheels would help the robot drive faster in general on the course. Future work would also involve the use of better materials for some of the mechanisms. For example, the use of paper and rubber bands on the switch toggling arms proved to cause some issues during testing. Although reliable in the competitions, using wood or Erector instead of paper would have reduce fail potential even further. These changes would improve the robot's success during a future competition and allow it to move from the prototype stage into full scale production for FEHSA's rocket launch site.

8. References

[1] Spring 2016 Robot Scenario Revision 1. 2016, February 25. www.carmen.osu.edu.

[2] Performance Tests Robot 2016. 2016, February 25. www.carmen.osu.edu.

[3] 2016 Scenario PowerPoint. 2016, February 25. www.carmen.osu.edu.

[4] Course Top View. 2016, February 25. www.carmen.osu.edu.

[5] 3 Inch Fast Linear Actuator. 2016, February 25. https://www.tampamotions.com/products/3-fast-linear-actuator-30mm-s-50lbs-lift-12v?utm_medium=cpc&utm_source=googlepla&variant=7789379589.

[6] Motor. 2016, March 22. https://feh.osu.edu/secure/store/The_Store/store/index.php.

APPENDIX A

The Course and Scoring

Table A1: Scoring System [1].

Primary Tasks	**Points**
Initiate on start light	8
Touch supplies	8
Control supplies	12
Toggle launch sequence switch	8
Toggle launch sequence switch in correct direction	12
Press a fuel pump button	8
Press the correct fuel pump button	11
Press the final launch button	8
Possible Primary Task Points	**75**
Secondary Tasks	
Deposit supplies	8
Toggle 3 launch sequence switches in correct directions	10
Hold the correct fuel pump button for 5 seconds	7
Possible Secondary Task Points	**25**
Total Possible Task Points	**100**
Penalties	
Knocking the orb off the weather station	-10
Disturbing a competitors' objects	-10
Failure to control objects	DQ
	Points

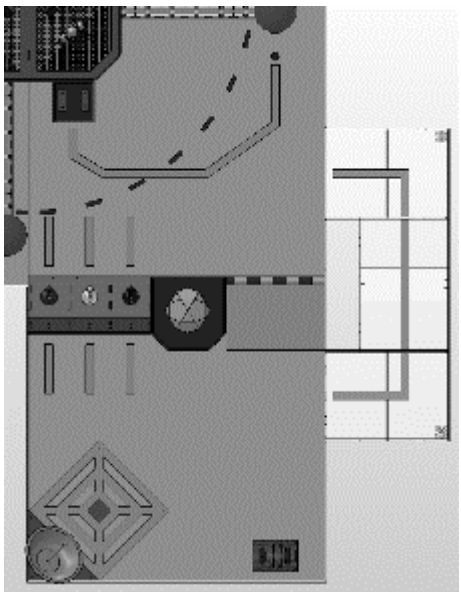


Figure A1: Top View of Course [4].

Figure A2: Main Ramp [3].

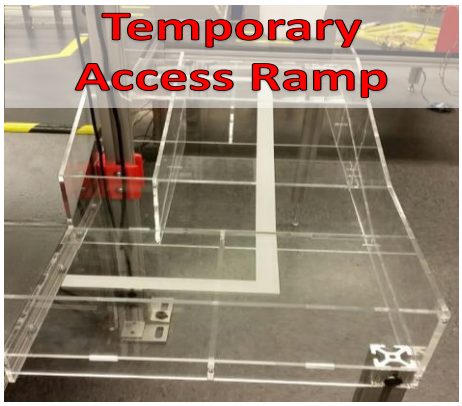


Figure A3: Temporary Access Ramp [3].

Figure A4: Weather Ball [3].

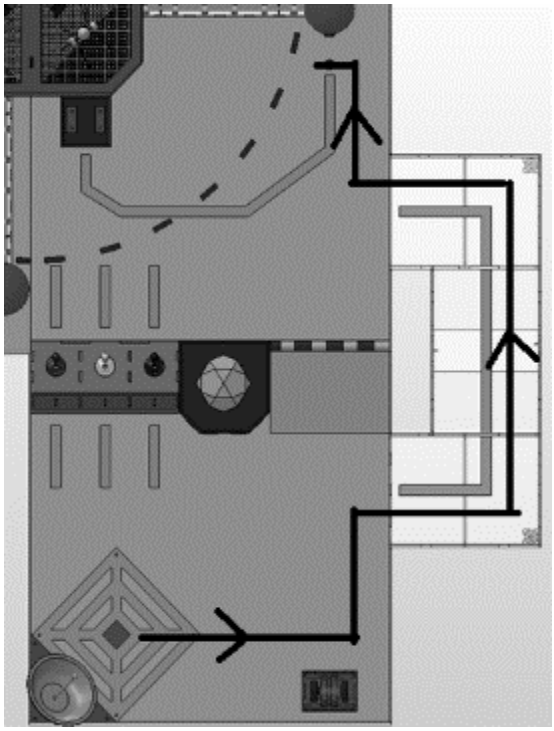


Figure A5: Proposed Course Strategy for First Design.

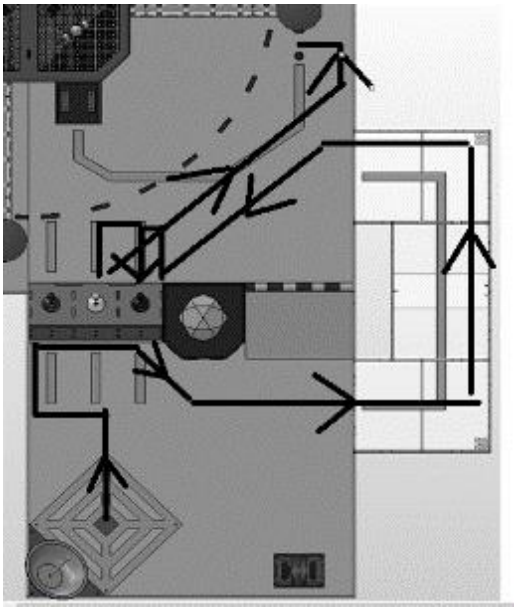


Figure A6: Proposed Course Strategy for Second Design.

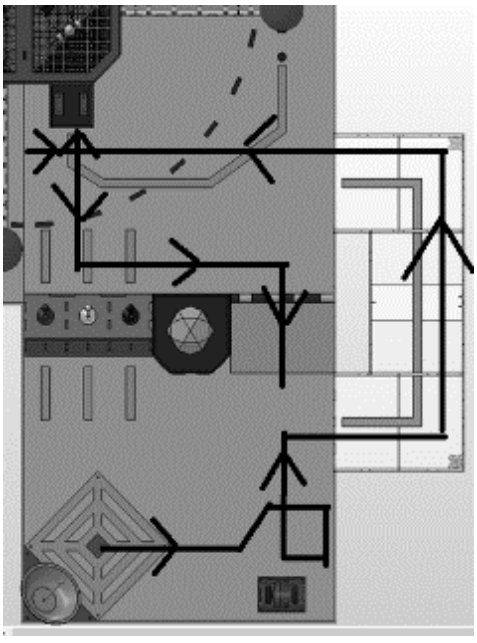


Figure A7: Proposed Course Strategy for Third Design.

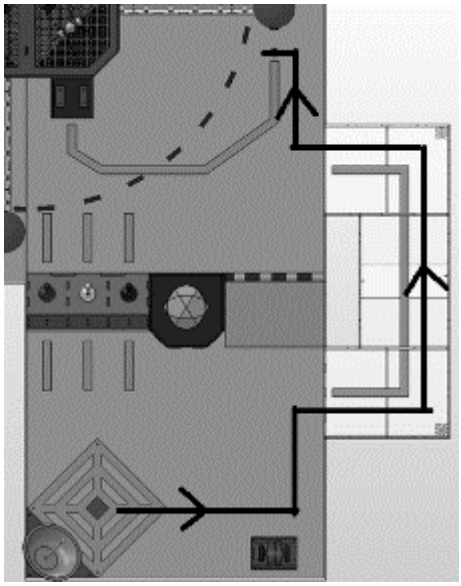


Figure A8: Performance Test 1 Path.

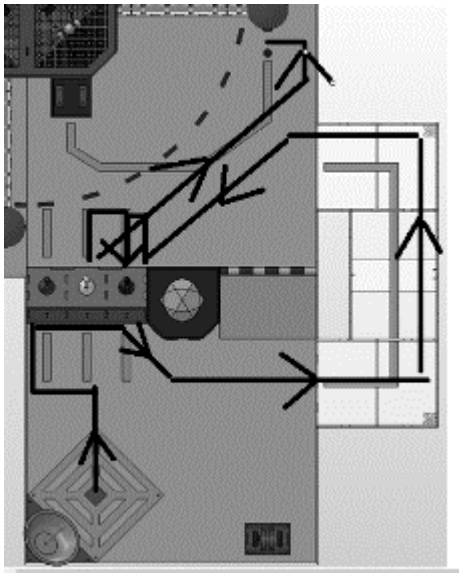


Figure A9: Performance Test 2 Path.

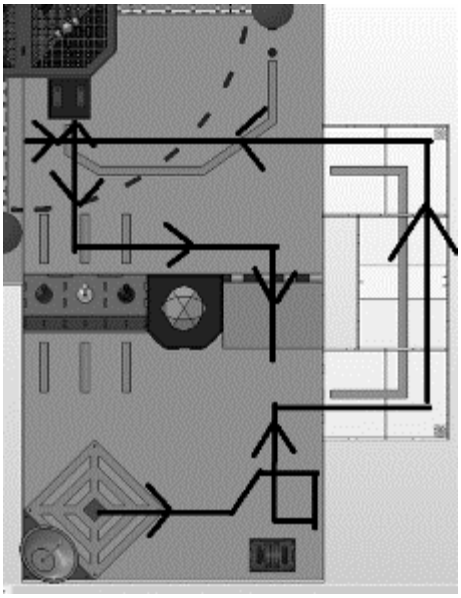


Figure A10: Performance Test 3 Path.

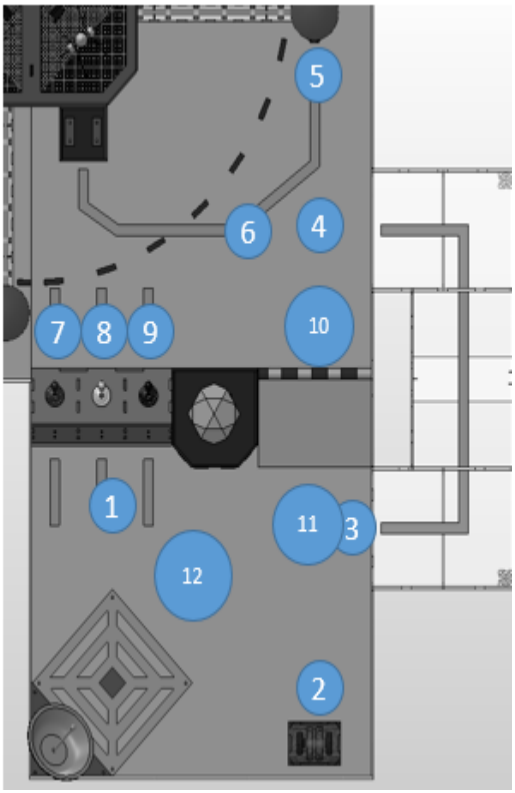


Figure A11: Key Locations for RPS.

Table A2: Locations and Corresponding RPS Values.

Location on Diagram	Description	X Value	Y Value
1	Off the Start Platform Towards Switches	Not Needed	24.0
2	Supplies Pick Up	Not Needed	10.8
3	Bottom of Access Ramp	Not Needed	24.0
4	Top of Course, Left of Access Ramp	31.3	Not Needed
5	Fuel Light	31.8	61.8
6	Set Up for Run Long Run to Supplies Release	Not Needed	50.0
7	Red Switch Top	1.5	43.5
8	White Switch Top	6.4	Not Needed
9	Blue Switch Top	11.5	Not Needed
10	Top Main Ramp	27.7	42.5
11	Bottom Main Ramp	Not Needed	22.0
12	Final Button Lineup	18.4	21.3

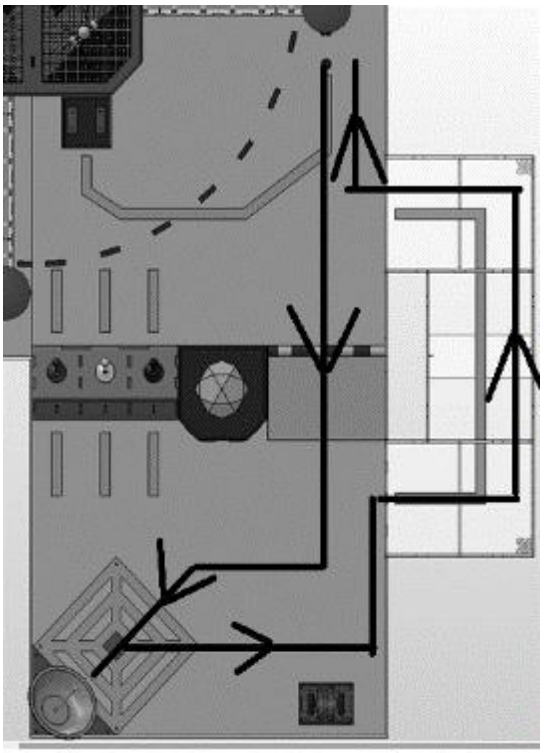


Figure A12: Performance Test 4 Path

APPENDIX B

Brainstorming Ideas and Design Concepts

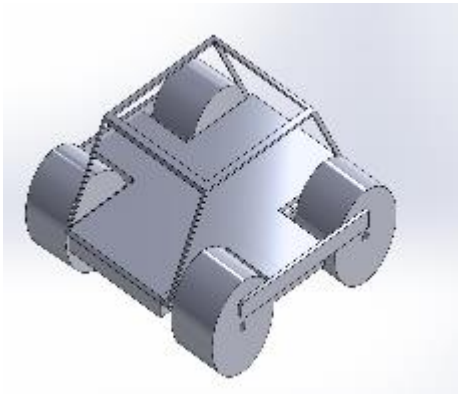


Figure B1: Train Chassis/Drivetrain Design Idea.

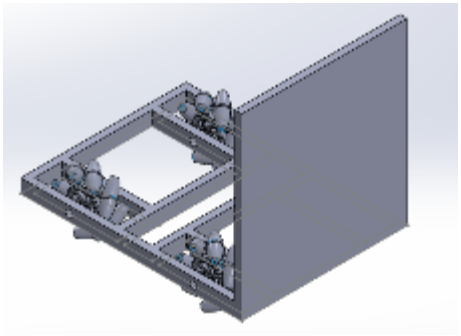


Figure B2: Mecanum Wheel Chassis Design.

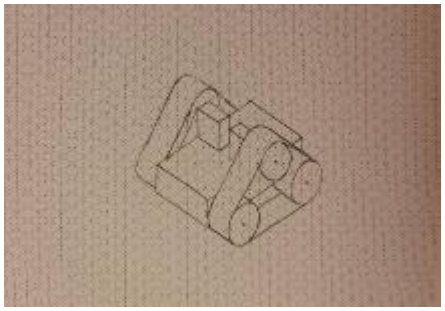


Figure B3: Tread Drivetrain Design.

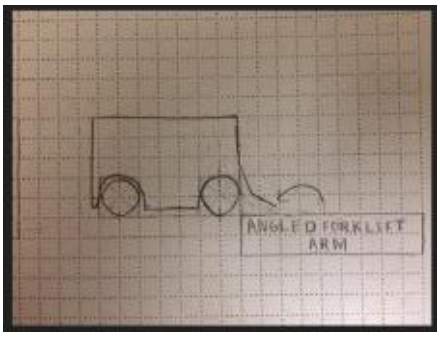


Figure B4: Forklift Design Idea.

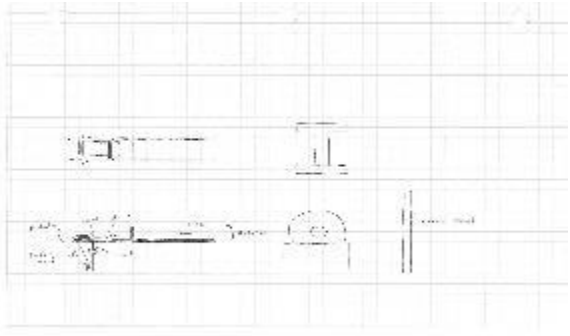


Figure B5: Supply Mechanism Proposal.

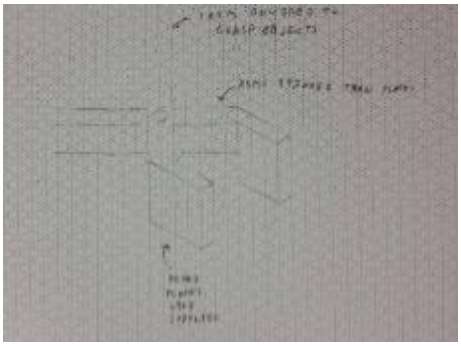


Figure B6: Vice Grip Mechanism.

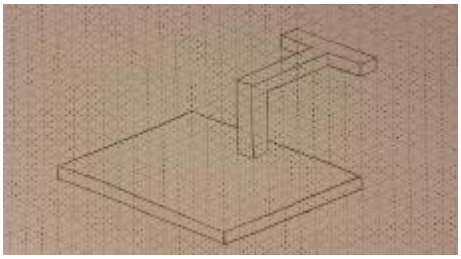


Figure B7: T-Shaped Hook Idea.

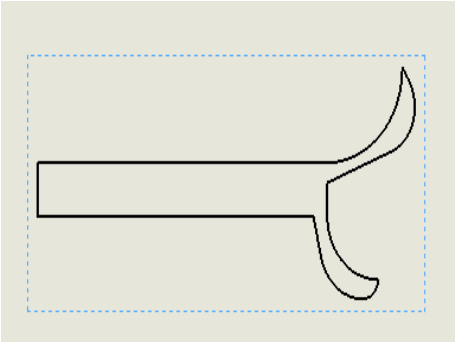


Figure B8: Double Sided Hook Idea.

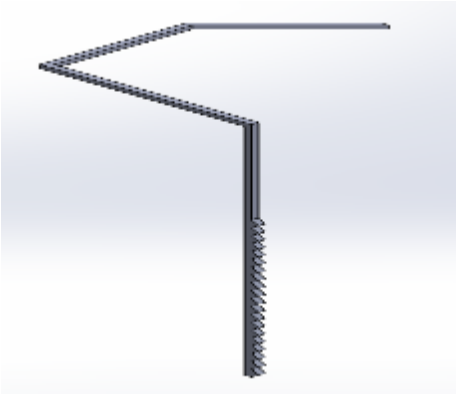


Figure B9: Vertically Moving Switch Arm Idea.

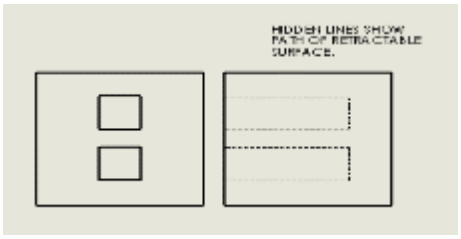


Figure B10: Retractable Slots Idea.

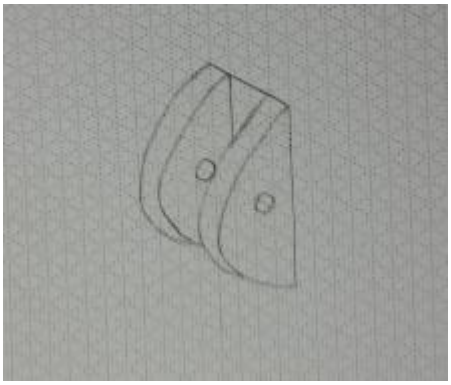


Figure B11: Seesaw Lever Idea.

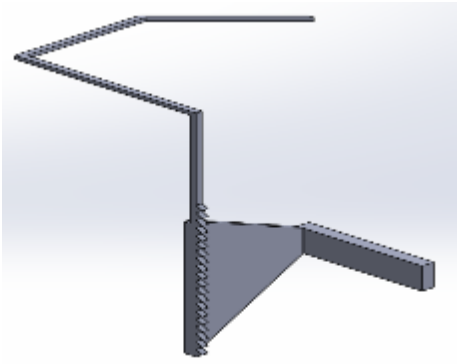


Figure B12: Vertically Moving Front Panel Idea.

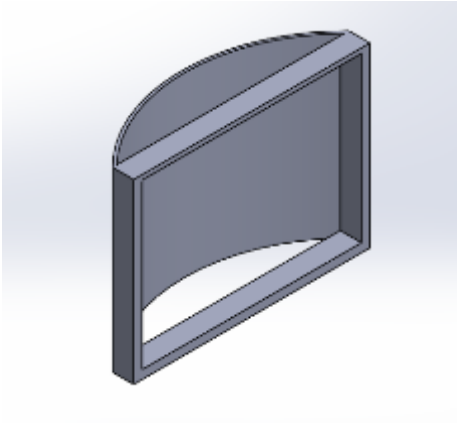


Figure B13: Potential Bumper Design.

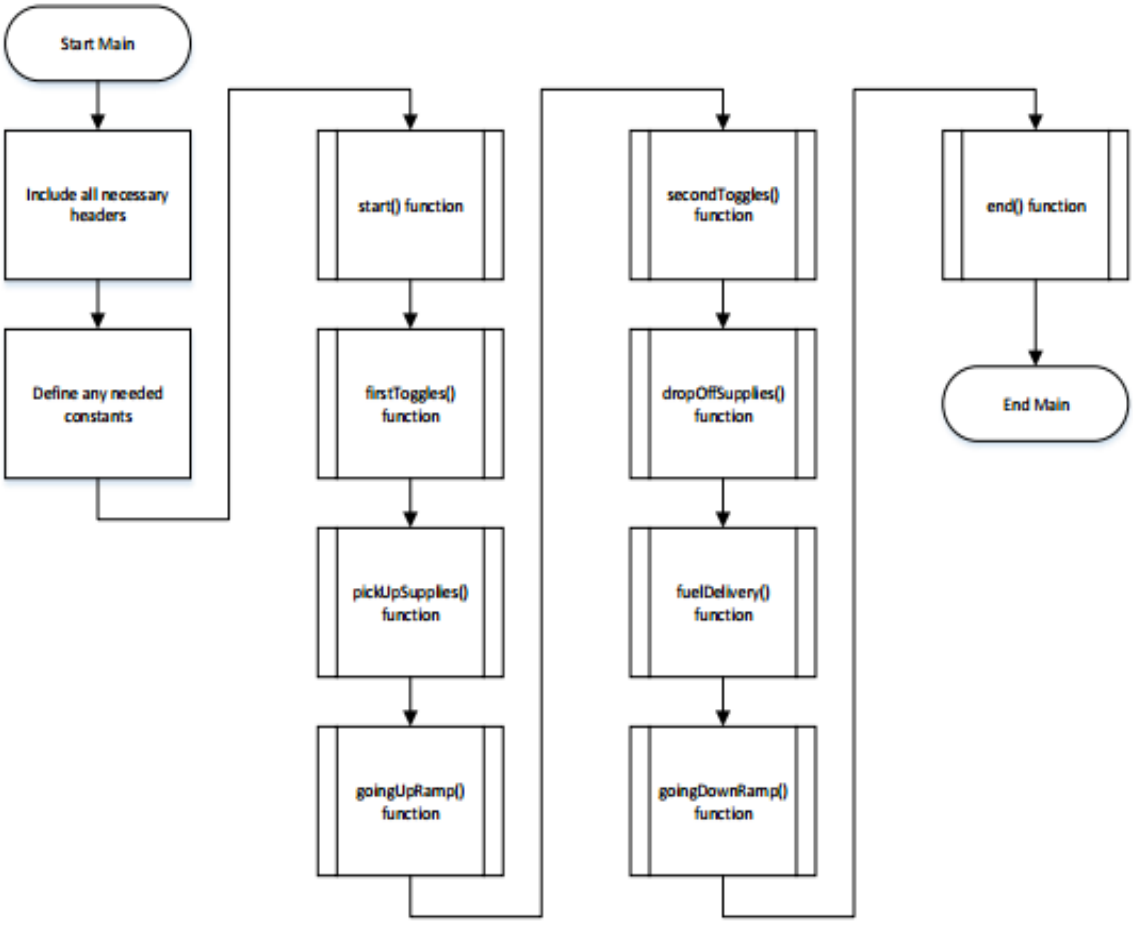


Figure B14: Preliminary Flowchart.

APPENDIX C

Decision Matrices

Table C1: Drivetrain Screening Matrix.

Drivetrain Design and Screening						
SUCCESS CRITERIA	FOUR WHEELS 2M (REF)	TREADS (2M)	MECANUM (4M)	2 DRIVE WHEELS (2M) w/ Skids	OMNI DIRECTIONAL (2M)	QUAD OMNI (3M)
BALANCE	0	0	0	0	0	0
MINIMAL BLOCKAGE	0	-	0	-	0	0
CENTER OF GRAVITY	0	0	0	-	0	0
DURABILITY	0	-	0	+	-	-
COST	0	0	-	+	+	+
STRUCTURE	0	+	-	0	-	-
TRACTION	0	+	+	-	+	+
WEIGHT	0	0	0	+	0	0
TURNING	0	0	+	+	+	+
Sum +s	0	2	2	4	3	3
Sum 0s	6	5	5	2	4	4
Sum -s	0	2	2	3	2	2
Net Score	0	0	0	1	1	1

Table C2: Chassis Screening Matrix.

Chassis Designs and Screening						
SUCCESS CRITERIA	REFERENCE	DESIGN A	DESIGN B	DESIGN C	DESIGN D	DESIGN E
BALANCE	0	0	0	0	-	0
MINIMAL BLOCKAGE	0	0	+	+	-	0
CENTER OF GRAVITY	0	0	0	0	-	0
DURABILITY	0	-	+	0	0	+
COST	0	0	-	0	+	+
STRUCTURE	0	-	+	-	-	-
WEIGHT	0	+	0	0	+	+
Sum +s	0	1	3	1	2	3
Sum 0s	6	4	3	5	1	3
Sum -s	0	2	1	1	4	1
Net Score	0	-1	2	0	-2	2

Chassis Design Descriptions				
Design	Shape	Material	Location of Wheels	Slots?
Reference	Rectangular	Acrylic (LC)	Outside	No
Design A	Rectangular	Aluminum	Outside	No
Design B	Rectangular	Acrylic (LC)	Enclosed	Yes
Design C	Circular	Acrylic (LC)	Outside	No
Design D	Triangular	Acrylic (LC)	Outside	No
Design E	Rectangular	PVC	Outside/Enclosed	Yes/No

Table C3: Switch Toggling Screening Matrix.

Switch Toggling Designs and Screening							
SUCCESS CRITERIA	REFERENCE	DESIGN A	DESIGN B	DESIGN C	DESIGN D	DESIGN E
BALANCE	0	0	0	-	0	0
MINIMAL BLOCKAGE	0	0	0	+	0	-
WEIGHT	0	0	0	-	0	0
MAINTANANCE	0	+	+	-	-	0
COST	0	+	+	-	-	0
HEIGHT	0	-	-	+	-	+
EXTENDABILITY	0	0	0	+	0	+
MOTORS NEEDED	0	+	+	-	0	0
Sum +s	0	3	3	3	0	2
Sum 0s	6	4	4	0	5	5
Sum -s	0	1	1	5	3	1
Net Score	0	2	2	-2	-3	1

Toggle Design Descriptions	
Design	Design Description
Reference	Arm that goes up and down on motor. Push only. Back only
Design A	T-shaped hook. Pull and push same side. Back only
Design B	Fixed stationary arm. Pull and push same side. Back only
Design C	Arm that goes up and down, forward and back, two motors.
Design D	Horizontal clamp that grabs switch. Drive forward or back to move. Back only
Design E	Arm that moves up a shaft, falls when at top. Push only.

Table C4: Fuel Delivery Matrix.

Fuel Delivery Designs and Screening						
SUCCESS CRITERIA	REFERENCE	DESIGN A	DESIGN B	DESIGN C	DESIGN D	DESIGN E
MINIMAL BLOCKAGE	0	0	0	0	0	0
DURABILITY	0	+	+	-	0	0
COST	0	+	+	-	0	0
STRUCTURE	0	+	+	0	0	0
ACCURACY	0	-	-	-	+	-
WEIGHT	0	0	0	0	0	0
MAINTENANCE	0	-	-	-	0	+
MOTORS	0	+	+	0	0	0
Sum +s	0	4	4	0	1	1
Sum 0s	6	2	1	4	7	6
Sum -s	0	2	2	4	0	1
Net Score	0	2	2	-4	1	0

Fuel Delivery Descriptions	
Design	Design Description
Reference	Flat Panel that moves up and down based on button to press
Design A	Checkered board where robot aligns based on button to press
Design B	Two different sides of the robot to press the different buttons
Design C	Flat panel, where part of panel is pulled into the robot
Design D	See sawing lever rotated by a motor
Design E	Rotating block 90 or 180 degrees of rotation on servo

Table C5: Final Button Matrix.

Final Button Pressing Design and Screening						
SUCCESS CRITERIA	REFERENCE	DESIGN A	DESIGN B	DESIGN C	DESIGN D	DESIGN E
CONSISTENCY	0	+	0	0	-	+
MINIMAL BLOCKAGE	0	+	0	0	0	+
DURABILITY	0	+	0	0	-	+
COST	0	+	0	0	-	++
STRUCTURE	0	+	0	0	0	+
WEIGHT	0	+	0	0	0	+
Sum +s	0	6	0	0	0	7
Sum 0s	6	0	6	6	3	0
Sum -s	0	0	0	0	3	0
Net Score	0	6	0	0	-3	7

Final Button Descriptions	
Design	Design Description
Reference	Extending Arm at fixed height that hits it
Design A	Run into it, use bumper/panel to protect
Design B	Slightly extended arm that rotates 90 degrees to slap button
Design C	Sliding arm on gear rail
Design D	Spring loaded steel ball, released based on position
Design E	Run into it, use foam as bumper

Table C6: Supplies Matrix.

Supply Control and Screening						
SUCCESS CRITERIA	REFERENCE	DESIGN A	DESIGN B	DESIGN C	DESIGN D	DESIGN E
BALANCE	0	0	0	0	-	-
MINIMAL BLOCKAGE	0	+	-	+	0	-
DROP POTENTIAL	0	+	+	+	-	-
MAINTENANCE	0	+	-	+	0	-
COST	0	0	0	0	0	-
STRUCTURE	0	+	-	0	0	0
ACCURACY	0	+	-	+	0	-
WEIGHT	0	+	+	0	0	-
Sum +s	0	6	2	4	0	0
Sum 0s	6	2	2	4	6	1
Sum -s	0	0	4	0	2	7
Net Score	0	6	-2	4	-2	-7

Supply Control Descriptions	
Design	Design Description
Reference	Forklift (1M) - to tilt
Design A	Magnets through Vertical Rod
Design B	Plastic Locking Clamp
Design C	Magnets on Horizontal PVC Pipe
Design D	Forklift (1M) - for vertical shifts
Design E	Vice grip from the sides

Table C7: Final Scoring Matrix.

Designs							
Success Criteria	Weight	Design A		Design B		Design C	
Balanced	5%	5	0.25	3	0.15	5	0.25
Minimal Blockage	15%	4	0.6	3	0.45	4	0.6
Center-of-Gravity Location	10%	5	0.5	4	0.4	5	0.5
Maintenance	15%	3	0.45	4	0.6	2	0.3
Durability	15%	3	0.45	4	0.6	2	0.3
Cost	20%	3	0.6	4	0.8	3	0.6
Ramp	5%	3	0.15	3	0.15	4	0.2
Mobility	15%	5	0.75	3	0.45	3	0.45
Total Score	100%	31	3.75	28	3.6	28	3.2

Layout of Each Design						
Design	Drive train	Fuel Delivery	Final Buttor	Supplies	Chassis	Toggle
A	Omni Wheels (3M)	Flat panel	Fixed flat panel	Vertical magnetic ro	PVC Sheets (surrounded)	T- shaped hook
B	2 Drive Wheels (skids	Checkered Board	Bumper	PVC Horizontal	PVC Sheets (surrounded)	Fixed Arm
C	Treads	Seesaw	Foam	Forklift	PVC Sheets (outside wheels)	Arm that moves up a shaft and falls down

APPENDIX D

Mockup Images

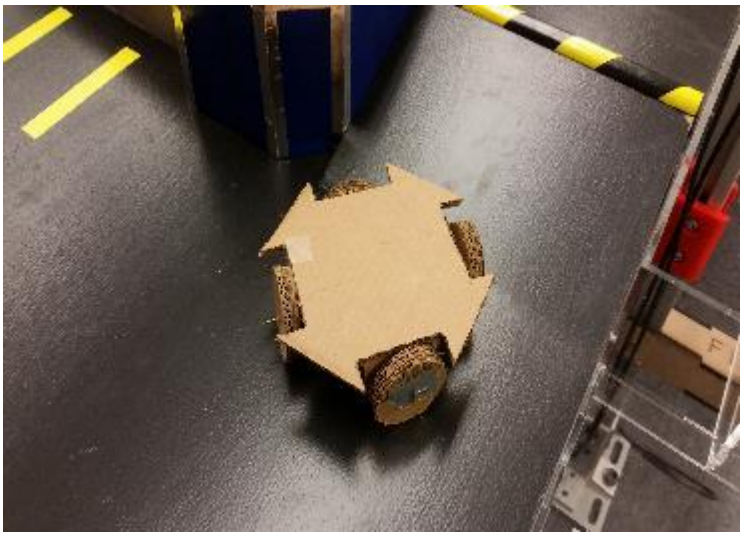


Figure D1: Physical Mockup Chassis.

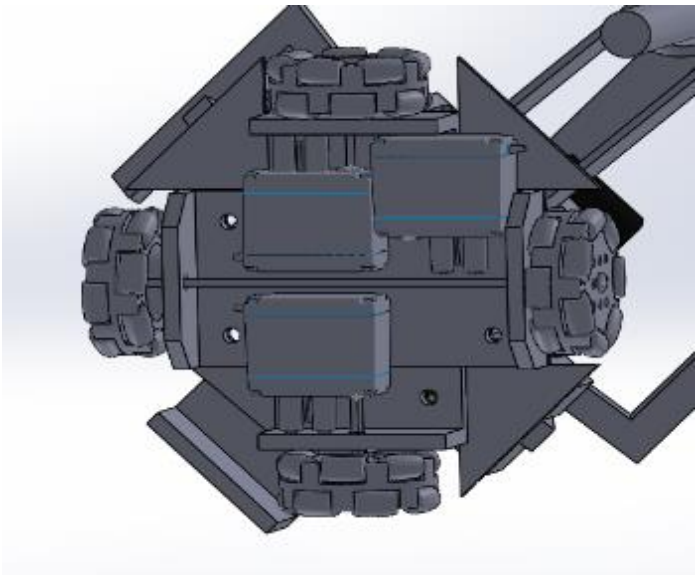
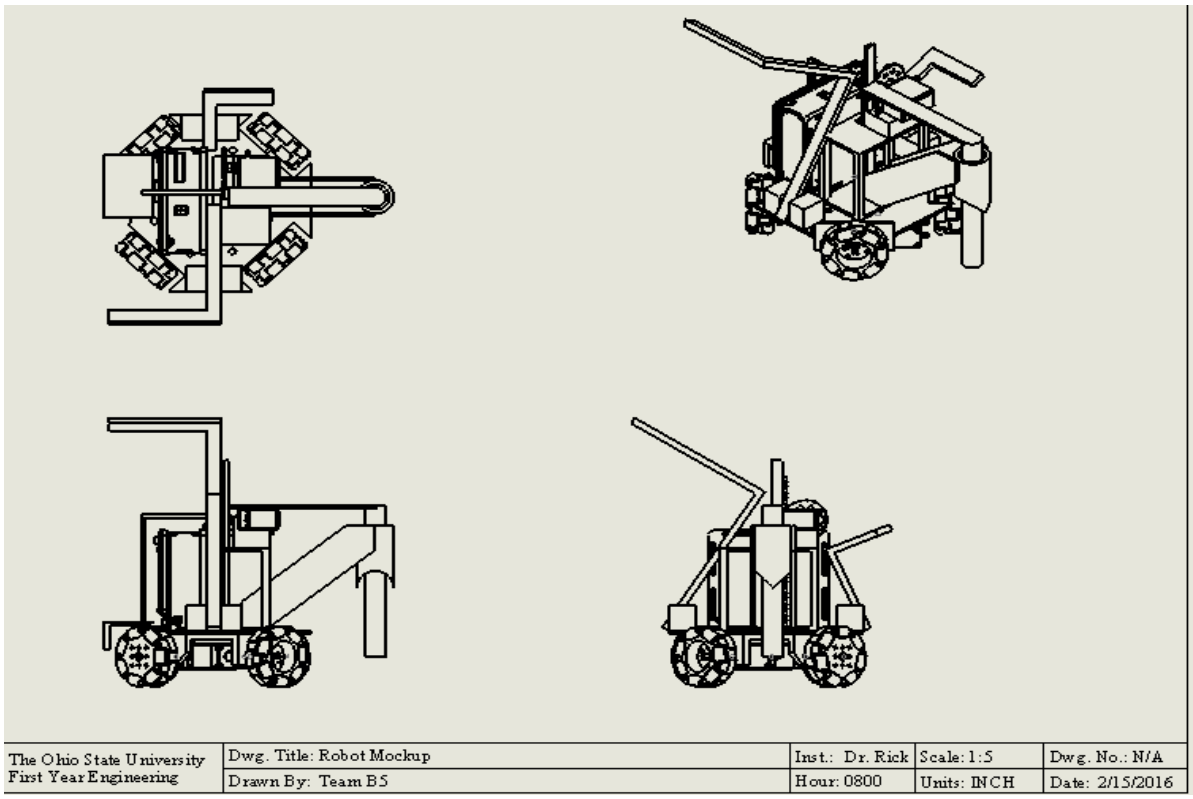


Figure D2: Underside of SolidWorks Model.



The Ohio State University First Year Engineering	Dwg. Title: Robot Mockup	Inst.: Dr. Rick	Scale: 1:5	Dwg. No.: N/A
	Drawn By: Team B5	Hour: 0800	Units: INCH	Date: 2/15/2016

Figure D3: Layout Drawing of Mockup.

APPENDIX E

Budgets

Table E1: Final Budget.

Item	Date of Purchase	Cost per Part	Cost per Inch	Cost per Inch^2	Quantity of Item (units)	Total Cost due to Part	Budget Remaining ($160 Start)
25 Inch Acrylic	2/18/2016	N/A	N/A	$0.06	38	$2.28	$157.72
Laser Path	2/18/2016	N/A	$0.07	N/A	75	$5.25	$152.47
Laser Cutting Fee	2/18/2016	$2.00	N/A	N/A	1	$2.00	$150.47
VEX 393 Motor	2/19/2016	$15.00	N/A	N/A	4	$60.00	$90.47
Motterm	2/19/2016	$0.05	N/A	N/A	4	$0.20	$90.27
12x12 PVC Sheet	2/19/2016	$3.00	N/A	N/A	1	$3.00	$87.27
6 Pack #6 Screws	2/19/2016	$0.20	N/A	N/A	2	$0.40	$86.87
Large Spacers	2/22/2016	$0.22	N/A	N/A	8	$1.76	$85.11
Small Spacers	2/22/2016	$0.22	N/A	N/A	4	$0.88	$84.23
VEX Motor Axle	2/22/2016	$0.50	N/A	N/A	4	$2.00	$82.23
Microswitch	2/22/2016	$1.20	N/A	N/A	3	$3.60	$78.63
Shaft Lock Collar	2/22/2016	$0.35	N/A	N/A	8	$2.80	$75.83
Angle Bracket	2/22/2016	$1.00	N/A	N/A	4	$4.00	$71.83
Red Filter	2/23/2016	$0.10	N/A	N/A	1	$0.10	$71.73
Microswitch	2/25/2016	$1.20	N/A	N/A	4	$4.80	$66.93
Double Bracket	2/26/2016	$0.74	N/A	N/A	2	$1.48	$65.45
Double Bracket	2/26/2016	$0.56	N/A	N/A	2	$1.12	$64.33
# 2 Screws	2/29/2016	$0.20	N/A	N/A	4	$0.80	$63.53
5.5 " Strip Erector	2/29/2016	$0.89	N/A	N/A	1	$0.89	$62.64
9.5" Strip Erector	2/29/2016	$1.65	N/A	N/A	1	$1.65	$60.99
Proteus Repair	3/2/2016	$2.00	N/A	N/A	1	$2.00	$58.99
9.5" Strip Erector	3/4/2016	$1.65	N/A	N/A	2	$3.30	$55.69
#6 Screws (3/8")	3/4/2016	$0.20	N/A	N/A	1	$0.20	$55.49
Triangle Flex Plate and Return	3/4/2016	$0.90	N/A	N/A	1	$0.90	$54.59
Trunnion	3/4/2016	$0.98	N/A	N/A	2	$1.96	$52.63
Angle Girder	3/6/2016	$1.58	N/A	N/A	2	$3.16	$50.14
Sheet of Paper	3/7/2016	$0.01	N/A	N/A	1	$0.01	$50.13
Rubber Band	3/7/2016	$0.01	N/A	N/A	2	$0.02	$50.11
Futaba Servo	3/7/2016	$10.00	N/A	N/A	1	$10.00	$40.11
Omni-Wheels	3/7/2016	$5.00	N/A	N/A	4	$20.00	$20.11
Magnet Square	3/7/2016	$1.35	N/A	N/A	2	$2.70	$17.41
Microswitch	3/31/2016	$1.20	N/A	N/A	1	$1.20	$16.21
Velcro	4/7/2016	N/A	$0.15	N/A	3	$0.45	$15.76

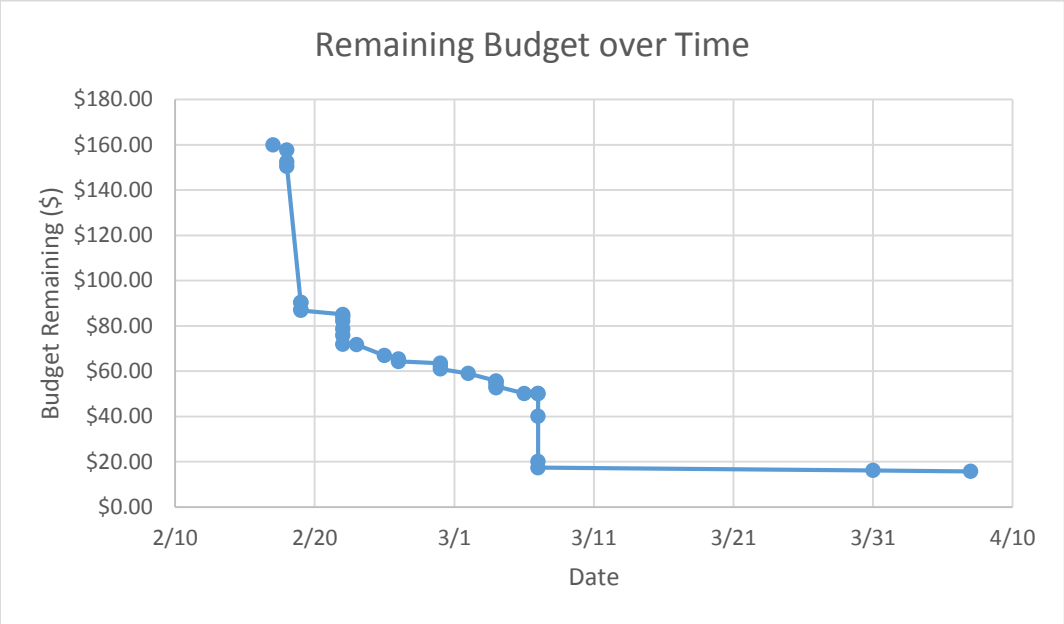


Figure E1: Line Chart Depiction of Budget over Time.

Table E2: Cost Breakdown of Budget by Part Type.

Type of Part	Total Cost by Type
Chassis/Drivetrain	$104.37
Electrical Systems	$11.90
Mechanisms	$27.52
Total	$143.79

APPENDIX F

Final Design and Time

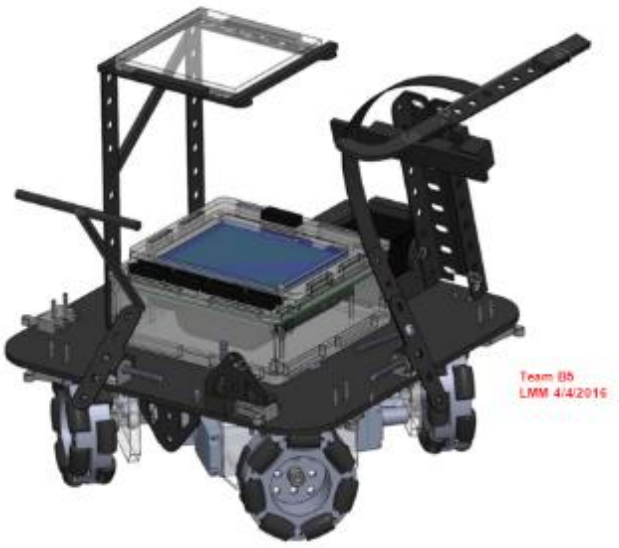


Figure F1: SolidWorks Model of Completed Robot.

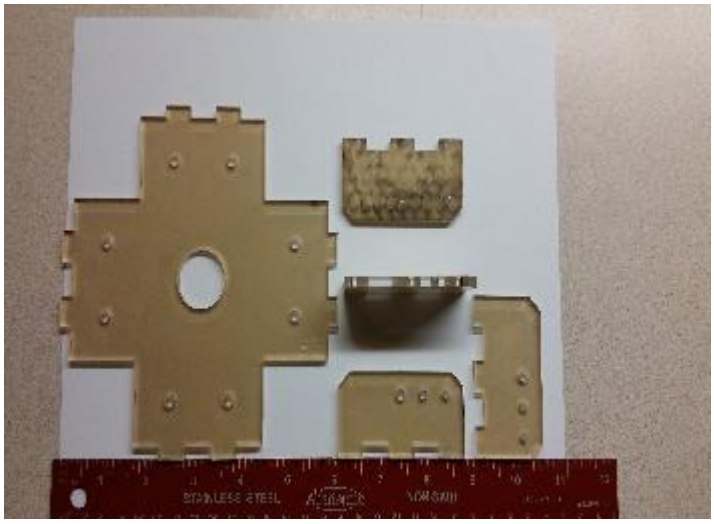


Figure F2: Disassembled Chassis.

Figure F3: Omnidirectional Wheel.

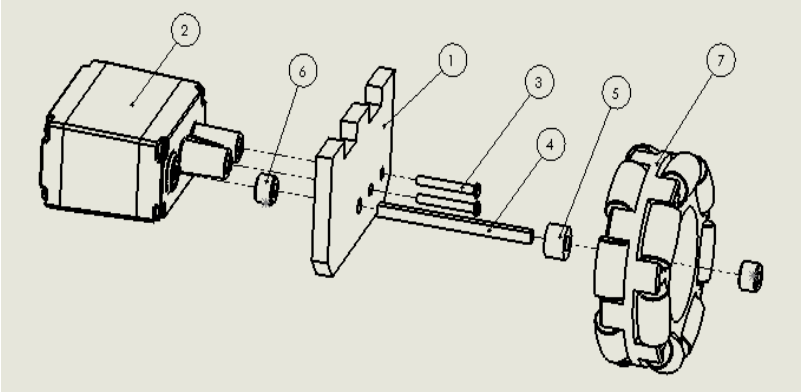


Figure F4: Motor and Wheel Connection.

Figure F5: Long Arm Final Design.

Figure F6: Short Arm Final Design.

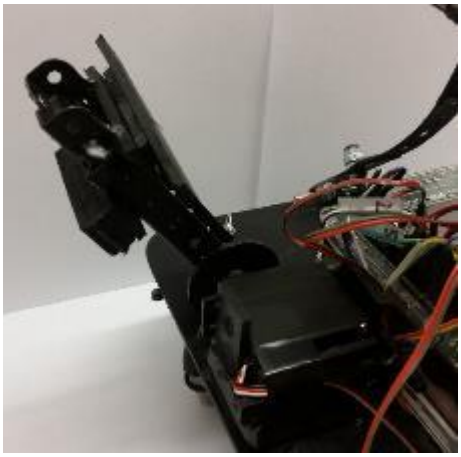


Figure F7: Supplies Arm Final Design.

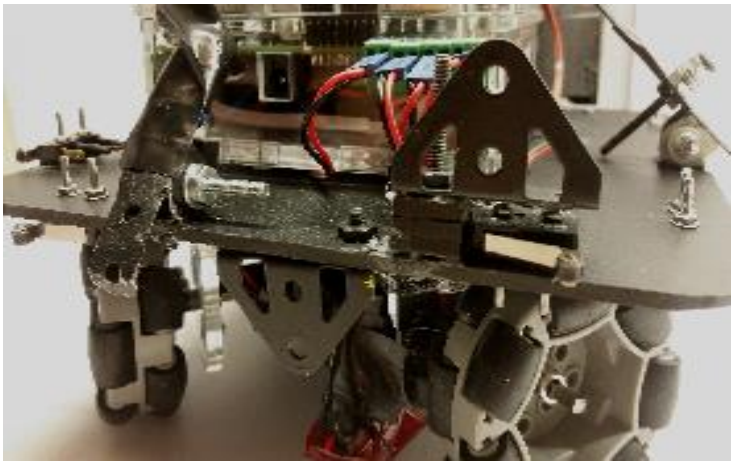


Figure F8: Trunion Checkerboard Final Design.

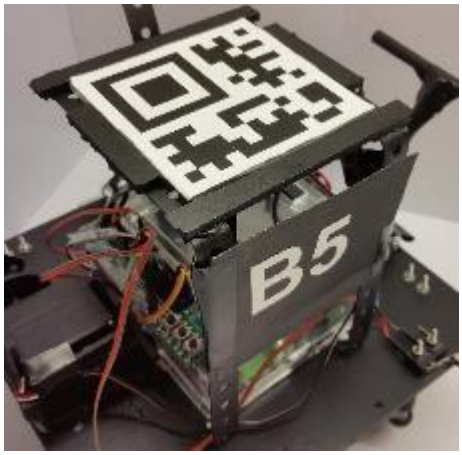


Figure F9: QR Code Mount Final Design.

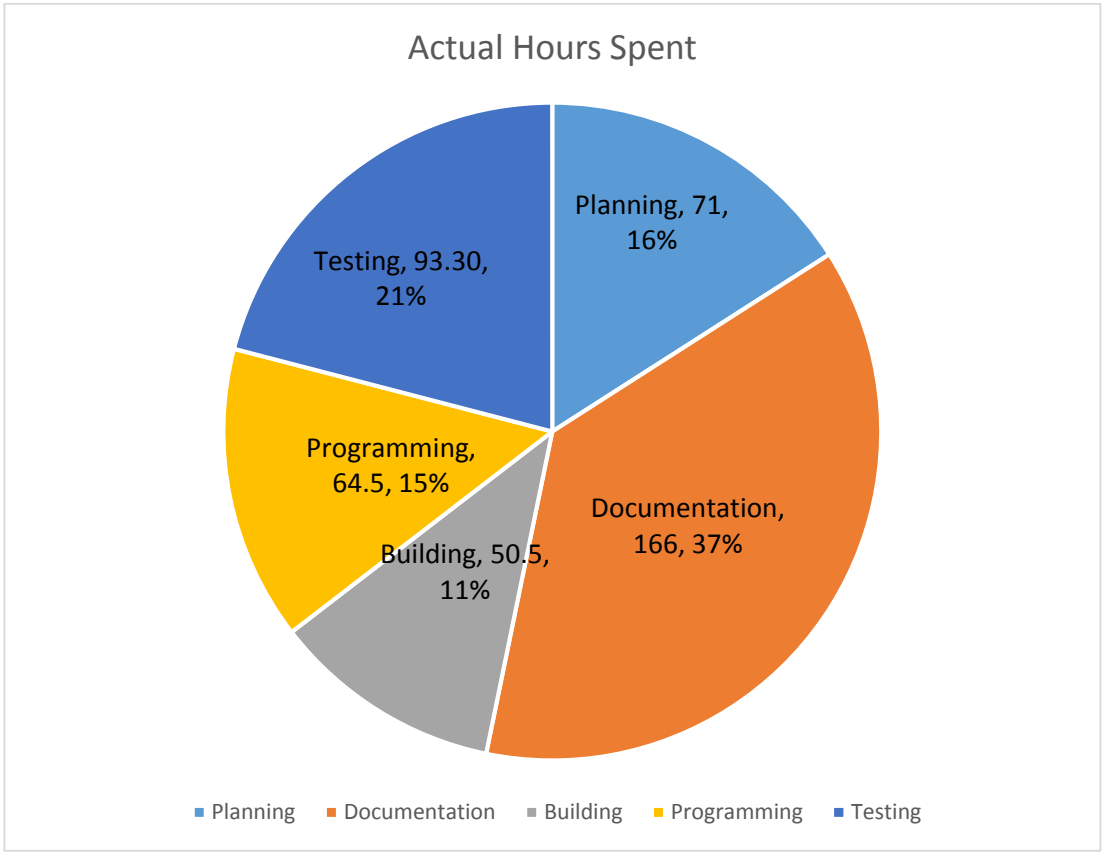


Figure F10: Pie Chart Breakdown of Hours Spent by Category.

Table F1: GPIO Information.

Port ID	Wire ID	In-Code Name (Object Name)	Type of Sensor	Purpose	Additional Notes
P0_0	CdS	cdscell	CdS Cell	Detects the start light/Fuel Light	White/Purple (Tape marked as C)
P0_1					
P0_2					
P0_3					
P0_4					
P0_5					
P0_6					
P0_7					
P1_0	Bump switch 1	microswitch1	Roller blade microswitch	Detect contact with wall	Brown/Red North Left (Tape marked as 1)
P1_1					
P1_2	Bump switch 2	microswitch2	Roller blade microswitch	Detect contact with wall	Black/White North Right (Tape marked as 2)
P1_3					
P1_4	Bump switch 3	microswitch3	Roller blade microswitch	Detect contact with wall	Brown/Red East Left (Tape marked as 3)
P1_5					
P1_6	Bump switch 4	microswitch4	Roller blade microswitch	Detect contact with wall	Brown/Red East Right (Tape marked as 4)
P1_7					
P2_0	Bump switch 5	microswitch5	Roller blade microswitch	Detect contact with wall	Green/Blue South Left (Tape marked as 5)
P2_1					
P2_2	Bump switch 6	microswitch6	Roller blade microswitch	Detect contact with wall	Yellow/Orange South Right (Tape marked as 6)
P2_3					
P2_4	Bump switch 7	microswitch7	Roller blade microswitch	Detect contact with wall	Green/Blue West Left (Tape marked as 7)
P2_5					
P2_6	Bump switch 8	microswitch8	Roller blade microswitch	Detect contact with wall	Yellow/Orange West Right (Tape marked as 8)
P2_7					
Bank 3 (P3_0 – P3_7)	FEH Button Board	button	3 buttons that allow for user input to Proteus	Send information to Proteus relating to continuing with program or logging values	Takes up all of bank 3, has 3 buttons

Table F2: Motor Electrical Information.

Port ID	Wire ID	Variable Name	Type of Motor	Purpose	Additional Notes
Motor0	Motor 1	motor1	Vex 393	Drive for Northwest wheel	Marked as 1 in Port 0
Motor1	Motor 2	motor2	Vex 393	Drive for Northeast wheel	Marked as 2 in Port 1
Motor2	Motor 3	motor3	Vex 393	Drive for Southeast wheel	Marked as 3 in Port 2
Motor3	Motor 4	motor4	Vex 393	Drive for Southwest wheel	Marked as 4 in Port 3
Servo0	Servo Arm	arm_servo	Futaba Servo	Lifts and holds dumbbell	Red, white, blue

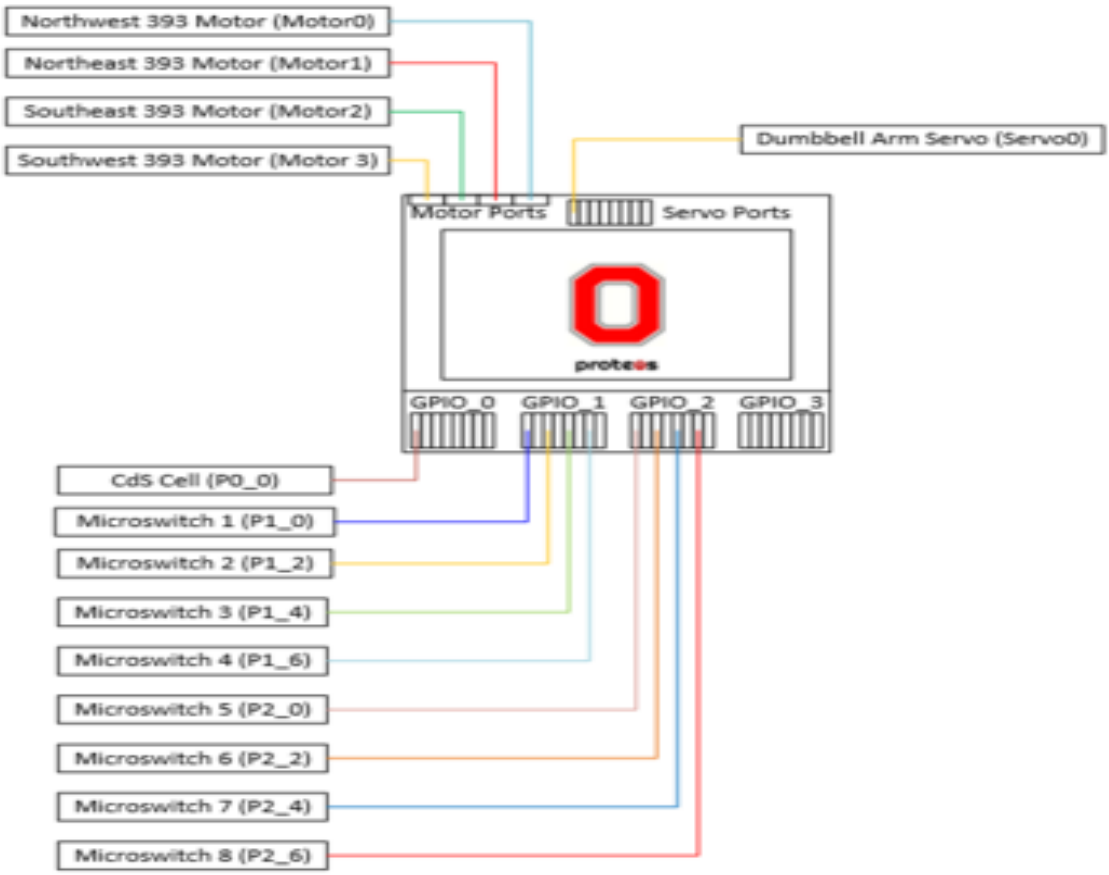


Figure F11: Electric Wiring Diagram.

Table F3: Condensed Final Design Schedule.

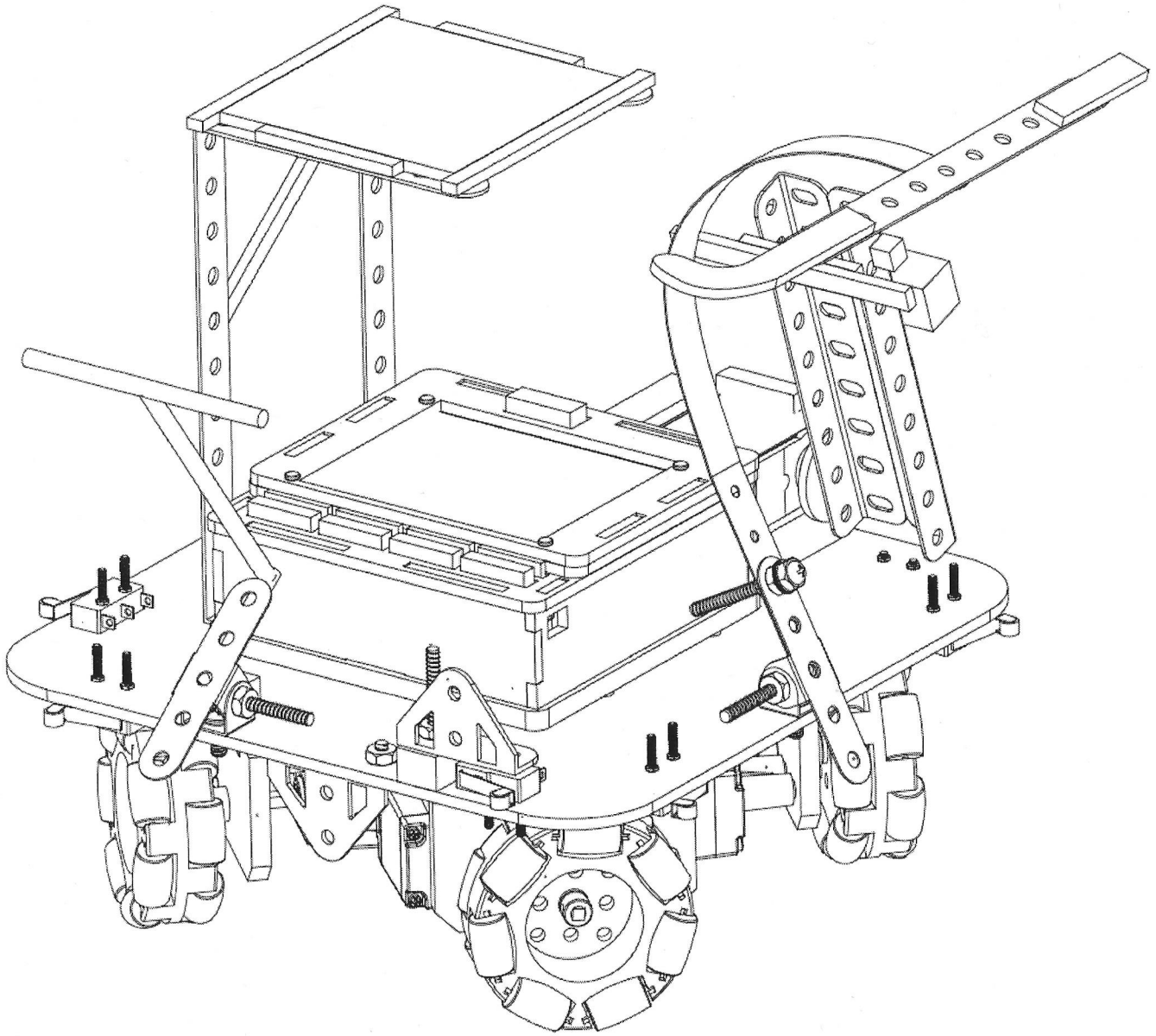
Task	Type of Task	Scheduled Dates		Actual Dates	
		Start	End	Start	End
R01: Individual Brainstorming	Planning	31-Jan	3-Feb	31-Jan	3-Feb
Team Formation	Planning	3-Feb	3-Feb	3-Feb	3-Feb
R02: Team Working Agreement	Planning	3-Feb	5-Feb	3-Feb	5-Feb
R03: Sketches, Decisions, and Strategy	Planning	4-Feb	9-Feb	4-Feb	9-Feb
R04: Design Schedule	Planning	9-Feb	11-Feb	9-Feb	11-Feb
R05: Mockup/Solid Model	Planning	10-Feb	14-Feb	10-Feb	14-Feb
R06: Drivetrain Analysis	Planning	12-Feb	16-Feb	12-Feb	15-Feb
EXP01: Sensors & Motors	Planning	12-Feb	18-Feb	12-Feb	18-Feb
R07: Pseudocode/Flowchart	Planning	15-Feb	21-Feb	15-Feb	22-Feb
Preliminary Budget	Planning	16-Feb	18-Feb	16-Feb	18-Feb
Build robot chassis	Building	15-Feb	21-Feb	17-Feb	24-Feb
EXP02: Line Following & Shaft Encoding	Programming	17-Feb	23-Feb	17-Feb	23-Feb
R08: Final Report Outline	Documentation	19-Feb	23-Feb	21-Feb	23-Feb
Performance Test 1	Programming	22-Feb	24-Feb	23-Feb	26-Feb
R09: Report Peer Review	Documentation	20-Feb	25-Feb	23-Feb	26-Feb
R10: Critical Design Review	Planning	12-Feb	3-Mar	2-Mar	2-Mar
Build robot attachments	Building	25-Feb	29-Feb	26-Feb	6-Mar
Performance Test 2	Programming	29-Feb	2-Mar	29-Feb	2-Mar
Technical Inspection 1	Building	2-Mar	6-Mar	7-Mar	7-Mar
EXP03: RPS & Data Logging	Programming	29-Feb	8-Mar	29-Feb	8-Mar
Build Robot Attachments	Building	1-Mar	8-Mar	4-Mar	8-Mar
Performance Test 3	Programming	4-Mar	9-Mar	8-Mar	9-Mar
R11: Final Report Draft 1	Documentation	7-Mar	22-Mar	10-Mar	22-Mar
Performance Test 4	Programming	10-Mar	23-Mar	11-Mar	23-Mar
R12: Electrical Documentation	Documentation	9-Mar	27-Mar	28-Feb	9-Mar
Individual Competition	Competition	1-Apr	1-Apr	1-Apr	1-Apr
R13: Final Report Draft 2	Documentation	23-Mar	3-Apr	31-Mar	3-Apr
R14: Isometric for Display	Documentation	1-Apr	5-Apr	31-Mar	N/A
Final Competition	Competition	9-Apr	9-Apr	9-Apr	9-Apr
Technical Inspection 2	Building	8-Apr	12-Apr	13-Apr	13-Apr
R15: Working Drawing Set	Documentation	25-Mar	17-Apr	13-Apr	17-Apr
R16: Oral Report	Documentation	23-Mar	19-Apr	14-Apr	19-Apr
R17: Final Report	Documentation	11-Apr	24-Apr	18-Apr	24-Apr
R18: Project Notebook	Documentation	3-Feb	24-Feb	3-Feb	25-Apr
Testing (from Testing Log)	Testing	20-Feb	8-Apr	22-Feb	9-Apr

Table F4: Outline of Different Functions Used in the Final Code.

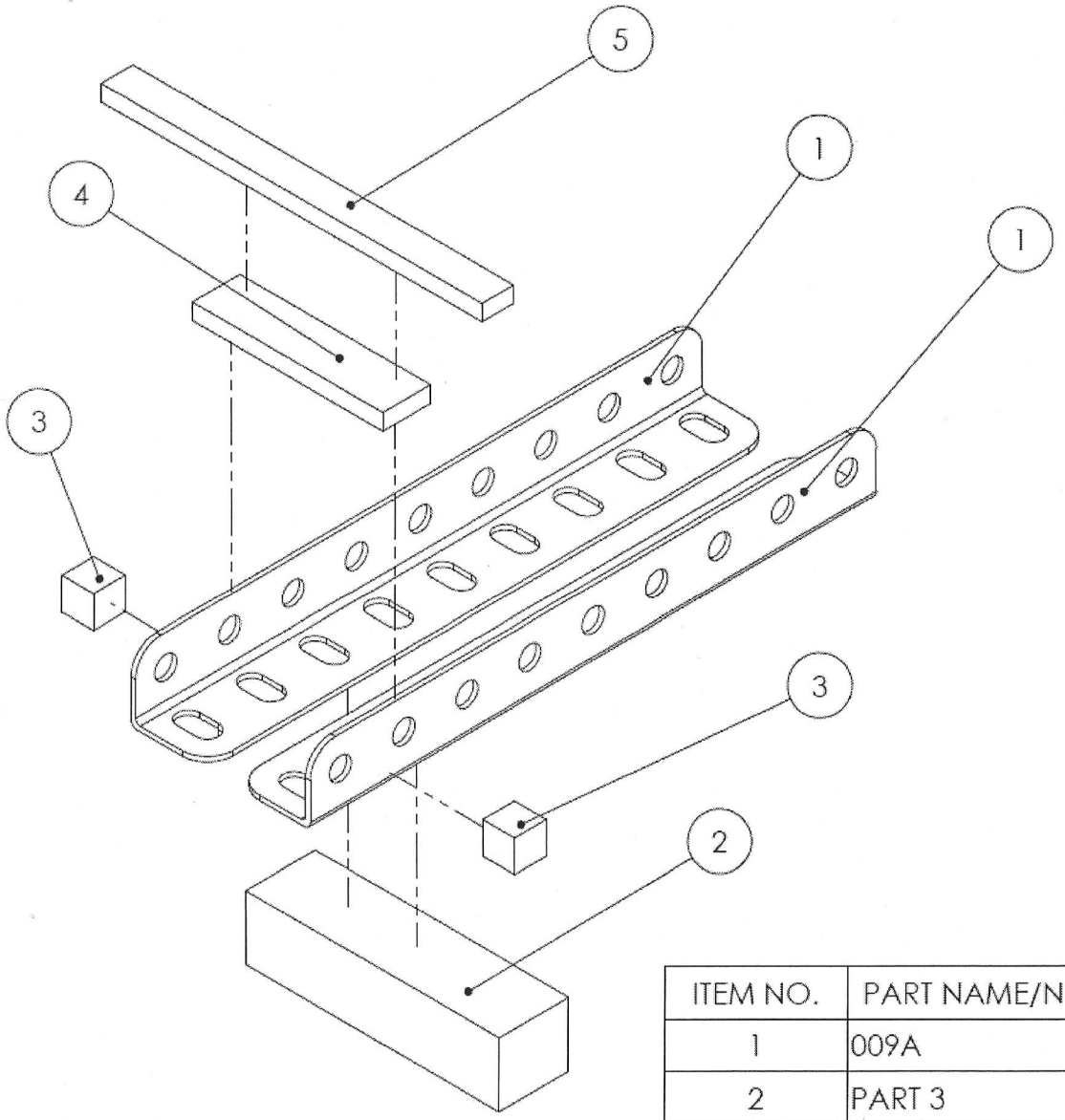
File	Function Name	Purpose
start	start	Run at the beginning of a run; Initializes RPS offsets and servo for the arm; Waits for the start light
toggles	togglesBottom	Controls the robot to hit the toggles from the bottom
	togglesTop	Controls the robot to hit the toggles from the top
supplies	pickupSupplies	Controls the robot to pick up the supplies
	dropOffSupplies	Controls the robot to drop off the supplies
ramp	tempRamp	Controls the robot to navigate up the temporary access ramp
	mainRamp	Controls the robot the drive down the main ramp
fuelbutton	fuelbutton	Controls the robot to press the correct fuel button depending on the fuel light
launchbutton	launchbutton	Controls the robot to press the final launch button ending the run
drive	driveUntilTime	Drives the robot for at a heading at a power for a duration
	driveWhileRotate	Drives the robot in a straight line while rotating the robot
	driveUntilBump	Drives the robot until microswitches are activated or deactivated; Used to stop at a wall or ride along a wall
	driveUntilBumpTimeout	The same as driveUntilBump with a timeout
	driveUntilCds	Drives the robot until a CdS value is read
	driveUntilRPSx	Drives the robot until the x-position of the robot is within a 0.5 inch range of the desired x-position
	driveUntilRPSy	Drives the robot until the y-position of the robot is within a 0.5 inch range of the desired y-position
	driveUntilRPSxRange	The same as driveUntilRPSx with a custom range
	driveUntilRPSyRange	The same as driveUntilRPSy with a custom range
	turnUntilTime	Rotates the robot at a power for a duration
	turnUntilRPS	Rotates until the robot is within a 3 degree range of the desired heading
supplyarm	initializeArm	Initializes the minimum and maximum of the servo
	lowerToPickupArm	Lowers the servo arm to pick up the supplies
	raiseToPickupArm	Raises the servo arm to pick up the supplies
	lowerToDepositArm	Lowers the servo arm to drop off the supplies
	raiseToDepositArm	Raises the servo arm after dropping off the supplie
worldstate	initializeLog	Closes any open logs, opens a new log, and writes a header to the opened log
	worldState	Prints all current sensor and motor information to the screen; Can write values to the opened log
	closeLog	Closes one open log

APPENDIX G

Drawing Set

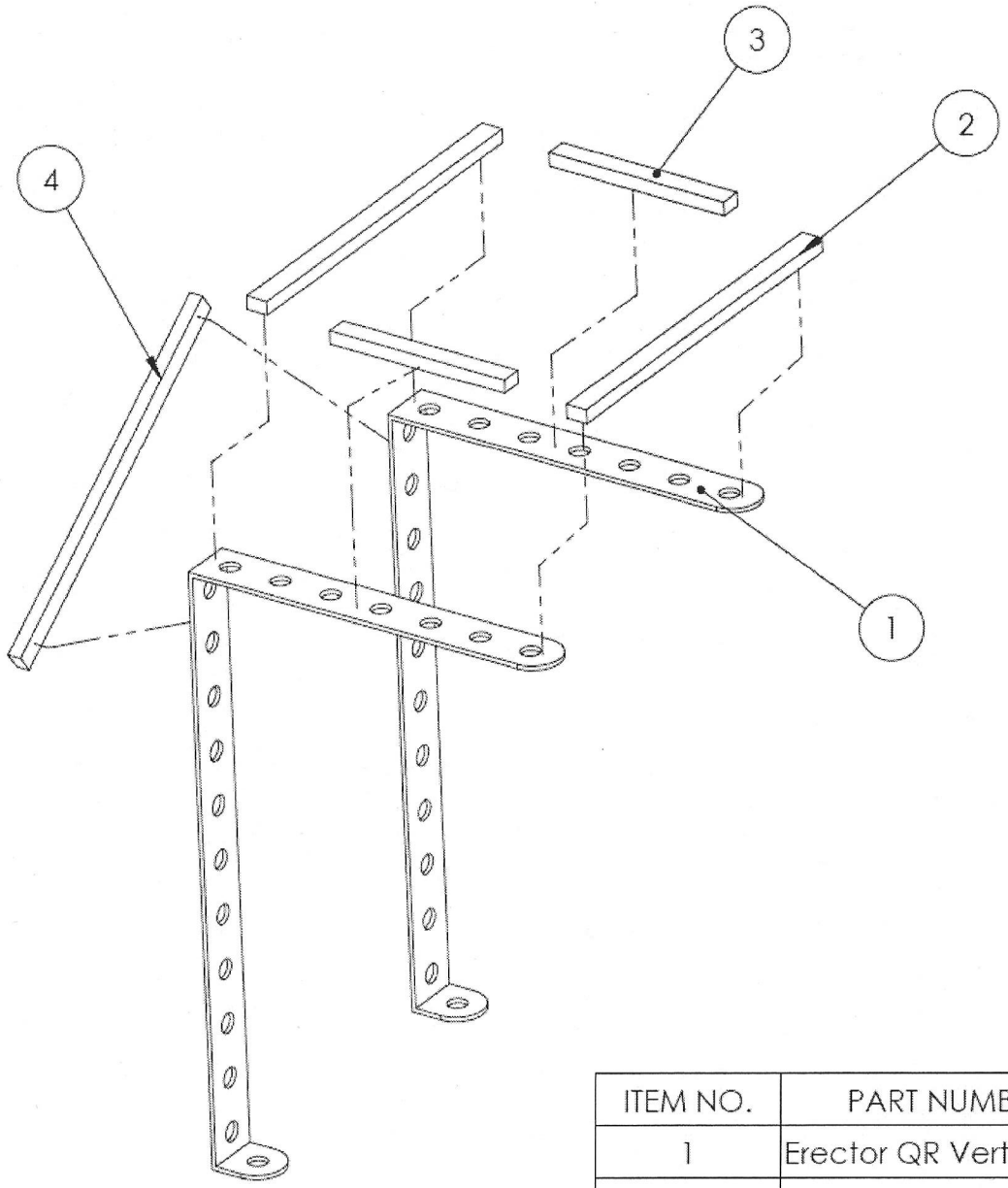


The Ohio State University First Year Engineering	Dwg. Title: FINAL DESIGN FULL ASSEMBLY	Inst.: RJF	Scale: 2:3	Dwg. No.: D-00
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/17/2016



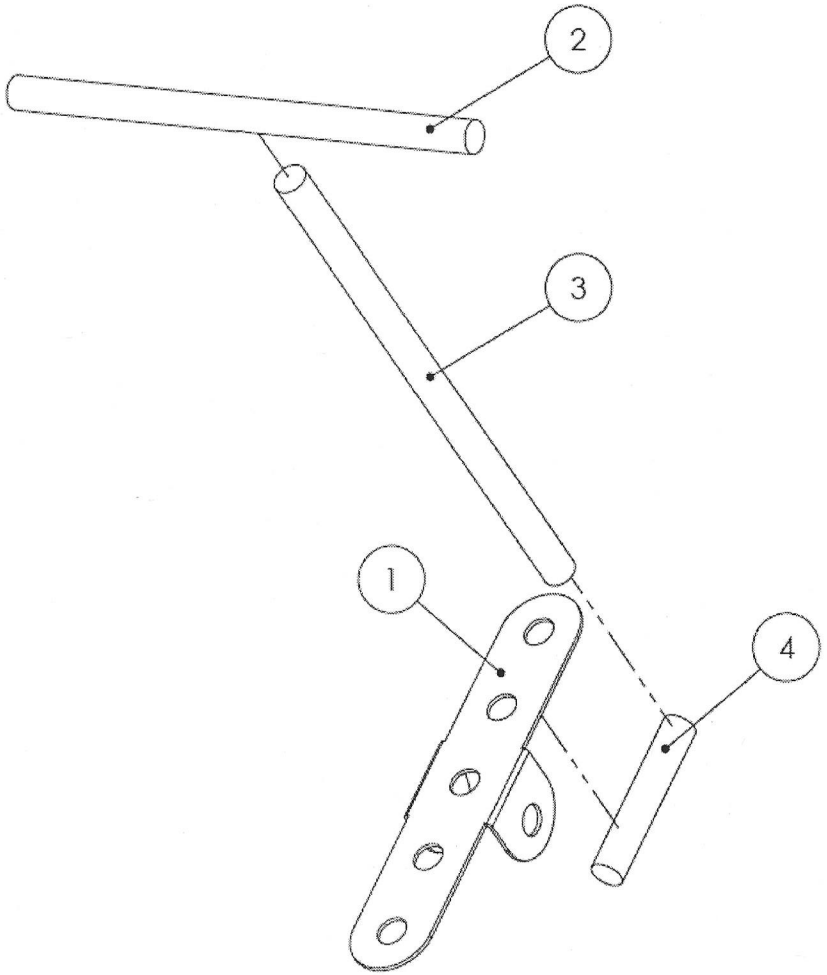
ITEM NO.	PART NAME/NUMBER	MATERIAL	QTY.
1	009A	ERECTOR	2
2	PART 3	PVC	1
3	MAG02	MAGNET	2
4	PART 6	PVC	1
5	SERVO ARM PART	PVC	1

The Ohio State University First Year Engineering	Dwg. Title: SERVO ARM EXPLODED ASSEMBLY	Inst.: RJF	Scale: 1:1	Dwg. No.: EX-00
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/6/2016



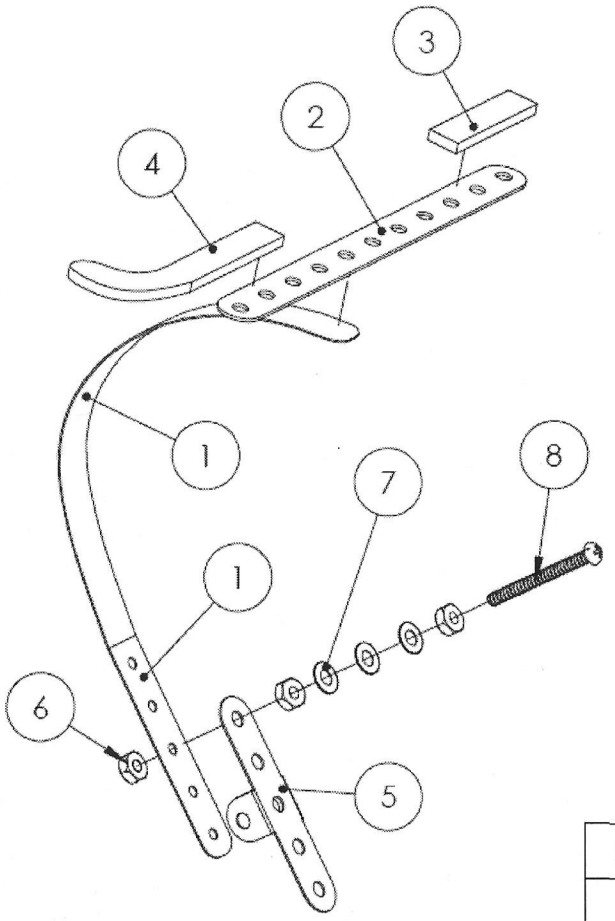
ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	Erector QR VertBrace	STEEL	2
2	pvc support qr	PVC	2
3	pvc support qr 2	PVC	2
4	Assem1^Qr code mount	PVC	1

The Ohio State University First Year Engineering	Dwg. Title: QR CODE MOUNT EXPLODED	Inst.: RJF	Scale: 2:3	Dwg. No.: EX-01
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016



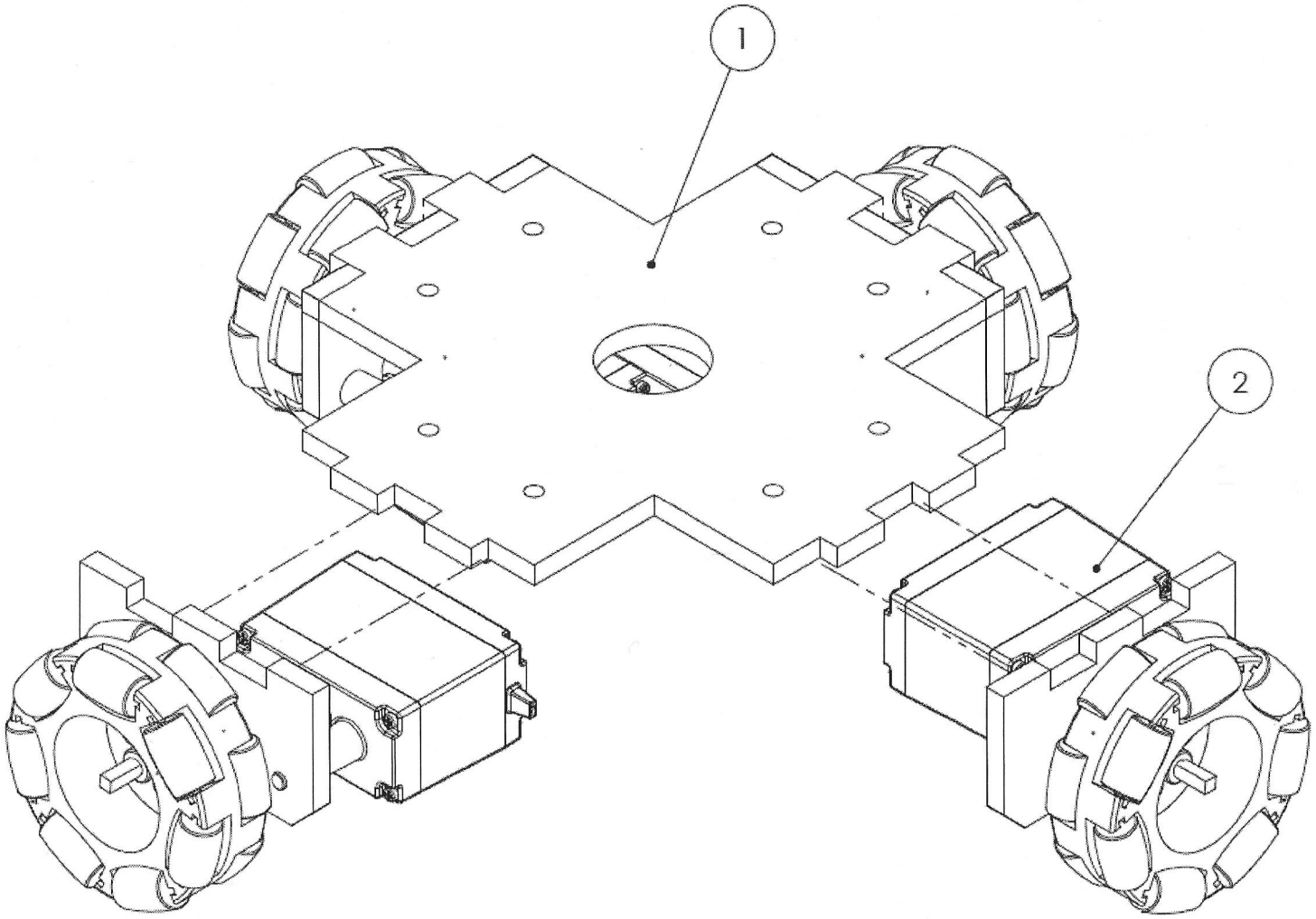
ITEM NO.	PART NAME/NUMBER	MATERIAL	QTY.
1	011B	ERECTOR	1
2	SmallArmPart	PAPER	1
3	SmallArmPart2	PAPER	1
4	SmallArmPart3	PAPER	1

The Ohio State University First Year Engineering	Dwg. Title: SHORT ARM EXPLODED VIEW	Inst.: RJF	Scale: 1:1	Dwg. No.: EX-02
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/7/2016



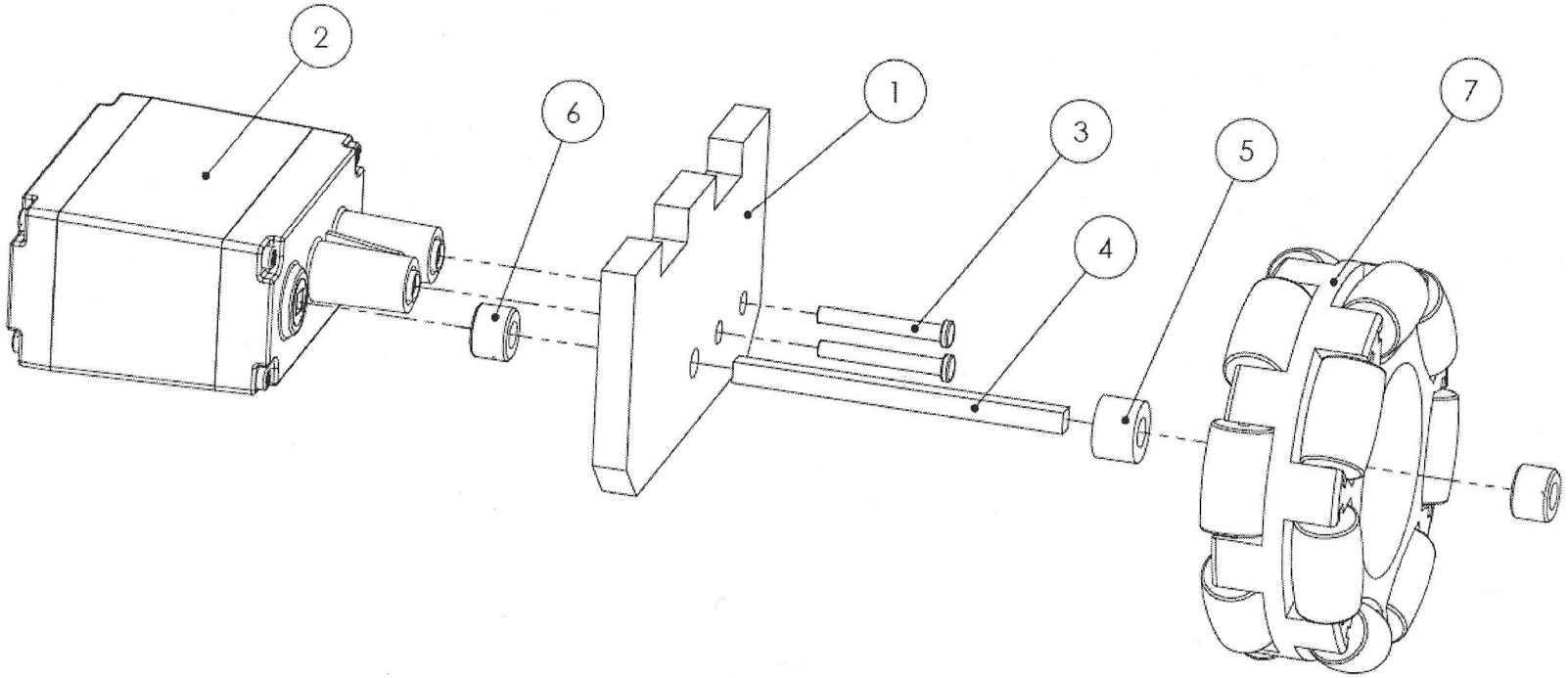
ITEM NO.	PART NAME/NUMBER	MATERIAL	QTY.
1	LongArmFinalRobot	ERECTOR	1
2	2	ERECTOR	1
3	Part6	PVC	1
4	Part7	PVC	1
5	011B	ERECTOR	1
6	LongScrewNut	STEEL	3
7	BigWasher	STEEL	3
8	SCREW09	STEEL	1

The Ohio State University First Year Engineering	Dwg. Title: LONG ARM EXPLODED VIEW	Inst.: RJF	Scale: 1:2	Dwg. No.: EX-03
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/6/2016



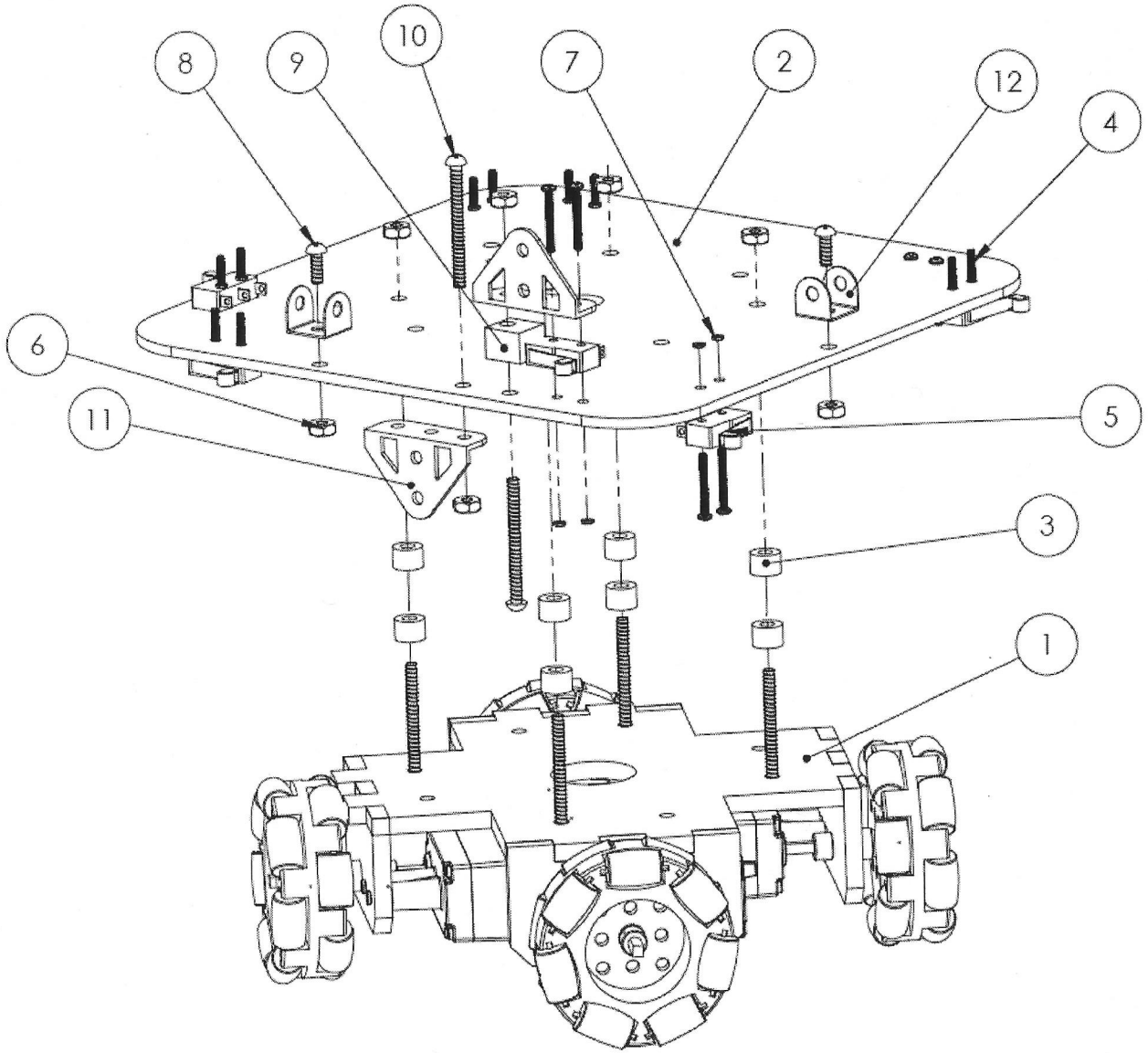
ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	motor_mount_base	ACRYLIC	1
2	omnimotoracrylicEXA	MISCELLANEOUS	4

The Ohio State University First Year Engineering	Dwg. Title: ACRYLIC/DRIVETRAIN EXPLODED	Inst.: RJF	Scale: 2:3	Dwg. No.: EX-04
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016

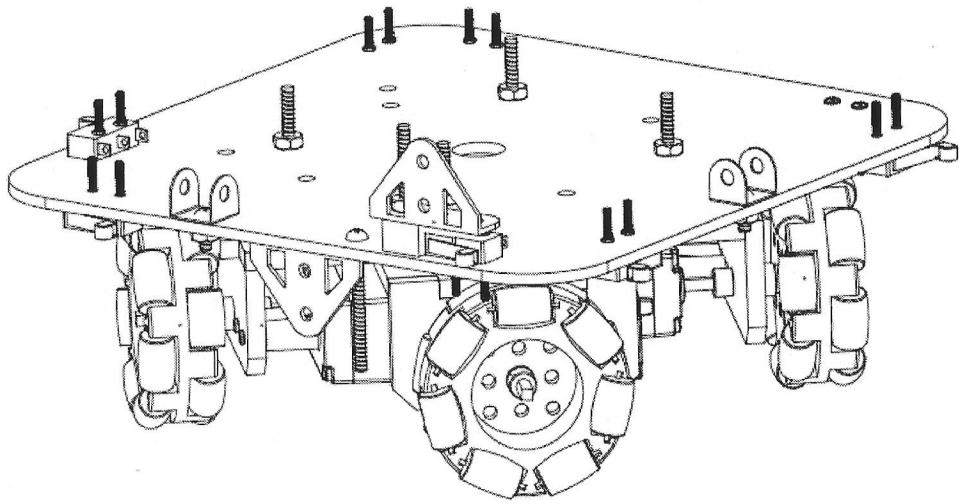


ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	motor_mount_sides	ACRYLIC (LASER CUT)	1
2	VEX_Motor	N/A	1
3	Long Screw	STEEL	2
4	VEXAX09	STEEL	1
5	038A	PLASTIC	1
6	ShaftLockCollar	STEEL	2
7	Omni Assembly	N/A	1

The Ohio State University First Year Engineering	Dwg. Title: DRIVETRAIN EXPLODED	Inst.: RJF	Scale: 1:1	Dwg. No.: EX-05
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016

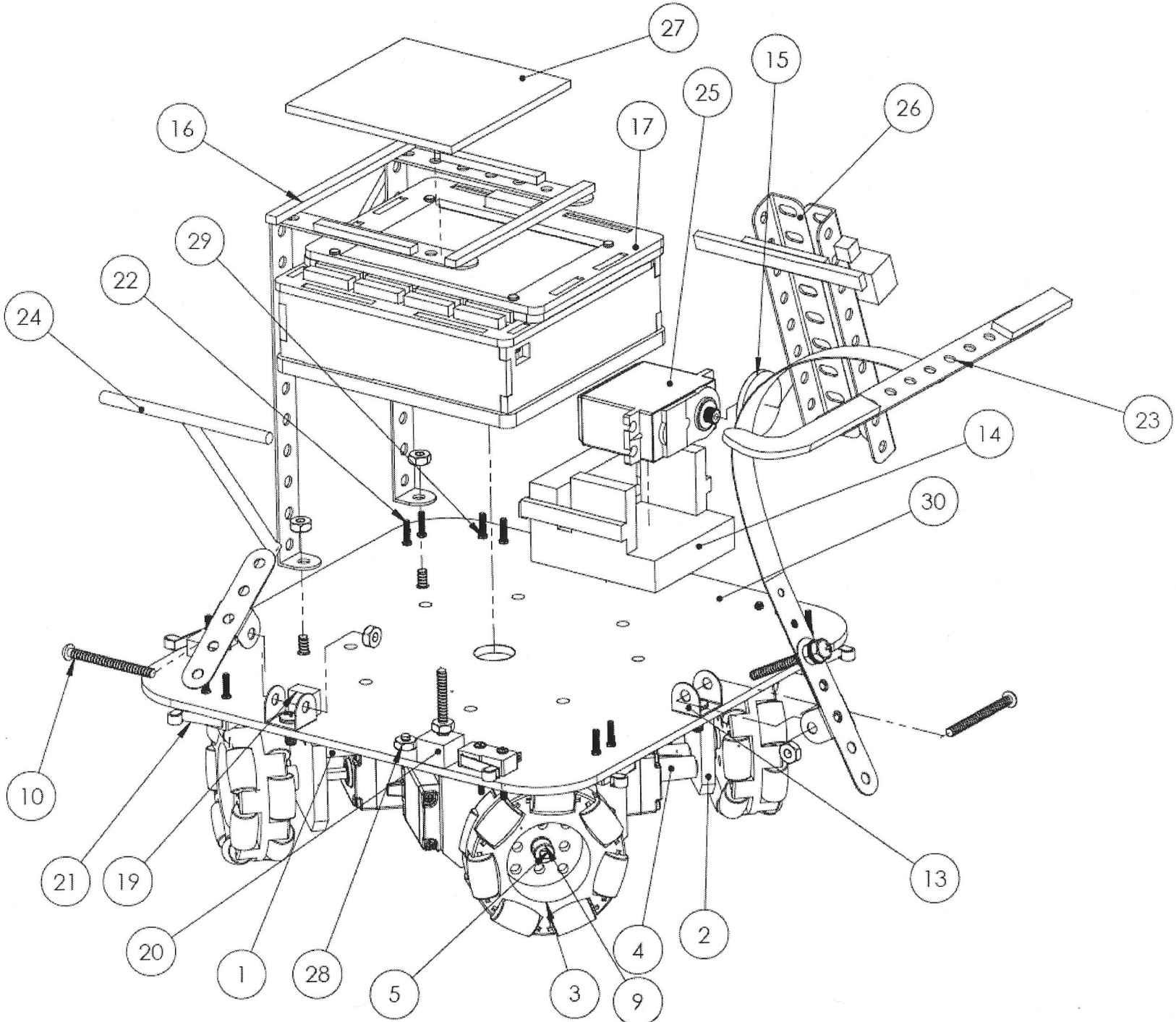


The Ohio State University First Year Engineering	Dwg. Title: PVC COMPONENTS EXPLOSION	Inst.: RJF	Scale: 1:2	Dwg. No.: EX-06
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016



ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	ACRYLIC&DRIVETRAIN		1
2	PVCsheet1	PVC	1
3	038B	PLASTIC	8
4	SCREW11	STEEL	16
5	SENSWRM		8
6	LongScrewNut	STEEL	8
7	Screw11Nut	STEEL	16
8	screw3.8	STEEL	2
9	redbuttonelevationblock	PVC	1
10	SCREW09	STEEL	6
11	Trunion	STEEL	2
12	11	STEEL	2

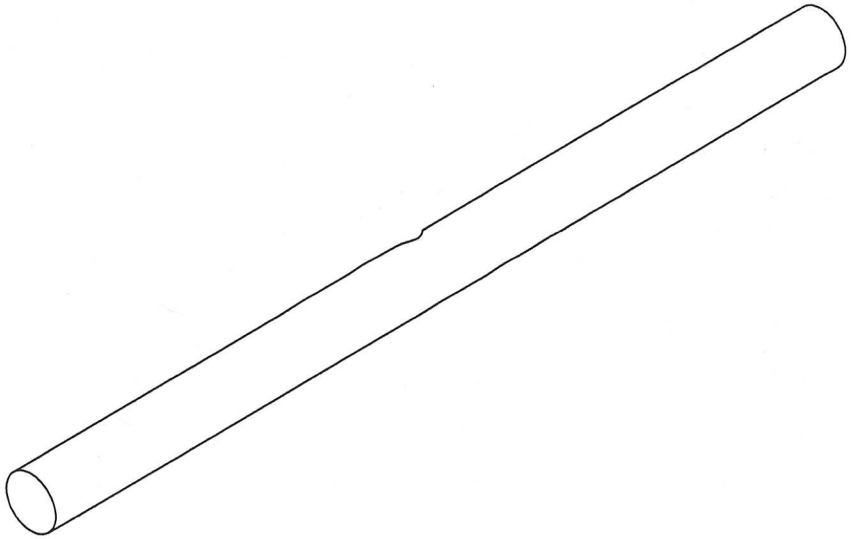
The Ohio State University First Year Engineering	Dwg. Title: PVC EXPLOSION B.O.M. (EX-06)	Inst.: RJF	Scale: 1:2	Dwg. No.: EX-06
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016

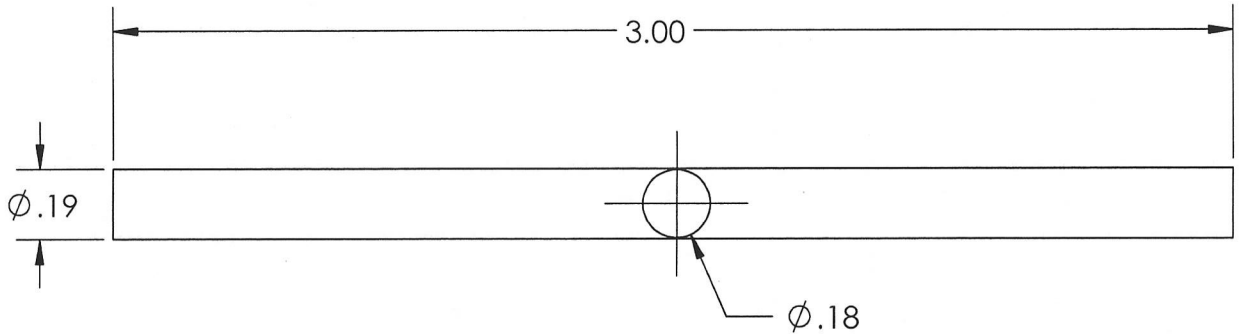


The Ohio State University First Year Engineering	Dwg. Title: FULL ASSEMBLY WITH MECHANISMS EXPLODED	Inst.: RJF	Scale: 1:2	Dwg. No.: EX-07
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016

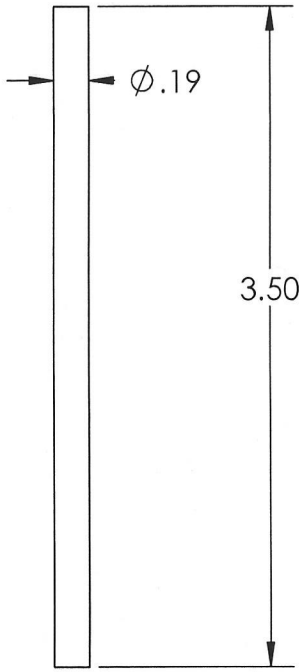
ITEM NO.	PART NUMBER	MATERIAL	QTY.
1	motor_mount_base	ACRYLIC	1
2	motor_mount_sides	ACRYLIC	4
3	Omni Assembly		4
4	VEX_Motor		4
5	Short Axle	STEEL	4
6	038B	PLASTIC	4
7	SCREW11	STEEL	8
8	Part1^FinalDesignFEHRobot2016	PVC	1
9	WHL00	STEEL	8
10	SCREW09	STEEL	8
11	038A	PLASTIC	8
12	SCREW07	STEEL	3
13	11	STEEL	2
14	Servomount	PVC	1
15	servomotorcirclepivot	PLASTIC	1
16	Qr code mount	MISCELLANEOUS	1
17	CONTROL01		1
18	SCREW02	STEEL	2
19	short arm catch	PVC	1
20	redbuttonelevationblock	PVC	1
21	SENSWRM		8
22	SCREW11 (1)	STEEL	16
23	LongArmFinal	MISCELLANEOUS	1
24	ShortArmFinal	MISCELLANEOUS	1
25	Futaba		1
26	PVCsheet1	PVC	1
27	nut		4
28	Trunion		2
29	ServoARM (1)		1

The Ohio State University First Year Engineering	Dwg. Title: FINAL ASSEMBLY B.O.M.	Inst.: RJF	Scale: 1:1	Dwg. No.: EX-07
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/15/2016

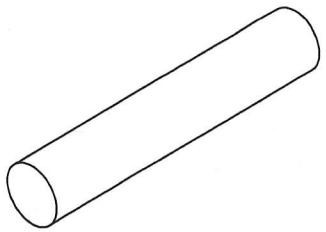


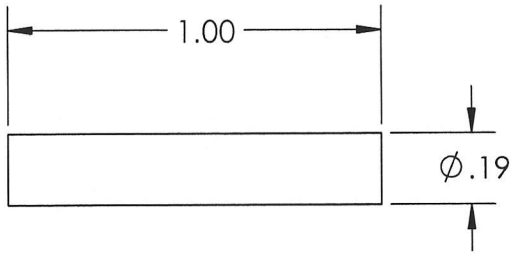


The Ohio State University First Year Engineering	Dwg. Title: SMALL ARM PART	Inst.: RJF	Scale: 2:1	Dwg. No.: DD-00
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

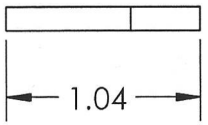


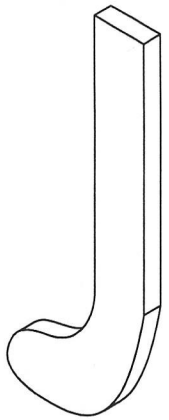
The Ohio State University First Year Engineering	Dwg. Title: SMALL ARM PART 2	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-01
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

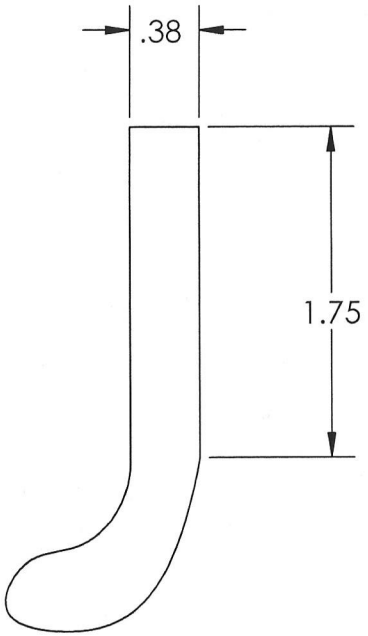


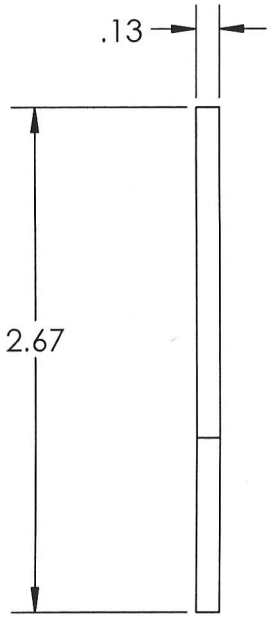


The Ohio State University First Year Engineering	Dwg. Title: SMALL ARM PART 3	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-02
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

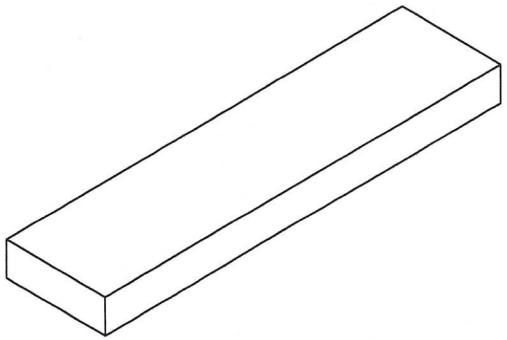


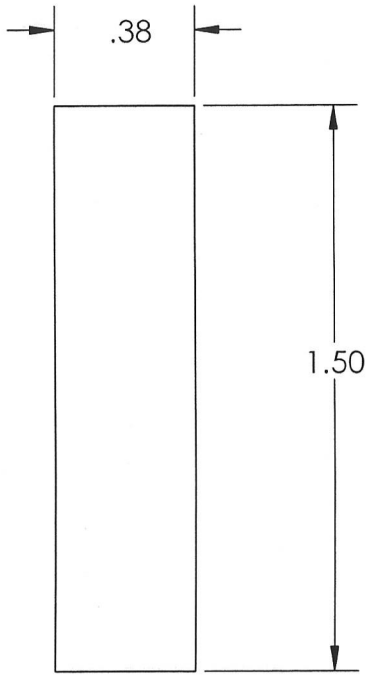


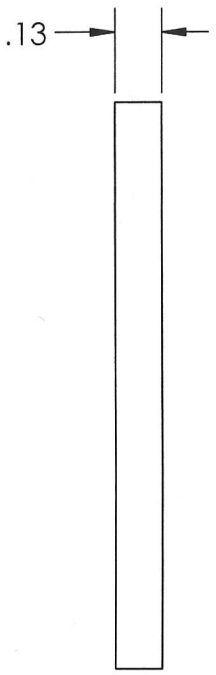




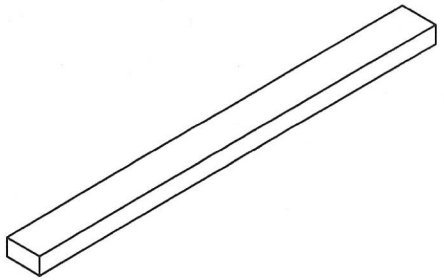
The Ohio State University First Year Engineering	Dwg. Title: PART 7	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-03
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

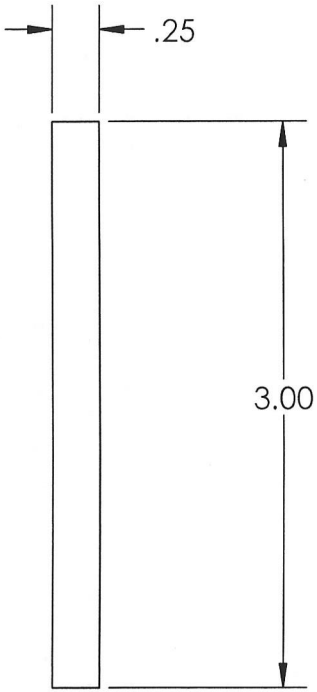


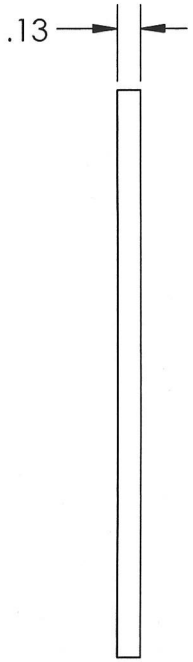




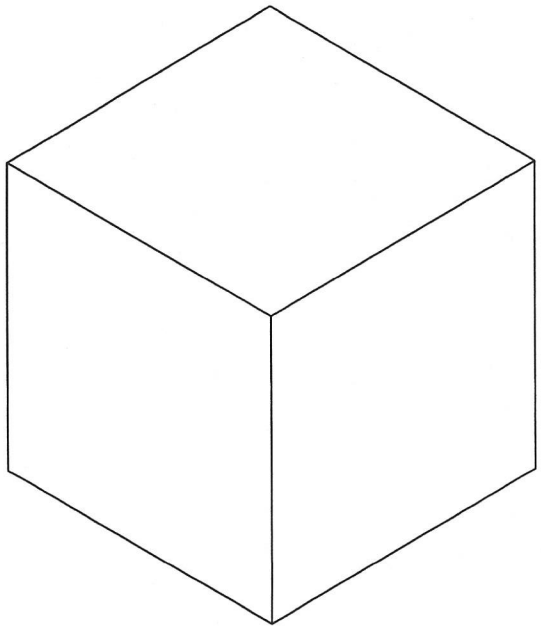
The Ohio State University First Year Engineering	Dwg. Title: PART 6	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-04
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

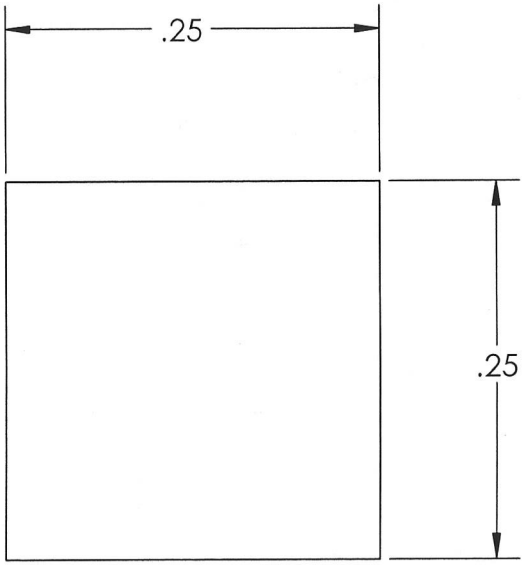


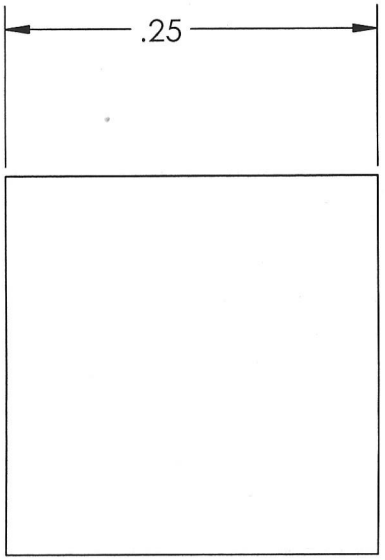




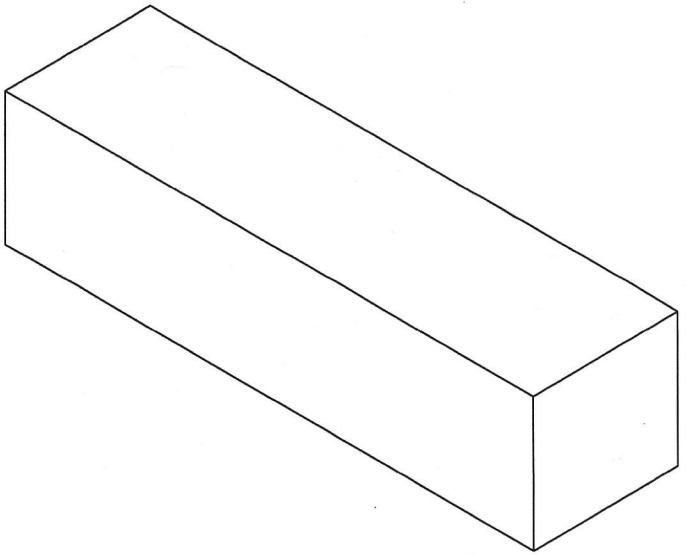
The Ohio State University First Year Engineering	Dwg. Title: SERVO ARM PART	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-05
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

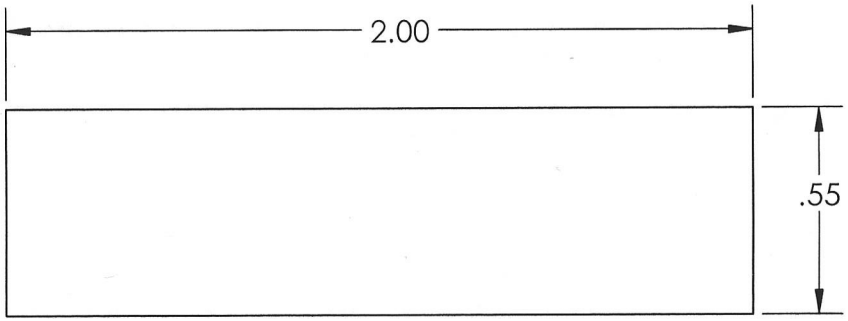


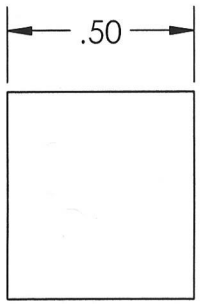




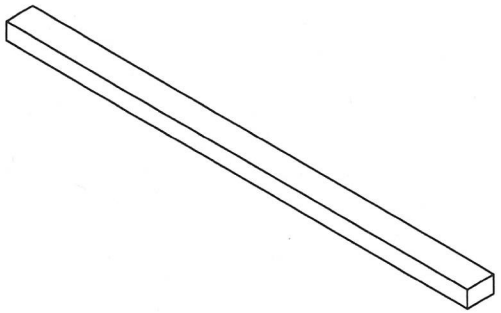
The Ohio State University First Year Engineering	Dwg. Title: PART 4	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-06
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

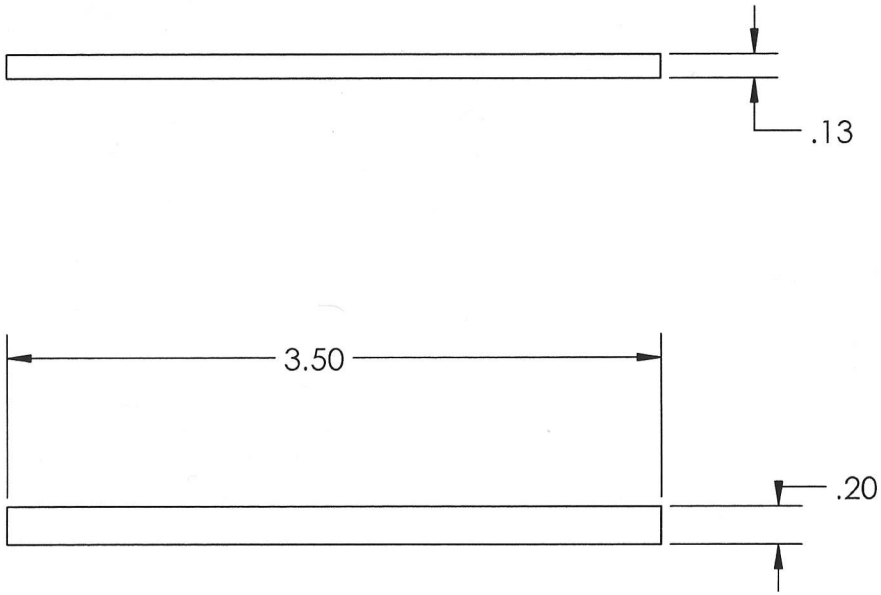




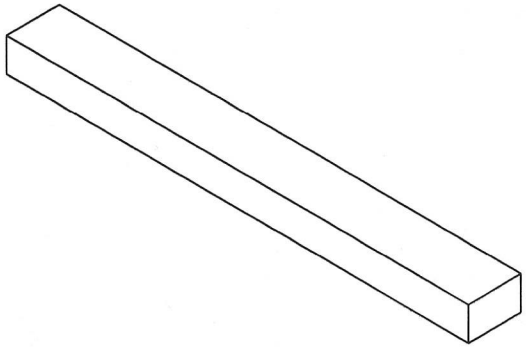


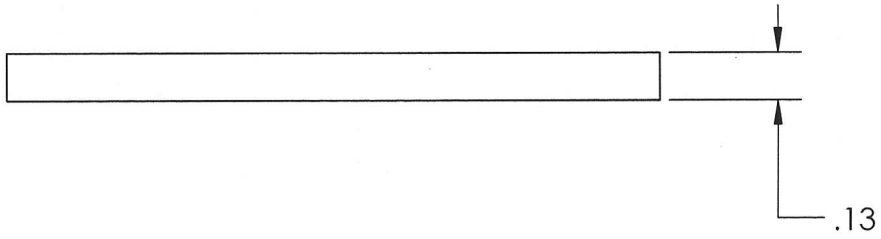
The Ohio State University First Year Engineering	Dwg. Title: PART 3	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-07
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

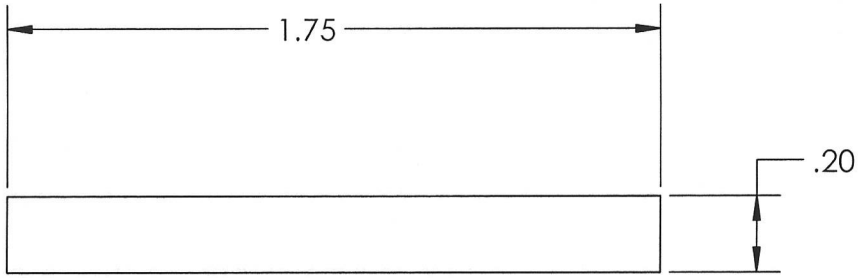




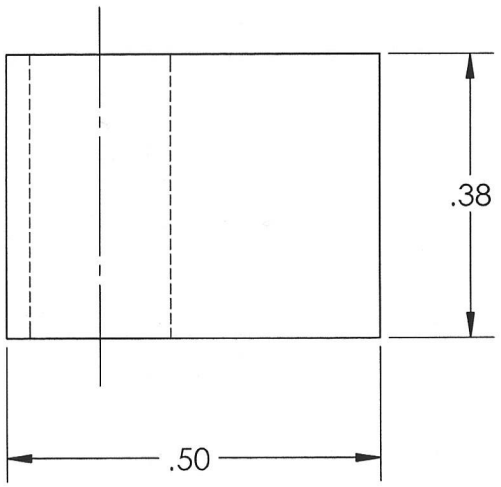
The Ohio State University First Year Engineering	Dwg. Title: PVC SUPPORT QR	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-08
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

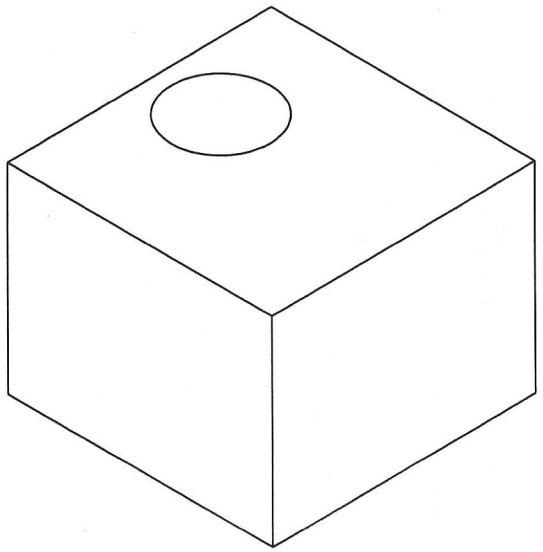


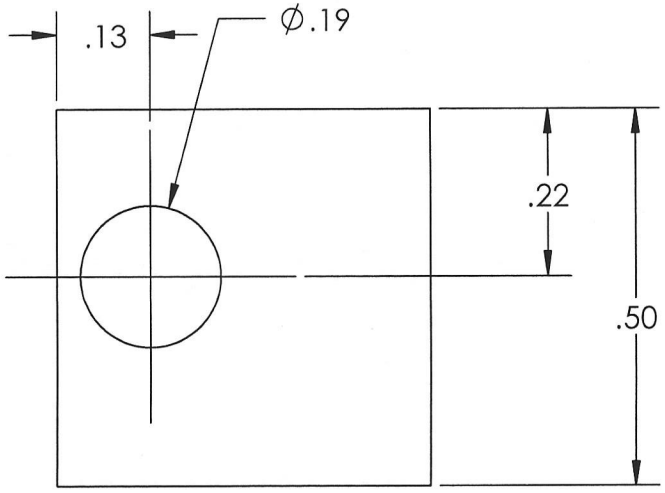




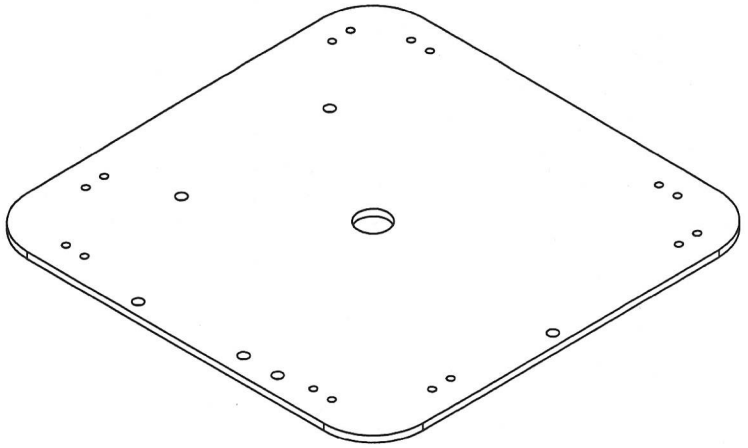
The Ohio State University First Year Engineering	Dwg. Title: PVC SUPPORT QR 2	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-09
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

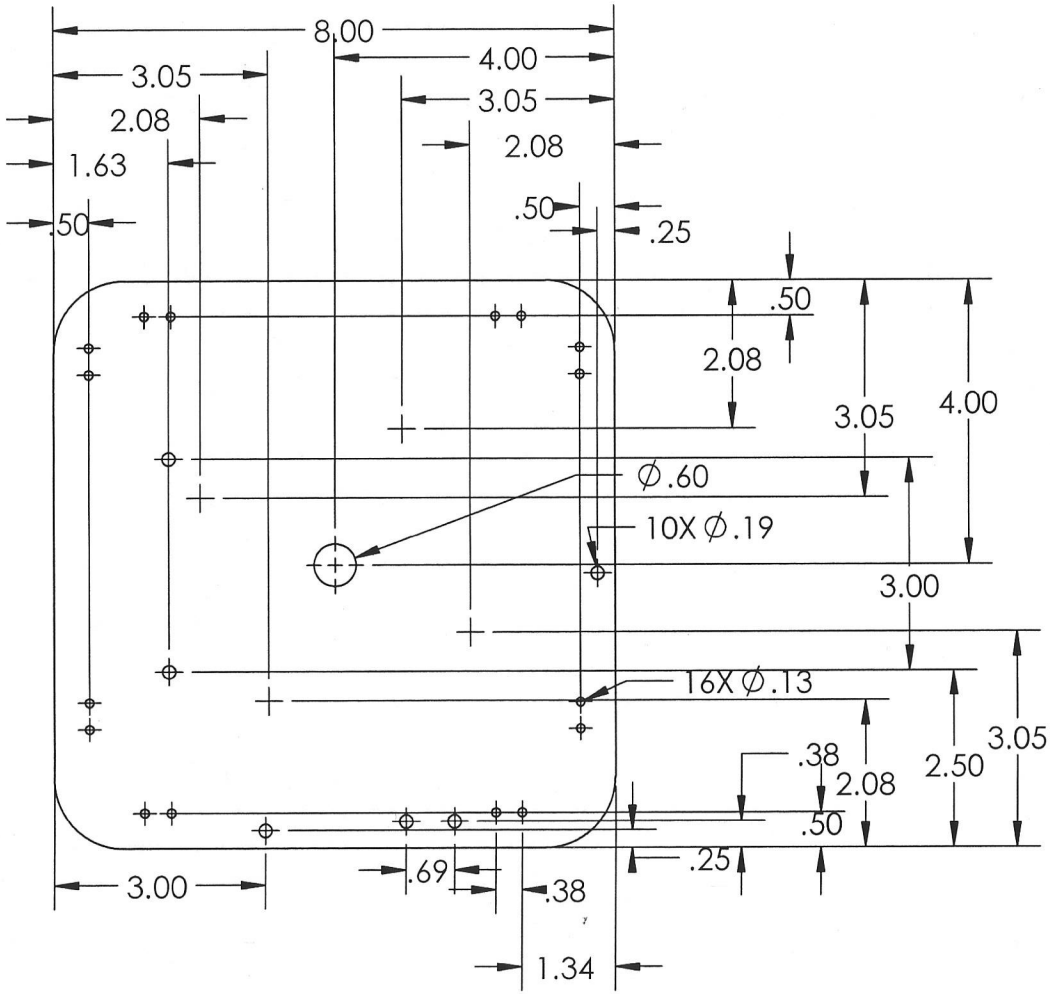


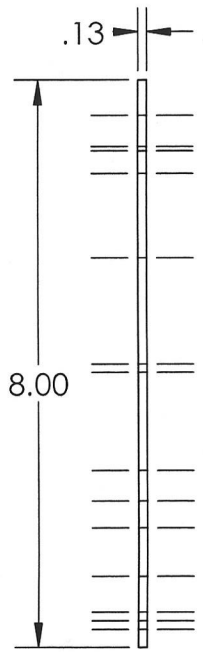




The Ohio State University First Year Engineering	Dwg. Title: RED BUTTON ELEVATION BLOCK	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-10
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

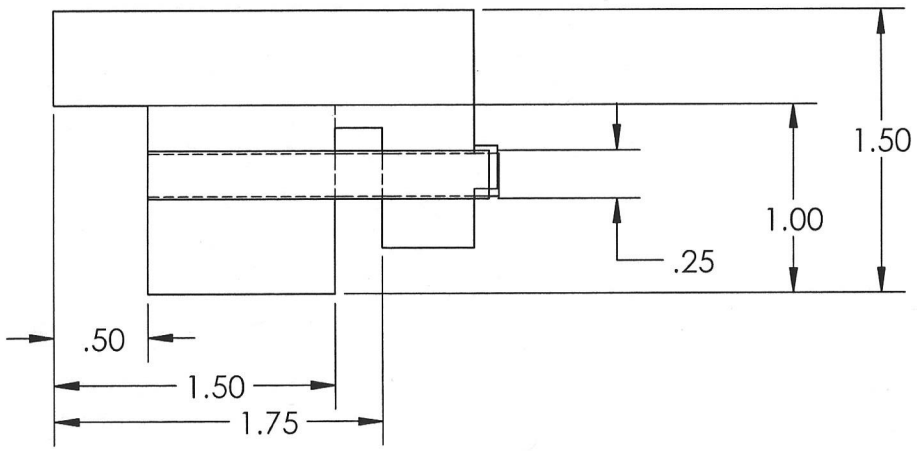


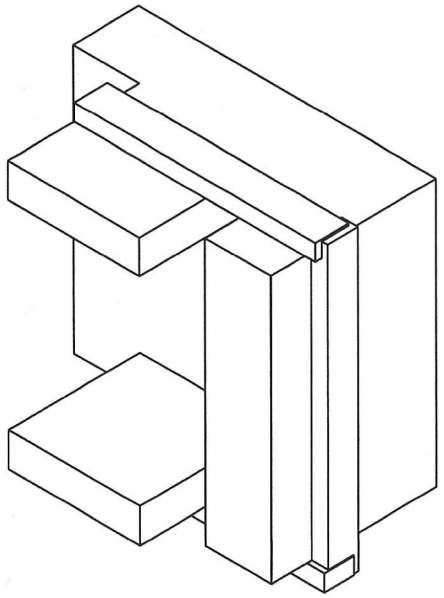


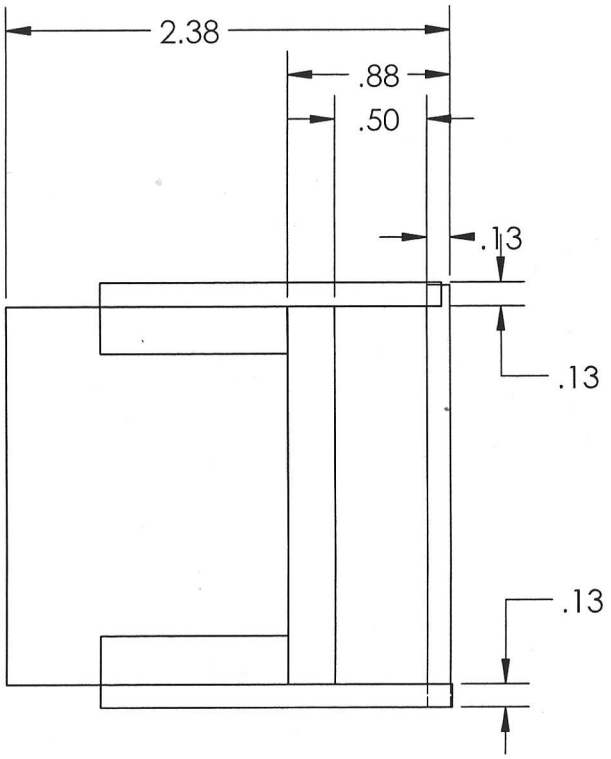


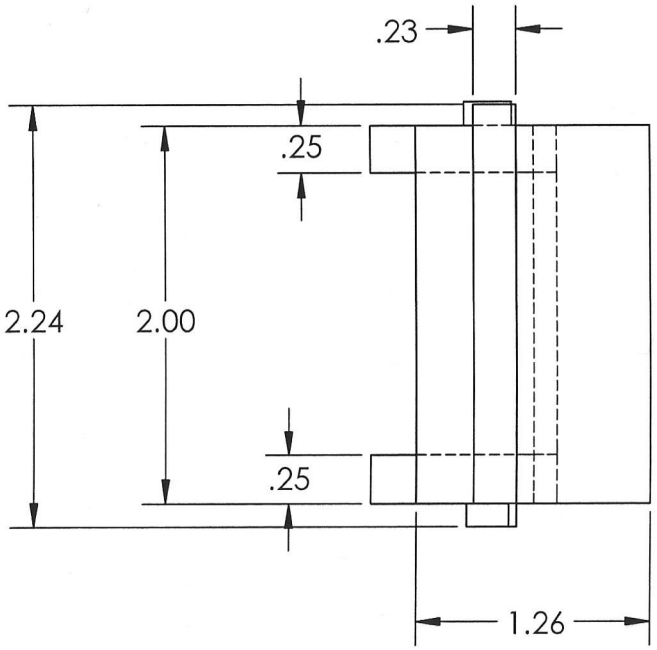
ALL PAIRS OF PERIMETER .13 IN DIAMETER HOLES ARE DIMENSIONED AS SHOWN IN THE PAIR IN THE BOTTOM RIGHT CORNER OF THE BASE.

The Ohio State University First Year Engineering	Dwg. Title: PVC SHEET 1	Inst.: RJF	Scale: 3:8	Dwg. No.: DD-11
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

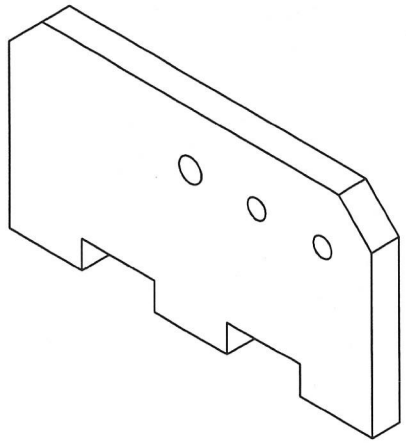


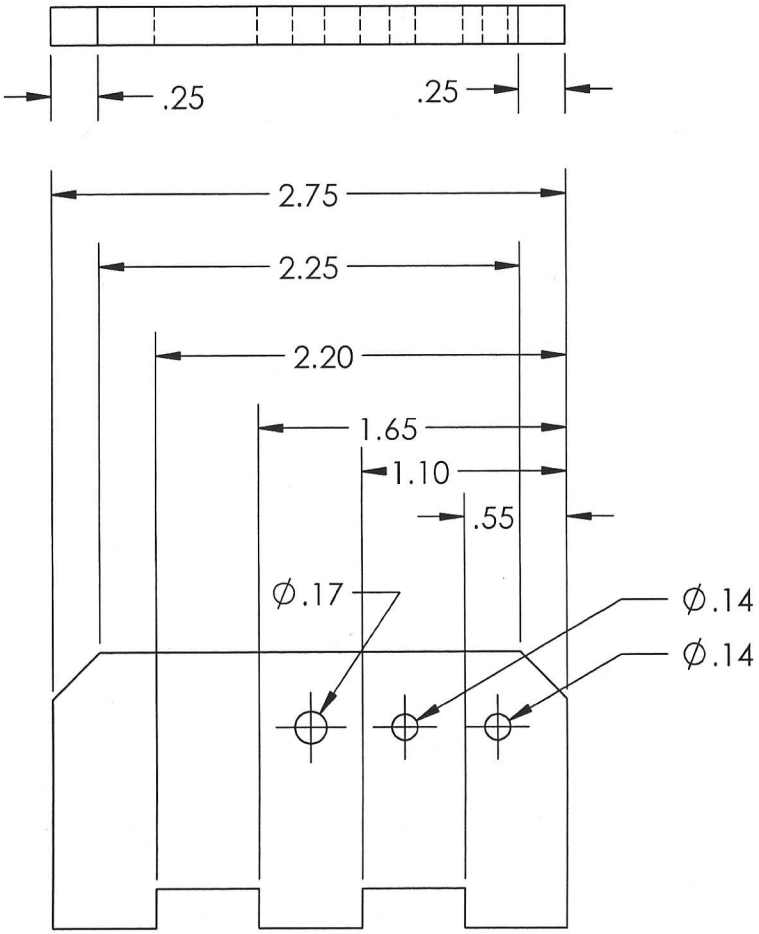


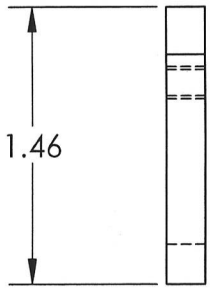




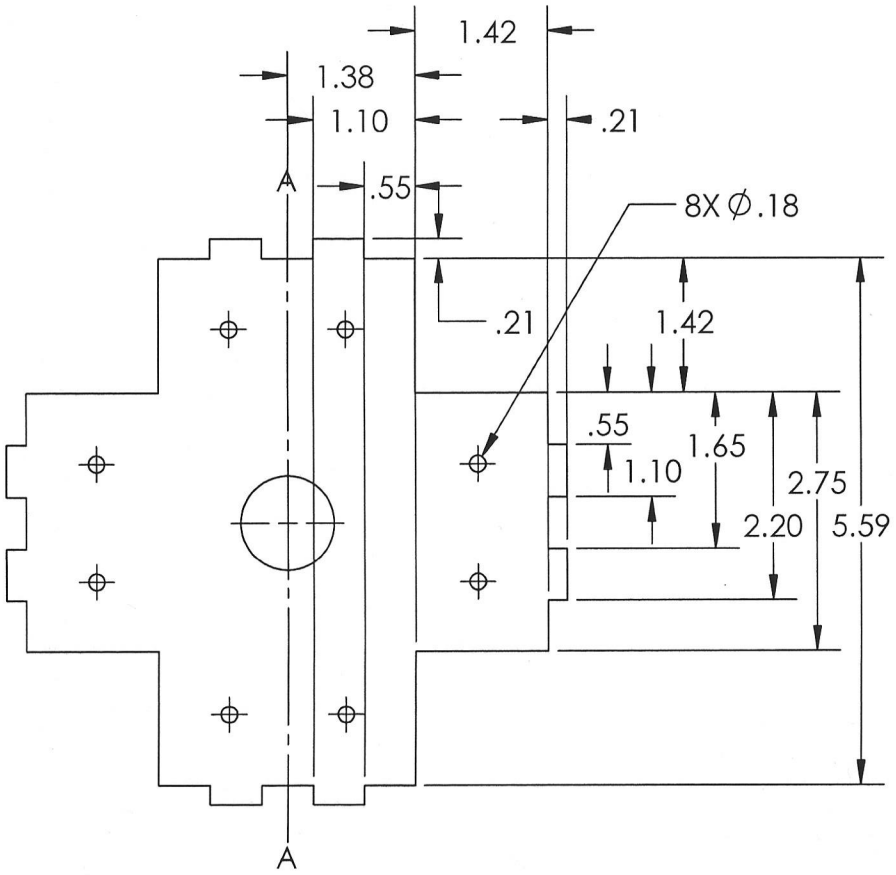
The Ohio State University First Year Engineering	Dwg. Title: SERVO MOUNT	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-12
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

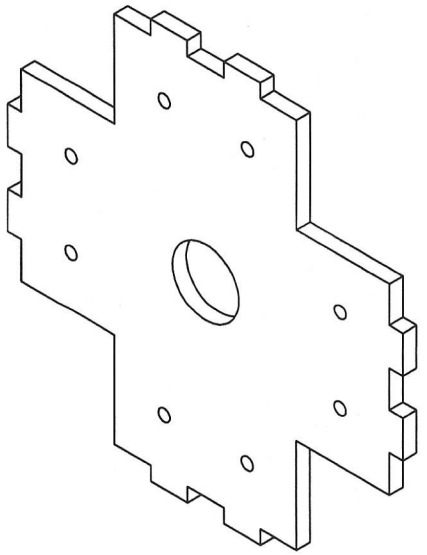


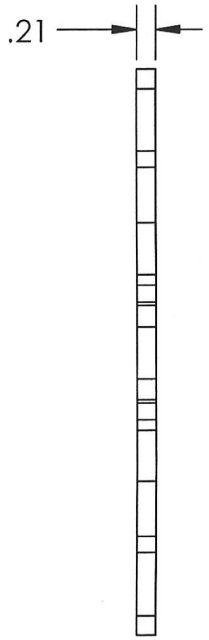




The Ohio State University First Year Engineering	Dwg. Title: MOTOR MOUNT SIDES	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-13
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

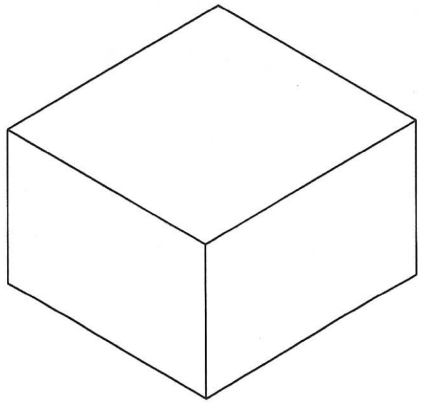


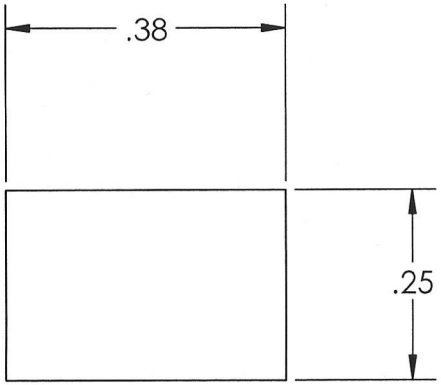


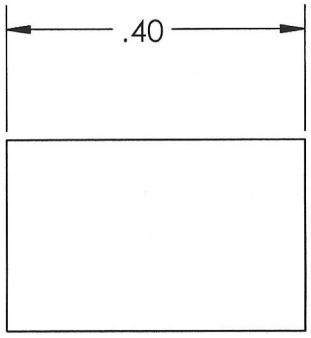


THE PART IS SYMMETRIC ABOUT THE LINE A-A

The Ohio State University First Year Engineering	Dwg. Title: MOTOR MOUNT BASE	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-14
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016







The Ohio State University First Year Engineering	Dwg. Title: SHORT ARM CATCH	Inst.: RJF	Scale: 1:1	Dwg. No.: DD-15
	Drawn By: LOGAN MEYER	Hour: 8:00	Units: IPS	Date: 4/10/2016

APPENDIX H

Final Code

main.cpp

```
// Required custom library
#include "RobotMenuMain.h"

/* This main function runs the RobotMainMenu, which can access all of the subprograms.
 * Last modified: 3/14/2016
 */
int main(void)
{
    while(true) {
        RobotMenu.Run();
    }
} // end main function
```

RobotMenuMain.h

```
#ifndef ROBOTMAINMENU_H
#define ROBOTMAINMENU_H

class RobotMenuMain {
public:
    void Run();
private:
    int MNMenu();
};

extern RobotMenuMain RobotMenu;

#endif // ROBOTMAINMENU_H
```

RobotMenuMain.cpp

```
// Required libraries
#include <FEHLCD.h>
#include <string.h>
#include "RobotMenuMain.h"

// Included custom libraries. Comment out the ones not needed.
// Be sure to comment out the functions in the code farther down in RobotMenuMain::Run()
//#include "ProteusTestMain.h"
#include "WorldStateMain.h"
#include "FinalCompetitionMain.h"
#include "FinalCompetitionFastMain.h"

// Define the colors for the menus
#define MENU_C SCARLET
#define TEXT_C BLACK

// Declare the object of type RobotMenuMain
RobotMenuMain RobotMenu;

/* This function draws the main menu, accepts input, requests for confirmation, and returns the
choice.
 * This code was mostly inspired by Proteus_Test.cpp and FEHRPS.cpp.
 * Last Modified: 3/14/2016 JKL
 */
int RobotMenuMain::MNMenu(void)
{
    // Initialize/reset choice and menu
    int confirmChoice = 0;
```

```
    int menu = 0;

    // Initialize the main menu title, main menu labels, confirmation title, and confirmation menu
labels.
    FEHIcon::Icon MAIN_T[1];
    char main_t_label[1][20] = {"Proteus Main Menu"};

    FEHIcon::Icon MAIN[6];
    char main_label[6][20] = {"Proteus Test","WorldState","Blank","FinalComp","Test1","FinalFast"};

    FEHIcon::Icon confirm_title[1];
    char confirm_title_label[1][20] = {""};

    FEHIcon::Icon confirm[2];
    char confirm_labels[2][20] = {"Ok", "Cancel"};

    while (confirmChoice != 1) {
        LCD.Clear(GRAY);

        // Draw the title block
        FEHIcon::DrawIconArray(MAIN_T, 1, 1, 1, 201, 1, 1, main_t_label, MENU_C, TEXT_C);
        MAIN_T[0].Select();

        // Draw the menu options with 3 rows and 2 columns
        FEHIcon::DrawIconArray(MAIN, 3, 2, 40, 1, 1, 1, main_label, MENU_C, TEXT_C);

        float x, y;

        menu = 0;

        while(menu==0) {
            if (LCD.Touch(&x, &y)) {
                for (int n=0; n<=9; n++) {
                    if (MAIN[n].Pressed(x, y, 0)) {
                        menu = n + 1;
                        MAIN[n].WhilePressed(x, y);
                        break;
                    }
                }
            }
        } // end while loop that repeats until a icon is pressed

        strcpy(confirm_title_label[0], main_label[menu-1]);

        LCD.Clear(GRAY);
        FEHIcon::DrawIconArray(confirm_title, 1, 1, 60, 201, 1, 1, confirm_title_label, MENU_C,
TEXT_C);
        FEHIcon::DrawIconArray(confirm, 1, 2, 60, 60, 1, 1, confirm_labels, MENU_C, TEXT_C);

        confirmChoice = 0;

        while (!confirmChoice) {
            if (LCD.Touch(&x, &y)) {
                for (int n=0; n <=1; n++) {
                    if (confirm[n].Pressed(x, y, 0)) {
                        confirm[n].WhilePressed(x, y);
                        confirmChoice = n+1;
                    }
                }
            }
        } // end while loop that repeats until a confirmation choice
    } // end while loop the repeats until positive confirmation
```

```
    return menu;
} // end RobotMenuMain::MNMenu function

/* This function is the main function that runs in main.cpp.
 * It contains the switch-case options for the menu choices and runs the included programs.
 * Be sure to comment out the subprograms not included at the top of this file.
 * Last Modified: 3/14/2016 JKL
 */
void RobotMenuMain::Run() {
    int choice = MNMenu();
    LCD.Clear(BLACK);
    switch (choice) {
    case 1:
        LCD.WriteLine("Proteus Test");
        //ProteusTest.Run();
        break;
    case 2:
        LCD.WriteLine("WorldState");
        WorldState.Run();
        break;
    case 3:
        LCD.WriteLine("Blank");
        break;
    case 4:
        LCD.WriteLine("Final Competition");
        FinalCompetition.Run();
        break;
    case 5:
        LCD.WriteLine("Test1");
        break;
    case 6:
        LCD.WriteLine("Final Fast");
        FinalCompetitionFast.Run();
        break;

    } // end switch-case structure
} // end RobotMenuMain::Run function
```

ProteusTest.h

```
#ifndef PROTEUSTEST_H
#define PROTEUSTEST_H

#define PROTEUSTESTMAIN 1

class ProteusTestMain {
public:
    void Run();
};

extern ProteusTestMain ProteusTest;

#endif // PROTEUSTEST_H
```

ProteusTest.cpp

```
/*******************************/
/*      Proteus Test Code      */
/*     OSU FEH Spring 2016     */
/*        Drew Phillips        */
```

```
/*    02/04/16  Version 3.0     */
/*******************************/

/* Include preprocessor directives */
#include <FEHLCD.h>
#include <FEHIO.h>
#include <FEHUtility.h>
#include <FEHMotor.h>
#include <FEHServo.h>
#include <FEHAccel.h>
#include <FEHBattery.h>
#include <FEHBuzzer.h>
#include <FEHRPS.h>
#include <string.h>
#include <stdio.h>
#include "ProteusTestMain.h"

/* Define colors for parts of menus */
#define MENU_C WHITE
#define TEXT_C GOLD
#define SELT_C RED
#define SHOW_C BLUE
#define HI_C GREEN

/* Define menu number values */
#define MN_MENU 0
#define DC_MENU 1
#define SV_MENU 2
#define DI_MENU 3
#define AI_MENU 4
#define AC_MENU 5
#define TO_MENU 6
#define DO_MENU 7
#define RP_MENU 8

/* Define time for beep */
#define beep_t 10 // int milliseconds

ProteusTestMain ProteusTest;

/* Global variable to keep track of being initialized to RPS */
int RPS_init = 0;

/* Function to write input value as true or false to icon */
void WriteLogicValue(FEHIcon::Icon icon, int val)
{
    if (val)
        icon.ChangeLabelString("True");
    else
        icon.ChangeLabelString("False");
} // end WriteLogicValue function

/* Main Menu */
int MNMenu()
{
    LCD.Clear(BLACK);

    /* Create icons for main menu */
    FEHIcon::Icon MAIN_T[1];
    char main_t_label[1][20] = {"PROTEUS TEST CODE"};
    FEHIcon::DrawIconArray(MAIN_T, 1, 1, 1, 201, 1, 1, main_t_label, HI_C, TEXT_C);
    MAIN_T[0].Select();
```

```
    FEHIcon::Icon MAIN[6];
    char main_label[8][20] = {"DC", "Servo", "Digital In", "Analog In", "Accel", "Touch", "Digital
Out", "RPS"};
    FEHIcon::DrawIconArray(MAIN, 4, 2, 40, 20, 1, 1, main_label, MENU_C, TEXT_C);

    LCD.SetFontColor(HI_C);
    LCD.WriteAt("AHP V3.0", 220, 222);

    LCD.SetFontColor(TEXT_C);
    LCD.WriteAt("BATT:        V", 0, 222);

    Buzzer.Buzz(beep_t);

    int menu=MN_MENU, n;
    float x, y;
    float m = 0, bat_v = 0;

    while(menu==MN_MENU)
    {
        /* Display average battery voltage to screen */
        bat_v = ((bat_v*m)+Battery.Voltage());
        bat_v = bat_v/(++m);
        LCD.WriteAt(bat_v, 72, 222);
        if (LCD.Touch(&x, &y))
        {
            /* Check to see if a main menu icon has been touched */
            for (n=0; n<=7; n++)
            {
                if (MAIN[n].Pressed(x, y, 0))
                {
                    menu = n+1;
                    MAIN[n].WhilePressed(x, y);
                    break;
                }
            }
        }
    } // end while loop
    return menu;
} // end MNMenu function

/* DC Motors Menu */
int DCMenu()
{
    /* Declare DC Motor ports */
    FEHMotor motor0 (FEHMotor::Motor0, 5.0);
    FEHMotor motor1 (FEHMotor::Motor1, 5.0);
    FEHMotor motor2 (FEHMotor::Motor2, 5.0);
    FEHMotor motor3 (FEHMotor::Motor3, 5.0);

    LCD.Clear(BLACK);

    /* Create icons for DC motors menu */
    FEHIcon::Icon DC_T[1];
    char dc_t_label[1][20] = {"DC Motors"};
    FEHIcon::DrawIconArray(DC_T, 1, 1, 1, 201, 1, 1, dc_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon DC[4];
```

```
    char dc_labels[4][20] = {"Motor0", "Motor1", "Motor2", "Motor3"};
    FEHIcon::DrawIconArray(DC, 2, 2, 40, 1, 1, 60, dc_labels, SHOW_C, TEXT_C);

    FEHIcon::Icon Run[2];
    char run_labels[2][20] = {"F", "B"};
    FEHIcon::DrawIconArray(Run, 2, 1, 40, 1, 261, 1, run_labels, SELT_C, TEXT_C);

    Buzzer.Buzz(beep_t);

    int menu=DC_MENU, n, m;
    float x, y;
    int run[4] = {0, 0, 0, 0};

    while(menu==DC_MENU)
    {
        if (LCD.Touch(&x, &y))
        {
            /* Check to see if a motor icon has been touched */
            for (n=0; n<=3; n++)
            {
                if (DC[n].Pressed(x, y, 0))
                {
                    /* Select or deselect motor icon for running */
                    if (!run[n])
                        run[n] = 1;
                    else if (run[n])
                        run[n] = 0;
                    DC[n].WhilePressed(x, y);
                }
            }
            /* Check to see if a forward or backward run button has been touched */
            for (m=0; m<=1; m++)
            {
                if (Run[m].Pressed(x, y, 0))
                {
                    /* Run motors according to direction */
                    if (run[0]==1)
                        motor0.SetPercent((m*(-100))+50);
                    if (run[1]==1)
                        motor1.SetPercent((m*(-100))+50);
                    if (run[2]==1)
                        motor2.SetPercent((m*(-100))+50);
                    if (run[3]==1)
                        motor3.SetPercent((m*(-100))+50);
                    Run[m].WhilePressed(x, y);
                    Run[m].Deselect();
                    motor0.Stop();
                    motor1.Stop();
                    motor2.Stop();
                    motor3.Stop();
                }
            }
            /* If back button has been touched, go to main menu */
            if (Back[0].Pressed(x, y, 0))
            {
                Back[0].WhilePressed(x, y);
                menu = MN_MENU;
            }
        }
    } // end while loop
    return menu;
} // end DCMenu function
```

```
/* Servo Motors Menu */
int SVMenu()
{
    /* Declare Servo Motor ports */
    FEHServo servo0 (FEHServo::Servo0);
    FEHServo servo1 (FEHServo::Servo1);
    FEHServo servo2 (FEHServo::Servo2);
    FEHServo servo3 (FEHServo::Servo3);
    FEHServo servo4 (FEHServo::Servo4);
    FEHServo servo5 (FEHServo::Servo5);
    FEHServo servo6 (FEHServo::Servo6);
    FEHServo servo7 (FEHServo::Servo7);

    LCD.Clear(BLACK);

    /* Create icons for servo motors menu */
    FEHIcon::Icon SV_T[1];
    char sv_t_label[1][20] = {"Servo Motors"};
    FEHIcon::DrawIconArray(SV_T, 1, 1, 1, 201, 1, 1, sv_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon SV[8];
    char sv_labels[8][20] = {"Servo0", "Servo1", "Servo2", "Servo3", "Servo4", "Servo5", "Servo6",
"Servo7"};
    FEHIcon::DrawIconArray(SV, 4, 2, 40, 1, 1, 60, sv_labels, SHOW_C, TEXT_C);

    FEHIcon::Icon Run[2];
    char run_labels[2][20] = {"F", "B"};
    FEHIcon::DrawIconArray(Run, 2, 1, 40, 41, 261, 1, run_labels, SELT_C, TEXT_C);

    FEHIcon::Icon Deg[1];
    char deg_labels[1][20] = {"0"};
    FEHIcon::DrawIconArray(Deg, 1, 1, 198, 4, 261, 1, deg_labels, SELT_C, TEXT_C);

    Buzzer.Buzz(beep_t);

    int menu=SV_MENU, n, m;
    float x, y;
    int deg=0;
    int run[8] = {0, 0, 0, 0, 0, 0, 0, 0};

    while(menu==SV_MENU)
    {
        if (LCD.Touch(&x, &y))
        {
            /* Check to see if a servo motor icon has been touched */
            for (n=0; n<=7; n++)
            {
                if (SV[n].Pressed(x, y, 0))
                {
                    /* Select or deselect motor icon for running */
                    if (!run[n])
                        run[n] = 1;
                    else if (run[n])
                        run[n] = 0;
                    SV[n].WhilePressed(x, y);
                }
            } // end for loop
```

```
            /* Check to see if the forward or backward run icons have been touched */
            for (m=0; m<=1; m++)
            {
                if (Run[m].Pressed(x, y, 0))
                {
                    while(Run[m].Pressed(x, y, 1))
                    {
                        /* While the run button is being touched, increase or decrease the servo
angle within the limits */
                        LCD.Touch(&x, &y);
                        deg = deg+1-(m*2);
                        if (deg<0)
                            deg = 0;
                        else if (deg>180)
                            deg = 180;
                        Deg[0].ChangeLabelInt(deg);
                        if (run[0]==1)
                            servo0.SetDegree(deg);
                        if (run[1]==1)
                            servo1.SetDegree(deg);
                        if (run[2]==1)
                            servo2.SetDegree(deg);
                        if (run[3]==1)
                            servo3.SetDegree(deg);
                        if (run[4]==1)
                            servo4.SetDegree(deg);
                        if (run[5]==1)
                            servo5.SetDegree(deg);
                        if (run[6]==1)
                            servo6.SetDegree(deg);
                        if (run[7]==1)
                            servo7.SetDegree(deg);
                    }
                    Run[m].Deselect();
                }
            } // end for loop
            /* If back button has been touched, go to main menu */
            if (Back[0].Pressed(x, y, 0))
            {
                Back[0].WhilePressed(x, y);
                menu = MN_MENU;
            }
        }
    } // end while loop
    return menu;
} // end SVMenu function

/* Digital Input Menu */
int DIMenu()
{
    /* Declare ports for digital input */
    DigitalInputPin di00 (FEHIO::P0_0);
    DigitalInputPin di01 (FEHIO::P0_1);
    DigitalInputPin di02 (FEHIO::P0_2);
    DigitalInputPin di03 (FEHIO::P0_3);
    DigitalInputPin di04 (FEHIO::P0_4);
    DigitalInputPin di05 (FEHIO::P0_5);
    DigitalInputPin di06 (FEHIO::P0_6);
    DigitalInputPin di07 (FEHIO::P0_7);
    DigitalInputPin di10 (FEHIO::P1_0);
    DigitalInputPin di11 (FEHIO::P1_1);
    DigitalInputPin di12 (FEHIO::P1_2);
```

```
    DigitalInputPin di13 (FEHIO::P1_3);
    DigitalInputPin di14 (FEHIO::P1_4);
    DigitalInputPin di15 (FEHIO::P1_5);
    DigitalInputPin di16 (FEHIO::P1_6);
    DigitalInputPin di17 (FEHIO::P1_7);
    DigitalInputPin di20 (FEHIO::P2_0);
    DigitalInputPin di21 (FEHIO::P2_1);
    DigitalInputPin di22 (FEHIO::P2_2);
    DigitalInputPin di23 (FEHIO::P2_3);
    DigitalInputPin di24 (FEHIO::P2_4);
    DigitalInputPin di25 (FEHIO::P2_5);
    DigitalInputPin di26 (FEHIO::P2_6);
    DigitalInputPin di27 (FEHIO::P2_7);
    DigitalInputPin di30 (FEHIO::P3_0);
    DigitalInputPin di31 (FEHIO::P3_1);
    DigitalInputPin di32 (FEHIO::P3_2);
    DigitalInputPin di33 (FEHIO::P3_3);
    DigitalInputPin di34 (FEHIO::P3_4);
    DigitalInputPin di35 (FEHIO::P3_5);
    DigitalInputPin di36 (FEHIO::P3_6);
    DigitalInputPin di37 (FEHIO::P3_7);

    LCD.Clear(BLACK);

    /* Create digital input menu icons */
    FEHIcon::Icon DI_T[1];
    char di_t_label[1][20] = {"Digital Input"};
    FEHIcon::DrawIconArray(DI_T, 1, 1, 1, 201, 1, 1, di_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon BANKS[4];
    char banks_label[4][20] = {"Bank0", "Bank1", "Bank2", "Bank3"};
    FEHIcon::DrawIconArray(BANKS, 1, 4, 40, 161, 1, 1, banks_label, SELT_C, TEXT_C);

    FEHIcon::Icon DI_B0[16];
    char di_b0_labels[16][20] = {"P0_0", "P0_1", "P0_2", "P0_3", " ", " ", " ", " ", "P0_4",
"P0_5", "P0_6", "P0_7", " ", " ", " ", " "};

    FEHIcon::Icon DI_B1[16];
    char di_b1_labels[16][20] = {"P1_0", "P1_1", "P1_2", "P1_3", " ", " ", " ", " ", "P1_4",
"P1_5", "P1_6", "P1_7", " ", " ", " ", " "};

    FEHIcon::Icon DI_B2[16];
    char di_b2_labels[16][20] = {"P2_0", "P2_1", "P2_2", "P2_3", " ", " ", " ", " ", "P2_4",
"P2_5", "P2_6", "P2_7", " ", " ", " ", " "};

    FEHIcon::Icon DI_B3[16];
    char di_b3_labels[16][20] = {"P3_0", "P3_1", "P3_2", "P3_3", " ", " ", " ", " ", "P3_4",
"P3_5", "P3_6", "P3_7", " ", " ", " ", " "};

    Buzzer.Buzz(beep_t);

    int menu=DI_MENU, bank=0, bank_i, n;
    float x, y;

    while(menu==DI_MENU)
    {
        /* Draw selected bank's ports */
        switch (bank)
```

```
{
case 0:
    FEHIcon::DrawIconArray(DI_B0, 4, 4, 80, 1, 1, 1, di_b0_labels, SHOW_C, TEXT_C);
    break;
case 1:
    FEHIcon::DrawIconArray(DI_B1, 4, 4, 80, 1, 1, 1, di_b1_labels, SHOW_C, TEXT_C);
    break;
case 2:
    FEHIcon::DrawIconArray(DI_B2, 4, 4, 80, 1, 1, 1, di_b2_labels, SHOW_C, TEXT_C);
    break;
case 3:
    FEHIcon::DrawIconArray(DI_B3, 4, 4, 80, 1, 1, 1, di_b3_labels, SHOW_C, TEXT_C);
    break;
}
bank_i = bank;
while(bank==bank_i)
{
    /* For each bank, display the digital input values */
    if (bank==0)
    {
        WriteLogicValue(DI_B0[4], di00.Value());
        WriteLogicValue(DI_B0[5], di01.Value());
        WriteLogicValue(DI_B0[6], di02.Value());
        WriteLogicValue(DI_B0[7], di03.Value());
        WriteLogicValue(DI_B0[12], di04.Value());
        WriteLogicValue(DI_B0[13], di05.Value());
        WriteLogicValue(DI_B0[14], di06.Value());
        WriteLogicValue(DI_B0[15], di07.Value());
    }
    else if (bank==1)
    {
        WriteLogicValue(DI_B1[4], di10.Value());
        WriteLogicValue(DI_B1[5], di11.Value());
        WriteLogicValue(DI_B1[6], di12.Value());
        WriteLogicValue(DI_B1[7], di13.Value());
        WriteLogicValue(DI_B1[12], di14.Value());
        WriteLogicValue(DI_B1[13], di15.Value());
        WriteLogicValue(DI_B1[14], di16.Value());
        WriteLogicValue(DI_B1[15], di17.Value());
    }
    else if (bank==2)
    {
        WriteLogicValue(DI_B2[4], di20.Value());
        WriteLogicValue(DI_B2[5], di21.Value());
        WriteLogicValue(DI_B2[6], di22.Value());
        WriteLogicValue(DI_B2[7], di23.Value());
        WriteLogicValue(DI_B2[12], di24.Value());
        WriteLogicValue(DI_B2[13], di25.Value());
        WriteLogicValue(DI_B2[14], di26.Value());
        WriteLogicValue(DI_B2[15], di27.Value());
    }
    else if (bank==3)
    {
        WriteLogicValue(DI_B3[4], di30.Value());
        WriteLogicValue(DI_B3[5], di31.Value());
        WriteLogicValue(DI_B3[6], di32.Value());
        WriteLogicValue(DI_B3[7], di33.Value());
        WriteLogicValue(DI_B3[12], di34.Value());
        WriteLogicValue(DI_B3[13], di35.Value());
        WriteLogicValue(DI_B3[14], di36.Value());
        WriteLogicValue(DI_B3[15], di37.Value());
    }
```

```
            if (LCD.Touch(&x, &y))
            {
                /* Check to see if any of the banks icons have been touched */
                for (n=0; n<=3; n++)
                {
                    if (BANKS[n].Pressed(x, y, 0))
                    {
                        /* Change selected bank number to update screen */
                        BANKS[n].WhilePressed(x, y);
                        BANKS[n].Deselect();
                        bank = n;
                    }
                }
                /* If back button has been touched, go to main menu */
                if (Back[0].Pressed(x, y, 0))
                {
                    Back[0].WhilePressed(x, y);
                    menu = MN_MENU;
                    bank = -1;
                }
            }
        } // end while loop
    } // end while loop
    return menu;
} // end DIMenu function

/* Analog Input Menu */
int AIMenu()
{
    /* Declare ports for digital input */  // Trust me, you have to declare them as digital and
analog inputs in order for this to work
    DigitalInputPin di00 (FEHIO::P0_0);
    DigitalInputPin di01 (FEHIO::P0_1);
    DigitalInputPin di02 (FEHIO::P0_2);
    DigitalInputPin di03 (FEHIO::P0_3);
    DigitalInputPin di04 (FEHIO::P0_4);
    DigitalInputPin di05 (FEHIO::P0_5);
    DigitalInputPin di06 (FEHIO::P0_6);
    DigitalInputPin di07 (FEHIO::P0_7);
    DigitalInputPin di10 (FEHIO::P1_0);
    DigitalInputPin di11 (FEHIO::P1_1);
    DigitalInputPin di12 (FEHIO::P1_2);
    DigitalInputPin di13 (FEHIO::P1_3);
    DigitalInputPin di14 (FEHIO::P1_4);
    DigitalInputPin di15 (FEHIO::P1_5);
    DigitalInputPin di16 (FEHIO::P1_6);
    DigitalInputPin di17 (FEHIO::P1_7);
    DigitalInputPin di20 (FEHIO::P2_0);
    DigitalInputPin di21 (FEHIO::P2_1);
    DigitalInputPin di22 (FEHIO::P2_2);
    DigitalInputPin di23 (FEHIO::P2_3);
    DigitalInputPin di24 (FEHIO::P2_4);
    DigitalInputPin di25 (FEHIO::P2_5);
    DigitalInputPin di26 (FEHIO::P2_6);
    DigitalInputPin di27 (FEHIO::P2_7);
    DigitalInputPin di30 (FEHIO::P3_0);
    DigitalInputPin di31 (FEHIO::P3_1);
    DigitalInputPin di32 (FEHIO::P3_2);
    DigitalInputPin di33 (FEHIO::P3_3);
    DigitalInputPin di34 (FEHIO::P3_4);
    DigitalInputPin di35 (FEHIO::P3_5);
    DigitalInputPin di36 (FEHIO::P3_6);
```

```
    DigitalInputPin di37 (FEHIO::P3_7);

    /* Declare ports for analog input */
    AnalogInputPin ai00 (FEHIO::P0_0);
    AnalogInputPin ai01 (FEHIO::P0_1);
    AnalogInputPin ai02 (FEHIO::P0_2);
    AnalogInputPin ai03 (FEHIO::P0_3);
    AnalogInputPin ai04 (FEHIO::P0_4);
    AnalogInputPin ai05 (FEHIO::P0_5);
    AnalogInputPin ai06 (FEHIO::P0_6);
    AnalogInputPin ai07 (FEHIO::P0_7);
    AnalogInputPin ai10 (FEHIO::P1_0);
    AnalogInputPin ai11 (FEHIO::P1_1);
    AnalogInputPin ai12 (FEHIO::P1_2);
    AnalogInputPin ai13 (FEHIO::P1_3);
    AnalogInputPin ai14 (FEHIO::P1_4);
    AnalogInputPin ai15 (FEHIO::P1_5);
    AnalogInputPin ai16 (FEHIO::P1_6);
    AnalogInputPin ai17 (FEHIO::P1_7);
    AnalogInputPin ai20 (FEHIO::P2_0);
    AnalogInputPin ai21 (FEHIO::P2_1);
    AnalogInputPin ai22 (FEHIO::P2_2);
    AnalogInputPin ai23 (FEHIO::P2_3);
    AnalogInputPin ai24 (FEHIO::P2_4);
    AnalogInputPin ai25 (FEHIO::P2_5);
    AnalogInputPin ai26 (FEHIO::P2_6);
    AnalogInputPin ai27 (FEHIO::P2_7);
    AnalogInputPin ai30 (FEHIO::P3_0);
    AnalogInputPin ai31 (FEHIO::P3_1);
    AnalogInputPin ai32 (FEHIO::P3_2);
    AnalogInputPin ai33 (FEHIO::P3_3);
    AnalogInputPin ai34 (FEHIO::P3_4);
    AnalogInputPin ai35 (FEHIO::P3_5);
    AnalogInputPin ai36 (FEHIO::P3_6);
    AnalogInputPin ai37 (FEHIO::P3_7);

    LCD.Clear(BLACK);

    /* Create analog input menu icons */
    FEHIcon::Icon AI_T[1];
    char ai_t_label[1][20] = {"Analog Input"};
    FEHIcon::DrawIconArray(AI_T, 1, 1, 1, 201, 1, 1, ai_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon BANKS[4];
    char banks_label[4][20] = {"Bank0", "Bank1", "Bank2", "Bank3"};
    FEHIcon::DrawIconArray(BANKS, 1, 4, 40, 161, 1, 1, banks_label, SELT_C, TEXT_C);

    FEHIcon::Icon AI_B0[16];
    char AI_b0_labels[16][20] = {"P0_0", "P0_1", "P0_2", "P0_3", " ", " ", " ", " ", "P0_4",
"P0_5", "P0_6", "P0_7", " ", " ", " ", " "};

    FEHIcon::Icon AI_B1[16];
    char AI_b1_labels[16][20] = {"P1_0", "P1_1", "P1_2", "P1_3", " ", " ", " ", " ", "P1_4",
"P1_5", "P1_6", "P1_7", " ", " ", " ", " "};

    FEHIcon::Icon AI_B2[16];
    char AI_b2_labels[16][20] = {"P2_0", "P2_1", "P2_2", "P2_3", " ", " ", " ", " ", "P2_4",
"P2_5", "P2_6", "P2_7", " ", " ", " ", " "};
```

```
    FEHIcon::Icon AI_B3[16];
    char AI_b3_labels[16][20] = {"P3_0", "P3_1", "P3_2", "P3_3", " ", " ", " ", " ", "P3_4",
"P3_5", "P3_6", "P3_7", " ", " ", " ", " "};

    Buzzer.Buzz(beep_t);

    int menu=AI_MENU, bank=0, bank_i, n;
    float x, y;

    while(menu==AI_MENU)
    {
        /* Draw selected bank's ports */
        switch (bank)
        {
        case 0:
            FEHIcon::DrawIconArray(AI_B0, 4, 4, 80, 1, 1, 1, AI_b0_labels, SHOW_C, TEXT_C);
            break;
        case 1:
            FEHIcon::DrawIconArray(AI_B1, 4, 4, 80, 1, 1, 1, AI_b1_labels, SHOW_C, TEXT_C);
            break;
        case 2:
            FEHIcon::DrawIconArray(AI_B2, 4, 4, 80, 1, 1, 1, AI_b2_labels, SHOW_C, TEXT_C);
            break;
        case 3:
            FEHIcon::DrawIconArray(AI_B3, 4, 4, 80, 1, 1, 1, AI_b3_labels, SHOW_C, TEXT_C);
            break;
        }
        bank_i = bank;
        while(bank==bank_i)
        {
            /* Draw each bank's analog input values */
            if (bank==0)
            {
                AI_B0[4].ChangeLabelFloat(ai00.Value());
                AI_B0[5].ChangeLabelFloat(ai01.Value());
                AI_B0[6].ChangeLabelFloat(ai02.Value());
                AI_B0[7].ChangeLabelFloat(ai03.Value());
                AI_B0[12].ChangeLabelFloat(ai04.Value());
                AI_B0[13].ChangeLabelFloat(ai05.Value());
                AI_B0[14].ChangeLabelFloat(ai06.Value());
                AI_B0[15].ChangeLabelFloat(ai07.Value());
            }
            else if (bank==1)
            {
                AI_B1[4].ChangeLabelFloat(ai10.Value());
                AI_B1[5].ChangeLabelFloat(ai11.Value());
                AI_B1[6].ChangeLabelFloat(ai12.Value());
                AI_B1[7].ChangeLabelFloat(ai13.Value());
                AI_B1[12].ChangeLabelFloat(ai14.Value());
                AI_B1[13].ChangeLabelFloat(ai15.Value());
                AI_B1[14].ChangeLabelFloat(ai16.Value());
                AI_B1[15].ChangeLabelFloat(ai17.Value());
            }
            else if (bank==2)
            {
                AI_B2[4].ChangeLabelFloat(ai20.Value());
                AI_B2[5].ChangeLabelFloat(ai21.Value());
                AI_B2[6].ChangeLabelFloat(ai22.Value());
                AI_B2[7].ChangeLabelFloat(ai23.Value());
                AI_B2[12].ChangeLabelFloat(ai24.Value());
                AI_B2[13].ChangeLabelFloat(ai25.Value());
```

```
                AI_B2[14].ChangeLabelFloat(ai26.Value());
                AI_B2[15].ChangeLabelFloat(ai27.Value());
            }
            else if (bank==3)
            {
                AI_B3[4].ChangeLabelFloat(ai30.Value());
                AI_B3[5].ChangeLabelFloat(ai31.Value());
                AI_B3[6].ChangeLabelFloat(ai32.Value());
                AI_B3[7].ChangeLabelFloat(ai33.Value());
                AI_B3[12].ChangeLabelFloat(ai34.Value());
                AI_B3[13].ChangeLabelFloat(ai35.Value());
                AI_B3[14].ChangeLabelFloat(ai36.Value());
                AI_B3[15].ChangeLabelFloat(ai37.Value());
            }
            if (LCD.Touch(&x, &y))
            {
                /* Check to see if any of the banks icons have been touched */
                for (n=0; n<=3; n++)
                {
                    if (BANKS[n].Pressed(x, y, 0))
                    {
                        /* Change selected bank to update */
                        BANKS[n].WhilePressed(x, y);
                        BANKS[n].Deselect();
                        bank = n;
                    }
                }
                /* If back button has been touched, go to main menu */
                if (Back[0].Pressed(x, y, 0))
                {
                    Back[0].WhilePressed(x, y);
                    menu = MN_MENU;
                    bank = -1;
                }
            }
        } // end while loop
    } // end while loop
    return menu;
} // end AIMenu function

/* Accelerometer Menu */
int ACMenu()
{
    LCD.Clear(BLACK);

    /* Create accelerometer menu icons */
    FEHIcon::Icon AC_T[1];
    char ac_t_label[1][20] = {"Accelerometer"};
    FEHIcon::DrawIconArray(AC_T, 1, 1, 1, 201, 1, 1, ac_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon AC_XYZ[3];
    char ac_xyz_label[3][20] = {"X", "Y", "Z"};
    FEHIcon::DrawIconArray(AC_XYZ, 1, 3, 40, 120, 1, 1, ac_xyz_label, SHOW_C, TEXT_C);

    FEHIcon::Icon AC_VAL[3];
    char ac_val_label[3][20] = {"", "", ""};
    FEHIcon::DrawIconArray(AC_VAL, 1, 3, 120, 40, 1, 1, ac_val_label, SHOW_C, TEXT_C);
```

```
    FEHIcon::Icon AC_CAL[1];
    char ac_cal_label[1][20] = {"Calibrate"};
    FEHIcon::DrawIconArray(AC_CAL, 1, 1, 201, 2, 1, 1, ac_cal_label, SELT_C, TEXT_C);

    Buzzer.Buzz(beep_t);

    int menu=AC_MENU;
    float x, y, xo=0, yo=0, zo=0;

    while(menu==AC_MENU)
    {
        /* Update accelerometer readings taking into account an offset (calibration) */
        AC_VAL[0].ChangeLabelFloat(Accel.X()-xo);
        AC_VAL[1].ChangeLabelFloat(Accel.Y()-yo);
        AC_VAL[2].ChangeLabelFloat(Accel.Z()-zo);
        if (LCD.Touch(&x, &y))
        {
            /* Check to see if the calibrate icon has been touched */
            if (AC_CAL[0].Pressed(x, y, 0))
            {
                /* Set offsets to current accelerometer data to calibrate to zeros at that
orientation */
                AC_CAL[0].WhilePressed(x, y);
                AC_CAL[0].Deselect();
                xo = Accel.X();
                yo = Accel.Y();
                zo = Accel.Z();
            }
            /* If back button has been touched, go to main menu */
            if (Back[0].Pressed(x, y, 0))
            {
                Back[0].WhilePressed(x, y);
                menu = MN_MENU;
            }
        }
    } // end while loop
    return menu;
} // end ACMenu

/* Touch Menu */
int TOMenu()
{
    LCD.Clear(BLACK);

    /* Create touch menu icons */
    FEHIcon::Icon TO_T[1];
    char to_t_label[1][20] = {"Touch Screen"};
    FEHIcon::DrawIconArray(TO_T, 1, 1, 1, 201, 1, 1, to_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon TO[4];
    char to_label[4][20] = {"X", " ", "Y", " "};
    FEHIcon::DrawIconArray(TO, 2, 2, 40, 161, 1, 1, to_label, SHOW_C, TEXT_C);

    FEHIcon::Icon TO_SW[1];
    char to_sw_label[1][20] = {"SWITCH"};
    FEHIcon::DrawIconArray(TO_SW, 1, 1, 80, 121, 1, 1, to_sw_label, SELT_C, TEXT_C);

    Buzzer.Buzz(beep_t);
```

```
    int menu=TO_MENU, side=0;
    float x, y;

    /* Sleep briefly to avoid touching after selecting menu */
    Sleep(100);

    while(menu==TO_MENU)
    {
        if (LCD.Touch(&x, &y))
        {
            /* Update x and y of touched locations */
            TO[1].ChangeLabelFloat(x);
            TO[3].ChangeLabelFloat(y);
            if (TO_SW[0].Pressed(x, y, 0))
            {
                /* If switch is touched, switch the side of the screen with the icons to allow
access to both halfs of the touch screen for testing */
                TO_SW[0].WhilePressed(x, y);
                TO_SW[0].Deselect();
                if (side==0)
                {
                    LCD.Clear(BLACK);
                    FEHIcon::DrawIconArray(TO_T, 1, 1, 121, 80, 1, 1, to_t_label, MENU_C, TEXT_C);
                    FEHIcon::DrawIconArray(Back, 1, 1, 121, 81, 1, 260, back_label, MENU_C,
TEXT_C);
                    FEHIcon::DrawIconArray(TO, 2, 2, 161, 40, 1, 1, to_label, SHOW_C, TEXT_C);
                    FEHIcon::DrawIconArray(TO_SW, 1, 1, 201, 2, 1, 1, to_sw_label, SELT_C, TEXT_C);
                    side = 1;
                }
                else if (side==1)
                {
                    LCD.Clear(BLACK);
                    FEHIcon::DrawIconArray(TO_T, 1, 1, 1, 201, 1, 1, to_t_label, MENU_C, TEXT_C);
                    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);
                    FEHIcon::DrawIconArray(TO, 2, 2, 40, 161, 1, 1, to_label, SHOW_C, TEXT_C);
                    FEHIcon::DrawIconArray(TO_SW, 1, 1, 80, 121, 1, 1, to_sw_label, SELT_C,
TEXT_C);
                    side = 0;
                }
            }
            /* If back button has been touched, go to main menu */
            if (Back[0].Pressed(x, y, 0))
            {
                Back[0].WhilePressed(x, y);
                menu = MN_MENU;
            }
        }
    } // end while loop
    return menu;
} // end TOMenu function

/* Digital Output Menu */
int DOMenu()
{
    /* Declare ports for digital output */
    DigitalOutputPin do00 (FEHIO::P0_0);
    DigitalOutputPin do01 (FEHIO::P0_1);
    DigitalOutputPin do02 (FEHIO::P0_2);
    DigitalOutputPin do03 (FEHIO::P0_3);
    DigitalOutputPin do04 (FEHIO::P0_4);
    DigitalOutputPin do05 (FEHIO::P0_5);
```

```
DigitalOutputPin do06 (FEHIO::P0_6);
DigitalOutputPin do07 (FEHIO::P0_7);
DigitalOutputPin do10 (FEHIO::P1_0);
DigitalOutputPin do11 (FEHIO::P1_1);
DigitalOutputPin do12 (FEHIO::P1_2);
DigitalOutputPin do13 (FEHIO::P1_3);
DigitalOutputPin do14 (FEHIO::P1_4);
DigitalOutputPin do15 (FEHIO::P1_5);
DigitalOutputPin do16 (FEHIO::P1_6);
DigitalOutputPin do17 (FEHIO::P1_7);
DigitalOutputPin do20 (FEHIO::P2_0);
DigitalOutputPin do21 (FEHIO::P2_1);
DigitalOutputPin do22 (FEHIO::P2_2);
DigitalOutputPin do23 (FEHIO::P2_3);
DigitalOutputPin do24 (FEHIO::P2_4);
DigitalOutputPin do25 (FEHIO::P2_5);
DigitalOutputPin do26 (FEHIO::P2_6);
DigitalOutputPin do27 (FEHIO::P2_7);
DigitalOutputPin do30 (FEHIO::P3_0);
DigitalOutputPin do31 (FEHIO::P3_1);
DigitalOutputPin do32 (FEHIO::P3_2);
DigitalOutputPin do33 (FEHIO::P3_3);
DigitalOutputPin do34 (FEHIO::P3_4);
DigitalOutputPin do35 (FEHIO::P3_5);
DigitalOutputPin do36 (FEHIO::P3_6);
DigitalOutputPin do37 (FEHIO::P3_7);

LCD.Clear(BLACK);

/* Create digital output menu icons */
FEHIcon::Icon DO_T[1];
char do_t_label[1][20] = {"Digital Output"};
FEHIcon::DrawIconArray(DO_T, 1, 1, 1, 201, 1, 1, do_t_label, MENU_C, TEXT_C);

FEHIcon::Icon Back[1];
char back_label[1][20] = {"<-"};
FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

FEHIcon::Icon BANKS[4];
char banks_label[4][20] = {"Bank0", "Bank1", "Bank2", "Bank3"};
FEHIcon::DrawIconArray(BANKS, 1, 4, 40, 161, 1, 1, banks_label, SELT_C, TEXT_C);

FEHIcon::Icon OUT[1];
char out_label[1][20] = {"Toggle"};
FEHIcon::DrawIconArray(OUT, 1, 1, 161, 3, 1, 1, out_label, SELT_C, TEXT_C);

FEHIcon::Icon DO_B0[8];
char do_b0_labels[8][20] = {"P0_0", "P0_1", "P0_2", "P0_3", "P0_4", "P0_5", "P0_6", "P0_7"};

FEHIcon::Icon DO_B1[8];
char do_b1_labels[8][20] = {"P1_0", "P1_1", "P1_2", "P1_3", "P1_4", "P1_5", "P1_6", "P1_7"};

FEHIcon::Icon DO_B2[8];
char do_b2_labels[8][20] = {"P2_0", "P2_1", "P2_2", "P2_3", "P2_4", "P2_5", "P2_6", "P2_7"};

FEHIcon::Icon DO_B3[8];
char do_b3_labels[8][20] = {"P3_0", "P3_1", "P3_2", "P3_3", "P3_4", "P3_5", "P3_6", "P3_7"};

Buzzer.Buzz(beep_t);

int menu=DO_MENU, bank=0, bank_i, n, m;
float x, y;
```

```
    int output[4][8] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0};

    while(menu==DO_MENU)
    {
        /* Deselect all selected ports when switching banks for visibility */
        for (n=0; n<=7; n++)
        {
            if (output[0][n]==1)
                DO_B0[n].Deselect();
        }
        for (n=0; n<=7; n++)
        {
            if (output[1][n]==1)
                DO_B1[n].Deselect();
        }
        for (n=0; n<=7; n++)
        {
            if (output[2][n]==1)
                DO_B2[n].Deselect();
        }
        for (n=0; n<=7; n++)
        {
            if (output[3][n]==1)
                DO_B3[n].Deselect();
        }
        /* Draw each bank's ports and re-draw selected ports if selected earlier */
        switch (bank)
        {
        case 0:
            FEHIcon::DrawIconArray(DO_B0, 2, 4, 80, 80, 1, 1, do_b0_labels, SHOW_C, TEXT_C);
            for (n=0; n<=7; n++)
            {
                if (output[0][n]==1)
                    DO_B0[n].Select();
            }
            break;
        case 1:
            FEHIcon::DrawIconArray(DO_B1, 2, 4, 80, 80, 1, 1, do_b1_labels, SHOW_C, TEXT_C);
            for (n=0; n<=7; n++)
            {
                if (output[1][n]==1)
                    DO_B0[n].Select();
            }
            break;
        case 2:
            FEHIcon::DrawIconArray(DO_B2, 2, 4, 80, 80, 1, 1, do_b2_labels, SHOW_C, TEXT_C);
            for (n=0; n<=7; n++)
            {
                if (output[2][n]==1)
                    DO_B0[n].Select();
            }
            break;
        case 3:
            FEHIcon::DrawIconArray(DO_B3, 2, 4, 80, 80, 1, 1, do_b3_labels, SHOW_C, TEXT_C);
            for (n=0; n<=7; n++)
            {
                if (output[3][n]==1)
                    DO_B0[n].Select();
            }
            break;
```

```
        }
        bank_i = bank;
        while(bank==bank_i)
        {
            /* Check to see if each bank's port icons are touched and set the output bit for each
port to be able to toggle or not */
            if (bank==0)
            {
                if (LCD.Touch(&x, &y))
                {
                    for (m=0; m<=7; m++)
                    {
                        if (DO_B0[m].Pressed(x, y, 0))
                        {
                            DO_B0[m].WhilePressed(x, y);
                            if (output[0][m]==0)
                                output[0][m] = 1;
                            else if (output[0][m]==1)
                                output[0][m] = 0;
                        }
                    }
                }
            }
            else if (bank==1)
            {
                if (LCD.Touch(&x, &y))
                {
                    for (m=0; m<=7; m++)
                    {
                        if (DO_B1[m].Pressed(x, y, 0))
                        {
                            DO_B1[m].WhilePressed(x, y);
                            if (output[1][m]==0)
                                output[1][m] = 1;
                            else if (output[1][m]==1)
                                output[1][m] = 0;
                        }
                    }
                }
            }
            else if (bank==2)
            {
                if (LCD.Touch(&x, &y))
                {
                    for (m=0; m<=7; m++)
                    {
                        if (DO_B2[m].Pressed(x, y, 0))
                        {
                            DO_B2[m].WhilePressed(x, y);
                            if (output[2][m]==0)
                                output[2][m] = 1;
                            else if (output[2][m]==1)
                                output[2][m] = 0;
                        }
                    }
                }
            }
            else if (bank==3)
            {
                if (LCD.Touch(&x, &y))
                {
                    for (m=0; m<=7; m++)
```

```
        {
            if (DO_B3[m].Pressed(x, y, 0))
            {
                DO_B3[m].WhilePressed(x, y);
                if (output[3][m]==0)
                    output[3][m] = 1;
                else if (output[3][m]==1)
                    output[3][m] = 0;
            }
        }
    }
}
if (LCD.Touch(&x, &y))
{
    /* Check to see if any of the banks icons have been touched */
    for (n=0; n<=3; n++)
    {
        if (BANKS[n].Pressed(x, y, 0))
        {
            /* Change selected bank to update */
            BANKS[n].WhilePressed(x, y);
            BANKS[n].Deselect();
            bank = n;
        }
    }
    /* Check to see if the toggle icon has been touched */
    if (OUT[0].Pressed(x, y, 0))
    {
        OUT[0].WhilePressed(x, y);
        OUT[0].Deselect();
        /* Toggle the state of all selected output ports */
        if (output[0][0]==1) do00.Toggle();
        if (output[0][1]==1) do01.Toggle();
        if (output[0][2]==1) do02.Toggle();
        if (output[0][3]==1) do03.Toggle();
        if (output[0][4]==1) do04.Toggle();
        if (output[0][5]==1) do05.Toggle();
        if (output[0][6]==1) do06.Toggle();
        if (output[0][7]==1) do07.Toggle();
        if (output[1][0]==1) do10.Toggle();
        if (output[1][1]==1) do11.Toggle();
        if (output[1][2]==1) do12.Toggle();
        if (output[1][3]==1) do13.Toggle();
        if (output[1][4]==1) do14.Toggle();
        if (output[1][5]==1) do15.Toggle();
        if (output[1][6]==1) do16.Toggle();
        if (output[1][7]==1) do17.Toggle();
        if (output[2][0]==1) do20.Toggle();
        if (output[2][1]==1) do21.Toggle();
        if (output[2][2]==1) do22.Toggle();
        if (output[2][3]==1) do23.Toggle();
        if (output[2][4]==1) do24.Toggle();
        if (output[2][5]==1) do25.Toggle();
        if (output[2][6]==1) do26.Toggle();
        if (output[2][7]==1) do27.Toggle();
        if (output[3][0]==1) do30.Toggle();
        if (output[3][1]==1) do31.Toggle();
        if (output[3][2]==1) do32.Toggle();
        if (output[3][3]==1) do33.Toggle();
        if (output[3][4]==1) do34.Toggle();
        if (output[3][5]==1) do35.Toggle();
        if (output[3][6]==1) do36.Toggle();
```

```
                    if (output[3][7]==1) do37.Toggle();
                }
                /* If back button has been touched, go to main menu */
                if (Back[0].Pressed(x, y, 0))
                {
                    Back[0].WhilePressed(x, y);
                    menu = MN_MENU;
                    bank = -1;
                }
            }
        } // end while loop
    } // end while loop
    /* Turn off all digital output ports when leaving menu */
    do00.Write(0);
    do01.Write(0);
    do02.Write(0);
    do03.Write(0);
    do04.Write(0);
    do05.Write(0);
    do06.Write(0);
    do07.Write(0);
    do10.Write(0);
    do11.Write(0);
    do12.Write(0);
    do13.Write(0);
    do14.Write(0);
    do15.Write(0);
    do16.Write(0);
    do17.Write(0);
    do20.Write(0);
    do21.Write(0);
    do22.Write(0);
    do23.Write(0);
    do24.Write(0);
    do25.Write(0);
    do26.Write(0);
    do27.Write(0);
    do30.Write(0);
    do31.Write(0);
    do32.Write(0);
    do33.Write(0);
    do34.Write(0);
    do35.Write(0);
    do36.Write(0);
    do37.Write(0);
    return menu;
} // end DOMenu function

/* RPS Menu function */
int RPMenu()
{
    LCD.Clear(BLACK);

    /* Check to see if the Proteus has already initialized to a course region */
    if (!RPS_init)
    {
        RPS.InitializeTouchMenu();
        RPS_init = 1;
    }

    LCD.Clear(BLACK);
```

```
    /* Create RPS menu icons */
    FEHIcon::Icon RP_T[1];
    char rp_t_label[1][20] = {"RPS"};
    FEHIcon::DrawIconArray(RP_T, 1, 1, 1, 201, 1, 1, rp_t_label, MENU_C, TEXT_C);

    FEHIcon::Icon Back[1];
    char back_label[1][20] = {"<-"};
    FEHIcon::DrawIconArray(Back, 1, 1, 1, 201, 1, 260, back_label, MENU_C, TEXT_C);

    FEHIcon::Icon RP_XYH[3];
    char rp_xyh_label[3][20] = {"X", "Y", "Heading"};
    FEHIcon::DrawIconArray(RP_XYH, 1, 3, 40, 120, 1, 1, rp_xyh_label, SHOW_C, TEXT_C);

    FEHIcon::Icon RP_VAL[3];
    char rp_val_label[3][20] = {"", "", ""};
    FEHIcon::DrawIconArray(RP_VAL, 1, 3, 120, 40, 1, 1, rp_val_label, SHOW_C, TEXT_C);

    FEHIcon::Icon RP_LOG[1];
    char rp_log_label[1][20] = {"Log Data"};
    FEHIcon::DrawIconArray(RP_LOG, 1, 1, 201, 2, 1, 1, rp_log_label, SELT_C, TEXT_C);

    Buzzer.Buzz(beep_t);

    int menu=RP_MENU;
    float x, y;

    while(menu==RP_MENU)
    {
        /* Update RPS x, y, and heading values */
        RP_VAL[0].ChangeLabelFloat(RPS.X());
        RP_VAL[1].ChangeLabelFloat(RPS.Y());
        RP_VAL[2].ChangeLabelFloat(RPS.Heading());
        if (LCD.Touch(&x, &y))
        {
            /* Check to see if log data icon has been touched */
            if (RP_LOG[0].Pressed(x, y, 0))
            {
                RP_LOG[0].WhilePressed(x, y);
                RP_LOG[0].Deselect();

                /* LOG SOME STUFF USING SD CARD */
                /* To be implemented in a later version */
            }
            /* If back button has been touched, go to main menu */
            if (Back[0].Pressed(x, y, 0))
            {
                Back[0].WhilePressed(x, y);
                menu = MN_MENU;
            }
        }
    } // end while loop
    return menu;
} // end RPMenu function

/* Main function to control menu system */
void ProteusTestMain::Run()
{
    int menu=MN_MENU;
    while (true)
    {
        switch (menu)
        {
```

```
        case MN_MENU:
            menu = MNMenu();
            break;
        case DC_MENU:
            menu = DCMenu();
            break;
        case SV_MENU:
            menu = SVMenu();
            break;
        case DI_MENU:
            menu = DIMenu();
            break;
        case AI_MENU:
            menu = AIMenu();
            break;
        case AC_MENU:
            menu = ACMenu();
            break;
        case TO_MENU:
            menu = TOMenu();
            break;
        case DO_MENU:
            menu = DOMenu();
            break;
        case RP_MENU:
            menu = RPMenu();
            break;
        }
    } // end while loop
} // end ProteusTestMain function
```

WorldStateMain.h

```
#ifndef WORLDSTATEMAIN_H
#define WORLDSTATEMAIN_H

class WorldStateMain {
public:
    void Run();
};

extern WorldStateMain WorldState;

#endif // WORLDSTATEMAIN_H
```

WorldStateMain.cpp

```
// Required FEH libraries
#include <FEHLCD.h>
#include <FEHIO.h>
#include <FEHUtility.h>
#include <FEHRPS.h>

// Required custom libraries
#include "constants.h"
#include "worldstate.h"
#include "WorldStateMain.h"

// Declare object WorldState of class WorldStateMain
WorldStateMain WorldState;
```

```
/* This function is a subprogram executed from the RobotMainMenu that continuously displays the
world state of the robot.
 * Last modified: 4/2/2016
 */
void WorldStateMain::Run()
{
    RPS.InitializeTouchMenu();
    initializeLog();

    LCD.WriteLine("Waiting for middle");
    while( !button.MiddlePressed() );

    worldState(false, 0, 0, 0, 0, 0, 0);
    LCD.WriteRC("Right button - log", 5, 3);
    LCD.WriteRC("Left button - quit", 6, 3);

    while( !button.LeftPressed() ) {
        worldState(false, 0,0,0,0,0,0);

        // Write to SD card file if right button is pressed
        if ( button.RightPressed() ) {
            worldState(true, 0, 0, 0, 0, 0, 0);
            LCD.WriteRC("Logged", 5, 3);
        }
        Sleep(50);
    }
    closeLog();
    LCD.WriteRC("SD Log Closed", 6, 3);
} // end WorldStateMain::Run function
```

FinalCompetition.h

```
#ifndef FINALCOMPETITIONMAIN_H
#define FINALCOMPETITIONMAIN_H

class FinalCompetitionMain {
public:
    void Run();
private:
    int breakpoint();
};

extern FinalCompetitionMain FinalCompetition;

#endif // FINALCOMPETITIONMAIN_H
```

FinalCompetition.cpp

```
// Required FEH libraries
#include <FEHLCD.h>
#include <FEHIO.h>

// Require custom libraries
#include "constants.h"
#include "worldstate.h"
#include "start.h"
#include "toggles.h"
#include "supplies.h"
#include "ramp.h"
#include "fuelbutton.h"
#include "launchbutton.h"
```

```
#include "FinalCompetitionMain.h"

// Declare object FinalCompetition of class FinalCompetitionMain
FinalCompetitionMain FinalCompetition;

/* This function adds a breakpoints to either continue or quit from the program.
 */
int FinalCompetitionMain::breakpoint() {
    if (false) {
        LCD.WriteRC("Press left button to", 5, 3);
        LCD.WriteRC("continue...", 6, 3);
        LCD.WriteRC("or middle to quit.", 7, 3);
        while(!button.LeftPressed() && !button.MiddlePressed());
        if (button.MiddlePressed()) {
            closeLog();
            return 1;
        }
        while(!button.LeftReleased() && !button.MiddleReleased());
        return 0;
    }
    return 0;
} // end breakpoint function

/* This function is a subprogram for the Final Competition that is run from the RobotMainMenu.
 * Last Modified: 3/14/2016 JKL
 */
void FinalCompetitionMain::Run() {

    start();

    togglesBottom();

    if (FinalCompetition.breakpoint())
        return;

    pickupSupplies();

    if (FinalCompetition.breakpoint())
        return;

    tempRamp();

    if (FinalCompetition.breakpoint())
        return;

    fuelbutton();

    if (FinalCompetition.breakpoint())
        return;

    dropOffSupplies();

    if (FinalCompetition.breakpoint())
        return;

    togglesTop();

    if (FinalCompetition.breakpoint())
        return;

    mainRamp();
```

```
    if (FinalCompetition.breakpoint())
        return;

    launchButton();

    return;
} // end FinalCompetitionMain::Run function
```

FinalCompetitionFast.h

```
#ifndef FINALCOMPETITIONFASTMAIN_H
#define FINALCOMPETITIONFASTMAIN_H

class FinalCompetitionFastMain {
public:
    void Run();
private:
    int breakpoint();
};

extern FinalCompetitionFastMain FinalCompetitionFast;

#endif // FINALCOMPETITIONFASTMAIN_H
```

FinalCompetitionFast.cpp

```
// Required FEH libraries
#include <FEHLCD.h>
#include <FEHIO.h>

// Require custom libraries
#include "constants.h"
#include "worldstate.h"
#include "start.h"
#include "toggles_fast.h"
#include "supplies.h"
#include "ramp_fast.h"
#include "fuelbutton.h"
#include "launchbutton_fast.h"
#include "FinalCompetitionFastMain.h"

// Declare object FinalCompetition of class FinalCompetitionFastMain
FinalCompetitionFastMain FinalCompetitionFast;

/* This function adds a breakpoints to either continue or quit from the program.
 */
int FinalCompetitionFastMain::breakpoint() {
    if (false) {
        LCD.WriteRC("Press left button to", 5, 3);
        LCD.WriteRC("continue...", 6, 3);
        LCD.WriteRC("or middle to quit.", 7, 3);
        while(!button.LeftPressed() && !button.MiddlePressed());
        if (button.MiddlePressed()) {
            closeLog();
            return 1;
        }
        while(!button.LeftReleased() && !button.MiddleReleased());
        return 0;
    }
    return 0;
} // end breakpoint function
```

```
/* This function is a subprogram for the Final Competition that is run from the RobotMainMenu.
 * Last Modified: 4/6/2016 JKL
 */
void FinalCompetitionFastMain::Run() {

    start();

    togglesBottom_fast();

    if (FinalCompetitionFast.breakpoint())
        return;

    pickupSupplies();

    if (FinalCompetitionFast.breakpoint())
        return;

    tempRamp_fast();

    if (FinalCompetitionFast.breakpoint())
        return;

    fuelbutton();

    if (FinalCompetitionFast.breakpoint())
        return;

    dropOffSupplies();

    if (FinalCompetitionFast.breakpoint())
        return;

    togglesTop_fast();

    if (FinalCompetitionFast.breakpoint())
        return;

    mainRamp_fast();

    if (FinalCompetitionFast.breakpoint())
        return;

    launchButton_fast();

    return;
} // end FinalCompetitionMain::Run function
```

start.h

```
#ifndef START_H
#define START_H

void start();

// extern these values so that other .cpp files can use the RPS offset values
extern float RPSSuppliesYOffset;
extern float RPSFuelLightXOffset;
extern float RPSFuelLightYOffset;
extern float RPSTopRampXOffset;

#endif // START_H
```

start.cpp

```
// Required FEH libraries
#include <FEHRPS.h>
#include <FEHLCD.h>
#include <FEHUtility.h>
#include <FEHSD.h>

// Required custom libraries
#include "constants.h"
#include "worldstate.h"
#include "supplyarm.h"
#include "start.h"

// Initialize the RPS offsets as global so that it can be passed out of this .cpp file
float RPSSuppliesYOffset = 0;
float RPSFuelLightXOffset = 0;
float RPSFuelLightYOffset = 0;
float RPSTopRampXOffset = 0;

/* This function is run at the beginning of a course run to go through the launch sequence of the
robot.
 * A log is opened, RPS is initialized, RPS offsets are calibrated, CdS cell is tested,
 * servo arm is initialized, and the robot waits for the start light.
 * Last Modified: 4/2/2016 JKL
 */
void start() {

    RPS.InitializeTouchMenu();

    initializeLog();

    // Calibrate the RPS Y value for the supplies
    while( 0 != ( microswitch1.Value() + microswitch2.Value() ) ) {
        worldState(false, 0,0,0,0,0,0);
        LCD.WriteRC("RPSSuppliesY", 5, 3);
        if ( (0 == ( microswitch1.Value() + microswitch2.Value() )) && (RPS.X() > 0) ) {
            worldState(true, 0, 0, 0, 0, 0, 0);
            SD.Printf("RPSSupplies\t");
            RPSSuppliesYOffset = RPSSuppliesY - RPS.Y();
            LCD.WriteRC("Logged", 5, 3);
        }
    }

    LCD.WriteRC(RPSSuppliesYOffset, 5, 3);
    Sleep(1000);

    // Calibrate the RPS X and Y values for the fuel light
    while( 0 != ( microswitch1.Value() + microswitch2.Value() ) ) {
        worldState(false, 0,0,0,0,0,0);
        LCD.WriteRC("RPSFuelLightX and Y", 5, 3);
        if ( (0 == ( microswitch1.Value() + microswitch2.Value() )) && (RPS.X() > 0) ) {
            worldState(true, 0, 0, 0, 0, 0, 0);
            SD.Printf("RPSFuelLight\t");
            RPSFuelLightXOffset = RPSFuelLightX - RPS.X();
            RPSFuelLightYOffset = RPSFuelLightY - RPS.Y();
            LCD.WriteRC("Logged", 5, 3);
        }
    }

    LCD.WriteRC(RPSFuelLightXOffset, 5, 3);
```

```
    LCD.WriteRC(RPSFuelLightYOffset, 6, 3);
    Sleep(1000);

    // Calibrate the RPS X value for the main ramp
    while( 0 != ( microswitch1.Value() + microswitch2.Value() ) ) {
        worldState(false, 0,0,0,0,0,0);
        LCD.WriteRC("RPSTopRampX", 5, 3);
        if ( (0 == ( microswitch1.Value() + microswitch2.Value() )) && (RPS.X() > 0) ) {
            worldState(true, 0, 0, 0, 0, 0, 0);
            SD.Printf("RPSTopRamp\t");
            RPSTopRampXOffset = RPSTopRampX - RPS.X();
            LCD.WriteRC("Logged", 5, 3);
        }
    }

    LCD.WriteRC(RPSTopRampXOffset, 5, 3);
    Sleep(1000);


    // Wait for the left button to be pressed
    while( !button.LeftPressed()) {
        worldState(false,0,0,0,0,0,0);
        LCD.WriteRC("CdS Cell test", 5, 3);
        LCD.WriteRC("Press left button...", 6, 3);
        LCD.WriteRC(cdscell.Value(), 7, 10);
    }
    while(!button.LeftReleased());

    initializeArm();

    worldState(true,0,0,0,0,0,0);

    LCD.WriteRC("Press left button...", 5, 3);
    while(!button.LeftPressed());
    while(!button.LeftReleased());

    LCD.WriteRC("Waiting for light...", 6, 3);
    float startTime = TimeNow();
    while(cdscell.Value() > 1 && TimeNow() - startTime < 30) {
        LCD.WriteRC( cdscell.Value(), 7, 10);
    }

    LCD.WriteRC("Beginning run", 8, 3);
} // end start function
```

toggles.h

```
#ifndef TOGGLES_H
#define TOGGLES_H

void togglesBottom();
void togglesTop();

#endif // TOGGLES_H
```

toggles.cpp

```
// Required FEH library
#include <FEHRPS.h>

// Required custom libraries
```

```
#include "constants.h"
#include "drive.h"
#include "toggles.h"

/* This function controls the robot to press the launch sequence toggles from the lower level.
 * This function starts after the start light has been detected on the start platform
 * and ends after backing up from the toggles.
 * Last modified: 4/2/2016 JKL
 */
void togglesBottom() {

    //driveUntilTime(225, 100, 800, true);
    driveUntilTime(180, 140, 500, true);
    driveUntilRPSyRange( RPSOffStart, 30, 90, 1, 3000);
    turnUntilTime(-70, 500);
    driveUntilBump(180, 60, 3);
    driveUntilBumpTimeout(265, 100, 4, 2000);

    driveUntilTime(90, 100, 10, false);
    driveUntilTime(0, 80, 450, false);

    turnUntilRPS(270, 20, 1000);

    // Hit the blue toggle only if needed
    if (RPS.BlueSwitchDirection() == 2) {
        driveUntilBumpTimeout(270, 80, 4, 2000);
        driveUntilTime(90, 80, 40, true);
    }
} // end togglesBottom function

/* This function controls the robot to press the launch sequence toggles from the upper level.
 * This function starts after the robot drives away from the drop zone
 * and ends after backing away from the blue toggle.
 * Last modified: 4/2/2016 JKL
 */
void togglesTop() {

    // If the robot is in the dead zone, continue to drive out of it
    if (RPS.X() < 0 )
        driveUntilTime(0, 60, 100, false);

    driveUntilRPSy( RPSRedTopY, 27, 0, 2000 );

    // check the RPS X position if the red toggle needs to be pressed from the top
    if (RPS.RedSwitchDirection() == 1)
        driveUntilRPSx( RPSRedTopX, 27, 0, 1000);

    turnUntilRPS(180, 20, 1000);

    if (RPS.RedSwitchDirection() == 1) {
        driveUntilBumpTimeout(0, 60, 1, 1000);
        driveUntilTime(180, 60, 400, true);
    }

    driveUntilTime(270, 60, 400, false);

    if (RPS.WhiteSwitchDirection() == 1) {
        // check the RPS X position if the red toggle was skipped from the top
        if (RPS.RedSwitchDirection() == 2)
            driveUntilRPSx( RPSWhiteTopX, 27, 90, 1000);
        driveUntilBumpTimeout(0, 60, 1, 1000);
        driveUntilTime(180, 60, 400, true);
```

```
    }

    driveUntilTime(270, 60, 400, false);

    if (RPS.BlueSwitchDirection() == 1) {
        driveUntilRPSx( RPSBlueTopX, 27, 90, 1000);
        driveUntilBumpTimeout(0, 80, 1, 1000);
        driveUntilTime(180, 60, 400, true);
    }
} // end togglesTop function
```

toggles_fast.h

```
#ifndef TOGGLES_FAST_H
#define TOGGLES_FAST_H

void togglesBottom_fast();
void togglesTop_fast();

#endif // TOGGLES_FAST_H
```

toggles_fast.cpp

```
// Required FEH library
#include <FEHRPS.h>

// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "toggles_fast.h"

/* This function controls the robot to press the launch sequence toggles from the lower level.
 * This function starts after the start light has been detected on the start platform
 * and ends after backing up from the toggles.
 * Last modified: 4/2/2016 JKL
 */
void togglesBottom_fast() {

    driveUntilTime(225, 100, 800, true);
    driveUntilRPSyRange( RPSOffStart, 30, 90, 1, 3000);
    turnUntilTime(-70, 250);
    driveUntilBump(180, 100, 3);
    driveUntilBumpTimeout(265, 100, 4, 2000);

    driveUntilTime(90, 100, 10, false);
    driveUntilTime(0, 80, 450, false);

    turnUntilRPS(270, 20, 1000);

    if (RPS.BlueSwitchDirection() == 2) {
        driveUntilBumpTimeout(270, 80, 4, 2000);
        driveUntilTime(90, 80, 40, true);
    }
} // end togglesBottom function

/* This function controls the robot to press the launch sequence toggles from the upper level.
 * This function starts after the robot drives away from the drop zone
 * and ends after backing away from the blue toggle.
 * Last modified: 4/2/2016 JKL
 */
void togglesTop_fast() {
```

```
    if (RPS.X() < 0 )
        driveUntilTime(0, 60, 100, false);

    driveUntilRPSyRange( RPSRedTopY, 27, 0, 0.3, 2000 );

    // check the RPS X position if the red toggle needs to be pressed from the top
    if (RPS.RedSwitchDirection() == 1)
        driveUntilRPSxRange( RPSRedTopX, 27, 0, 0.4, 1000);

    turnUntilRPS(180, 20, 1000);

    if (RPS.RedSwitchDirection() == 1) {
        driveUntilBumpTimeout(0, 60, 1, 1000);
        driveUntilTime(180, 60, 400, true);
    }

    driveUntilTime(270, 60, 400, false);

    if (RPS.WhiteSwitchDirection() == 1) {
        // check the RPS X position if the red toggle was skipped from the top
        if (RPS.RedSwitchDirection() == 2)
            driveUntilRPSxRange( RPSWhiteTopX, 27, 90, 0.4, 1000);
        driveUntilBumpTimeout(0, 60, 1, 1000);
        driveUntilTime(180, 60, 400, true);
    }

    driveUntilTime(270, 60, 400, false);

    if (RPS.BlueSwitchDirection() == 1) {
        driveUntilRPSxRange( RPSBlueTopX, 27, 90, 0.4, 1000);
        driveUntilBumpTimeout(0, 80, 1, 1000);
        driveUntilTime(180, 60, 400, true);
    }
} // end togglesTop_fast function
```

supplies.h

```
#ifndef SUPPLIES_H
#define SUPPLIES_H

void pickupSupplies();
void dropOffSupplies();

#endif // SUPPLIES_H
```

supplies.cpp

```
// Required FEH libraries
#include <FEHIO.h>
#include <FEHLCD.h>

// Required custom libraries
#include "drive.h"
#include "supplyarm.h"
#include "start.h"
#include "constants.h"
#include "supplies.h"

/* This function makes the robot align with and pick up the supplies.
 * This function starts after the robot backs up from the blue toggle
```

```
 * and ends after aligning with the lower entrance if the temporary access ramp.
 * Last modified: 4/2/2016 JKL
 */
void pickupSupplies() {

    driveUntilTime(45, 90, 1450, true);
    driveUntilRPSyRange(11.9, 30, 0, 1.5, 4000); // CHECK THIS OUT!
    turnUntilTime(-70, 550);
    turnUntilRPS(0,20, 2000);

    driveUntilTime(90, 100, 600, false);
    driveUntilBump(90, 60, 2);

    driveUntilRPSy( RPSSuppliesY - RPSSuppliesYOffset, 27, 180, 8000);

    // Failsafe code
    // If the robot gets caught on the supply box and the bumpSide is not triggered,
    // drive away and retry alignment with the wall.
    if ( driveUntilBumpTimeout(90, 30, 2, 5000) == 1 ) {
        driveUntilTime( 0, 40, 500, true);
        driveUntilBump(90, 60, 2);
        driveUntilRPSy( RPSSuppliesY - RPSSuppliesYOffset, 27, 180, 8000);
        driveUntilBumpTimeout( 90, 30, 2, 2000);
    }

    lowerToPickupArm();
    raiseToPickupArm();

    driveUntilTime(90, 30, 200, true);
    driveUntilBump( 9,80,-2);

    driveUntilRPSy( RPSTempRampBottomY, 27, 270, 4000);
    turnUntilRPS(0, 20, 1000);
} // end pickupSupplies function

/* This function makes the robot drop off the supplies at the drop zone.
 * This function starts after a RPS heading check after the fuel button
 * and ends after driving away from the drop zone.
 * Last modified: 4/2/2016 JKL
 */
void dropOffSupplies() {

    driveUntilTime(88, 140, 950, false);
    driveUntilBump(90, 60, 2);

    // Drive with heading 180 until microswitch 6 is bumped
    while(microswitch6.Value()) {
        motor1.SetPercent(-20);
        motor2.SetPercent(20);
        motor3.SetPercent(20);
        motor4.SetPercent(-20);
    }
    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);

    driveUntilTime(270, 50, 150, true);

    lowerToDepositArm();
    driveUntilTime(350, 80, 550, true);
    raiseToDepositArm();
```

```
} // end dropOffSupplies function
```

ramp.h

```
#ifndef RAMP_H
#define RAMP_H

void tempRamp();
void mainRamp();

#endif // RAMP_H
```

ramp.cpp

```
// Required FEH libraries
#include <FEHRPS.h>

// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "start.h"
#include "ramp.h"

/* This function drives the robot up the temporary access ramp.
 * This function starts after the RPS heading check at the lower entrance to the ramp
 * and ends after the RPS X check with the fuel light at the top of the ramp.
 * Last modified: 4/2/2016 JKL
 */
void tempRamp() {

    driveUntilBump(90, 120, 2);
    driveUntilBump(7, 120, 1);
    driveUntilBump(0, 50, 1);
    driveUntilBump(275, 70, -1);
    driveUntilTime(270, 60, 200, false);
    driveUntilRPSxRange( RPSTempRampTopX, 25, 270, 0.3, 3000);

    // if the robot is on the ramp still, keep driving until RPS is reacquired
    while (RPS.X() == -1) {
        driveUntilTime( 270, 90, 100, false);
    }

    driveUntilRPSxRange( RPSTempRampTopX, 25, 270, 0.3, 3000);
    turnUntilRPS(0, 20, 1000);
} // end tempRamp function

/* This function drives the robot down the main ramp.
 * This function starts at the blue toggle and ends after a heading check at the bottom of the main
ramp.
 * Last modified: 4/2/2016 JKL
 */
void mainRamp() {

    driveUntilTime(270, 100, 1100, false);
    driveUntilRPSx( RPSTopRampX - RPSTopRampXOffset, 28, 0, 1000);
    driveUntilRPSyRange( RPSTopRampY, 28, 0, 0.4, 1000);
    turnUntilRPS(180, 20, 1000);

    // 358 heading to ensure that robot does not get caught on the weather station
    driveUntilTime(358, 50, 3200, true);
```

```
    driveUntilRPSyRange( RPSBottomRampY, 27, 0, 0.75, 4000);
    turnUntilRPS(180, 20, 1000);
} // end mainRamp function
```

ramp_fast.h

```
#ifndef RAMP_FAST_H
#define RAMP_FAST_H

void tempRamp_fast();
void mainRamp_fast();

#endif // RAMP_H
```

ramp_fast.cpp

```
// Required FEH libraries
#include <FEHRPS.h>

// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "start.h"
#include "ramp_fast.h"

/* This function drives the robot up the temporary access ramp.
 * This function starts after the RPS heading check at the lower entrance to the ramp
 * and ends after the RPS X check with the fuel light at the top of the ramp.
 * Last modified: 4/2/2016 JKL
 */
void tempRamp_fast() {

    driveUntilBump(90, 120, 2);
    driveUntilBump(7, 120, 1);
    driveUntilBump(275, 70, -1);
    driveUntilTime(270, 60, 200, false);
    driveUntilRPSxRange( RPSTempRampTopX, 25, 270, 0.3, 3000);
    while (RPS.X() == -1) {
        driveUntilTime( 270, 90, 150, false);
    }
    driveUntilRPSxRange( RPSTempRampTopX, 25, 270, 0.3, 3000);
    turnUntilRPS(0, 20, 1000);
} // end tempRamp function

/* This function drives the robot down the main ramp.
 * This function starts at the blue toggle and ends after a heading check at the bottom of the main
ramp.
 * Last modified: 4/2/2016 JKL
 */
void mainRamp_fast() {

    driveUntilTime(270, 130, 800, false);
    driveUntilRPSx( RPSTopRampX - RPSTopRampXOffset, 28, 0, 1000);
    driveUntilRPSyRange( RPSTopRampY, 28, 0, 0.4, 1000);
    turnUntilRPS(180, 20, 1000);
    // 358 heading to ensure that robot does not get caught on the weather station
    driveUntilTime(358, 50, 3000, true);
    driveUntilRPSyRange( RPSBottomRampY, 27, 0, 1, 4000);
    turnUntilRPS(180, 20, 1000);
} // end mainRamp function
```

fuelbutton.h

```
#ifndef FUELBUTTON_H
#define FUELBUTTON_H

void fuelbutton();

#endif // FUELBUTTON_H
```

fuelbutton.cpp

```
// Required FEH libraries
#include <FEHIO.h>
#include <FEHLCD.h>
#include <FEHUtility.h>

// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "start.h"
#include "fuelbutton.h"

/* This function presses the correct fuel button based on the color of the fuel light.
 * This starts after the robot aligns with the fuel light right after the robot leaves the
temporary access ramp
 * and ends after turning/heading check for the drive to the supplies drop zone.
 * Last modified: 4/7/2016 JKL
 */
void fuelbutton() {

    driveUntilCds(0, 60, 1, 1300);

    int redbutton = 0;

    // if a red light has been detected
    if (cdscell.Value() < 1)
        redbutton = 1;

    // if a red light has not been detected
    if (redbutton != 1) {
        driveUntilBump(90, 60, 2);
        driveUntilRPSy( RPSFuelLightY - RPSFuelLightYOffset, 25, 0, 1000);
        driveUntilTime(270, 40, 350, true);
        driveUntilRPSx( RPSFuelLightX - RPSFuelLightXOffset, 25, 90, 1000);
        turnUntilRPS(0, 20, 1000);
    }

    // if red light has been detected, press the red fuel button
    if (redbutton == 1) {
        LCD.WriteRC("Red!!!", 8, 5);
        driveUntilBump(90, 50, 2);
        driveUntilTime(10,50, 800, true);
        LCD.WriteRC("Sleep 5.1 sec", 5, 3);
        Sleep(5100);

        driveUntilTime(180, 60, 50, false);
    }
    else if (CDSBlueLow < cdscell.Value() && cdscell.Value() < CDSBlueHigh) {
       LCD.WriteRC("Blue!!!", 8, 5);
       driveUntilTime(0,50, 800,true);
```

```
       LCD.WriteRC("Sleep 5.1 sec", 5, 3);
       Sleep(5100);

       driveUntilTime(180, 60, 50, false);
    }
    else if (CDSRedLow < cdscell.Value() && cdscell.Value() < CDSRedHigh) {

        LCD.WriteRC("Red!!!", 8, 5);
        driveUntilBump(90, 50, 2);
        driveUntilTime(10,50, 800, true);

        LCD.WriteRC("Sleep 5.1 sec", 5, 3);
        Sleep(5100);

        driveUntilTime(180, 60, 50, false);
    }
    else {
        // There is a 50% probability that the light is blue.
        // Plus, the robot has failed to detect a red light twice and a blue light once, if it gets
this far.
        LCD.WriteRC("NOPE! HOPE IT'S BLUE", 5, 3);
        driveUntilBump(90, 60, 2);
        driveUntilRPSy( RPSFuelLightY - RPSFuelLightYOffset, 25, 0, 1000);
        driveUntilTime(270, 40, 350, true);
        driveUntilRPSx( RPSFuelLightX - RPSFuelLightXOffset, 25, 90, 1000);
        turnUntilRPS(0, 20, 1000);

        driveUntilTime(0,50, 800, true);
        Sleep(5100);

        driveUntilTime(180, 60, 50, false);
    }

    //driveUntilTime(210, 100, 1000, false);

    // Cool code to do cool stuff
    // Values determined empirically a day before the final competition
    driveWhileRotate(205, 180, 80, -57, 1275);

    driveUntilRPSy( RPSLongRunY, 26, 225, 1000);

    //turnUntilTime(60, 1350);
    turnUntilRPS(180, 20, 4000);
} // end fuelbutton function
```

launchbutton.h

```
#ifndef LAUNCHBUTTON_H
#define LAUNCHBUTTON_H

void launchButton();

#endif // LAUNCHBUTTON_H
```

launchbutton.cpp

```
// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "worldstate.h"
#include "launchbutton.h"
```

```
/* This function drives the robot into the final launch button.
 * This function starts after the RSP y check after the robot goes down the main ramp
 * and ends after the robot hits the launch button.
 * Last modified: 4/2/2016 JKL
 */
void launchButton() {

    driveUntilTime(90, 100, 730, true);
    turnUntilTime(-60, 360);
    turnUntilRPS(225, 20, 2000);

    // The robot is given a 2x2 inch box it can correct its position to
    driveUntilRPSxRange( RPSFinalButtonX, 30, 45, 1, 1000);
    driveUntilRPSyRange( RPSFinalButtonY, 30, 45, 1, 1000);

    turnUntilRPS(225, 20, 2000);
    closeLog();
    driveUntilTime(90, 130, 3000, true);
} // end launchButton function
```

launchbutton_fast.h

```
#ifndef LAUNCHBUTTON_FAST_H
#define LAUNCHBUTTON_FAST_H

void launchButton_fast();

#endif // LAUNCHBUTTON_H
```

launchbutton_fast.cpp

```
// Required custom libraries
#include "constants.h"
#include "drive.h"
#include "worldstate.h"
#include "launchbutton_fast.h"

/* This function drives the robot into the final launch button.
 * This function starts after the RSP y check after the robot goes down the main ramp
 * and ends after the robot hits the launch button.
 * Last modified: 4/2/2016 JKL
 */
void launchButton_fast() {

    driveUntilTime(90, 120, 600, true);
    turnUntilTime(-60, 360);
    turnUntilRPS(225, 20, 2000);
    driveUntilRPSxRange( RPSFinalButtonX, 30, 45, 1, 1000);
    driveUntilRPSyRange( RPSFinalButtonY, 30, 45, 1, 1000);
    closeLog();
    driveUntilTime(90, 130, 3000, true);
} // end launchButton function
```

constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
#include <FEHMotor.h>
#include <FEHIO.h>
#include <FEHServo.h>
```

```
#define PI 3.14159

// This information is logged in the SD log files
// VERSION is [major build].[month][day]
// TITLE is the program name
#define VERSION 3.0409
#define TITLE "FEHRobotCode"

// Define the min and max for the arm servo
#define SERVO_MIN 606
#define SERVO_MAX 2016

// Define CdS cell values
#define CDSRedLow      0.0
#define CDSRedHigh     0.8
#define CDSBlueLow     1.0
#define CDSBlueHigh    2.6

// Define all RPS values
#define RPSOffStart         24.0
#define RPSSuppliesY        10.8 // calibrate per course
#define RPSTempRampBottomY  24.0
#define RPSTempRampTopX     31.3
#define RPSFuelLightY       61.8 // calibrate per course
#define RPSFuelLightX       31.8 // calibrate per course
#define RPSLongRunY         50.0
#define RPSRedTopY          43.5
#define RPSRedTopX           1.5
#define RPSWhiteTopX         6.4
#define RPSBlueTopX         11.5
#define RPSTopRampX         27.7 // calibrate per course
#define RPSTopRampY         42.5
#define RPSBottomRampY      21.3
#define RPSFinalButtonX     18.4
#define RPSFinalButtonY     21.3

// Use of extern so that these can be used elsewhere in the code
extern FEHMotor motor1;
extern FEHMotor motor2;
extern FEHMotor motor3;
extern FEHMotor motor4;

extern FEHServo arm_servo;

extern DigitalInputPin microswitch1;
extern DigitalInputPin microswitch2;
extern DigitalInputPin microswitch3;
extern DigitalInputPin microswitch4;
extern DigitalInputPin microswitch5;
extern DigitalInputPin microswitch6;
extern DigitalInputPin microswitch7;
extern DigitalInputPin microswitch8;

extern AnalogInputPin cdscell;
extern ButtonBoard button;

#endif // CONSTANTS_H
```

constants.cpp

```
// Required FEH libraries
```

```
#include <FEHIO.h>
#include <FEHMotor.h>
#include <FEHServo.h>

// Required custom library
#include "constants.h"

// Declare the four drive motors going in a clockwise direction
FEHMotor motor1(FEHMotor::Motor0, 7.2);
FEHMotor motor2(FEHMotor::Motor1, 7.2);
FEHMotor motor3(FEHMotor::Motor2, 7.2);
FEHMotor motor4(FEHMotor::Motor3, 7.2);

// Declare the servo for the arm for the supplies
FEHServo arm_servo(FEHServo::Servo0);

// Declare the microswitches going in a clockwise direction
DigitalInputPin microswitch1(FEHIO::P1_0);
DigitalInputPin microswitch2(FEHIO::P1_2);
DigitalInputPin microswitch3(FEHIO::P1_4);
DigitalInputPin microswitch4(FEHIO::P1_6);
DigitalInputPin microswitch5(FEHIO::P2_0);
DigitalInputPin microswitch6(FEHIO::P2_2);
DigitalInputPin microswitch7(FEHIO::P2_4);
DigitalInputPin microswitch8(FEHIO::P2_6);

// Declare other sensors and the ButtonBoard
AnalogInputPin cdscell(FEHIO::P0_0);
ButtonBoard button(FEHIO::Bank3);
```

drive.h

```
#ifndef DRIVE_H
#define DRIVE_H
#include "constants.h"

void driveUntilTime(int heading, float power, int time, bool stop);
void driveWhileRotate(int startHeading, float turnDegree, float power, float turnPower, int time);
void driveUntilBump(int heading, float power, int bumpSide);
int driveUntilBumpTimeout(int heading, float power, int bumpSide, int timeout);
void driveUntilCds(int heading, float power, float cdsvalue, int timeout);
void driveUntilRPSx(float desiredX, float power, int faultHeading, int timeout);
void driveUntilRPSy(float desiredY, float power, int faultHeading, int timeout);
void driveUntilRPSxRange(float desiredX, float power, int faultHeading, float range, int timeout);
void driveUntilRPSyRange(float desiredY, float power, int faultHeading, float range, int timeout);

void turnUntilTime(float power, int time);
void turnUntilRPS(int desiredHeading, int power, int timeout);

#endif // DRIVE_H
```

drive.cpp

```
// Required FEH libraries
#include <FEHLCD.h>
#include <FEHBuzzer.h>
#include <FEHUtility.h>
#include <FEHMotor.h>
#include <FEHIO.h>
#include <FEHRPS.h>
#include <FEHSD.h>
```

```
// Required library to calculate motor ratios based on heading
#include <math.h>

// Required custom libraries
#include "constants.h"
#include "worldstate.h"
#include "drive.h"

//Declare global motor ratios
float motor1ratio;
float motor2ratio;
float motor3ratio;
float motor4ratio;

/* This function calculates the motor ratios for a given heading. The calculation is derived from
vector quantities.
 * The calculated ratios are assigned to each individual motor ratio. The ratios range from -1 to
1.
 * If the heading is a cardinal direction (N/S/E/W), the ratio is 0.707.
 * Last Modified: 3/14/2016 JKL
 */
void ratios(int heading) {

    LCD.WriteRC("Calculating motor ratios", 5,3);
    float x, y;
    x = cos( (45-heading) * PI / 180.0 );
    y = sin( (45-heading) * PI / 180.0 );
    LCD.WriteRC(x, 6, 3);
    LCD.WriteRC(y, 7, 3);
    motor1ratio = x;
    motor2ratio = -y;
    motor3ratio = -x;
    motor4ratio = y;
} // end ratios function

/* This function drives the robot with a given heading at a given power for a given time in
milleseconds.
 * If bool stop is false, the robot does not stop the motors. There will not be a pause in between
the next drive function.
 * Last Modified: 3/15/2016 JKL
 */
void driveUntilTime(int heading, float power, int time, bool stop) {

    ratios(heading);

    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilTime start");
    LCD.WriteRC("Time:", 5, 3);
    LCD.WriteRC(time, 5, 9);

    motor1.SetPercent(power * motor1ratio);
    motor2.SetPercent(power * motor2ratio);
    motor3.SetPercent(power * motor3ratio);
    motor4.SetPercent(power * motor4ratio);

    Sleep(time);
    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilTime stop");
    if (stop) {
        motor1.SetPercent(0);
        motor2.SetPercent(0);
```

```
        motor3.SetPercent(0);
        motor4.SetPercent(0);

        Sleep(100);
    }
} // end driveUntilTime function

/* This function drives the robot while rotating it.
 * startHeading: the initial heading the robot drives.
 * turnDegree: the total degrees the robot rotates during the drive
 * power: the power the robot drives at; changing this does not affect the total turn degree or
drift angle
 * turnPower: the constant power added to turn the robot; this is dependent upon turnDegree and
time
 * time: the duration the robot drives in milleseconds
 * Last modified: 4/8/2016 JKL
 */
void driveWhileRotate(int startHeading, float turnDegree, float power, float turnPower, int time) {
    // x is a fraction of the turnDegree used to add to the increment
    float x = turnDegree / (time / 50.0);
    for( int increment = 0; increment <= turnDegree; increment += x) {
        ratios(startHeading + increment);
        motor1.SetPercent(power * motor1ratio + turnPower);
        motor2.SetPercent(power * motor2ratio + turnPower);
        motor3.SetPercent(power * motor3ratio + turnPower);
        motor4.SetPercent(power * motor4ratio + turnPower);
        Sleep(50);
    }
    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);
}

/* This function drives the robot with a given heading at a given power until a side is bumped (or
not).
 * int bumpSide ranges from -4 to 4. The sides are numbered in a clockwise fashion. The front of
the robot is side 1.
 * A positive value allows for a robot drive until that side is bumped.
 * A negative value allows for the robot to ride along that side until a microswitch is released.
 * Last Modified: 3/14/2016
 */
void driveUntilBump(int heading, float power, int bumpSide) {

    ratios(heading);

    LCD.WriteRC("Bump:", 5, 3);
    LCD.WriteRC(bumpSide, 5, 9);
    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilBump start");
    motor1.SetPercent(power * motor1ratio);
    motor2.SetPercent(power * motor2ratio);
    motor3.SetPercent(power * motor3ratio);
    motor4.SetPercent(power * motor4ratio);

    // if any of the microswitches are released on the specified side
    if (bumpSide == -1)
        while( 2 != (microswitch1.Value() + microswitch2.Value()) );
    if (bumpSide == -2)
        while( 2 != (microswitch3.Value() + microswitch4.Value()) );
    if (bumpSide == -3)
        while( 2 != (microswitch5.Value() + microswitch6.Value()) );
```

```
    if (bumpSide == -4)
        while( 2 != (microswitch7.Value() + microswitch8.Value()) );

    // if both of the microswitches are depressed on the specified side
    if (bumpSide == 1)
        while( 0 != (microswitch1.Value() + microswitch2.Value()) );
    if (bumpSide == 2)
        while( 0 != (microswitch3.Value() + microswitch4.Value()) );
    if (bumpSide == 3)
        while( 0 != (microswitch5.Value() + microswitch6.Value()) );
    if (bumpSide == 4)
        while( 0 != (microswitch7.Value() + microswitch8.Value()) );

    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilBump stop");
    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);

    Sleep(100);
} // end driveUntilBump function

/* This function drives the robot with a given heading at a given power until a side is bumped (or
not).
 * int bumpSide ranges from -4 to 4. The sides are numbered in a clockwise fashion. The front of
the robot is side 1.
 * A positive value allows for a robot drive until that side is bumped.
 * A negative value allows for the robot to ride along that side until a microswitch is released.
 * int timeout limits the duration the robot drives in milleseconds
 * The return value is 0 is the function fulfills bumpSide or 1 if the function times out.
 * Last Modified: 3/31/2016
 */
int driveUntilBumpTimeout(int heading, float power, int bumpSide, int timeout) {

    ratios(heading);

    LCD.WriteRC("Bump:", 5, 3);
    LCD.WriteRC(bumpSide, 5, 9);
    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilBumpTimeout start");
    motor1.SetPercent(power * motor1ratio);
    motor2.SetPercent(power * motor2ratio);
    motor3.SetPercent(power * motor3ratio);
    motor4.SetPercent(power * motor4ratio);

    int returnValue = 0;
    int startTime = TimeNowMSec();
    // if any of the microswitches are released on the specified side
    if (bumpSide == -1)
        while( 2 != (microswitch1.Value() + microswitch2.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == -2)
        while( 2 != (microswitch3.Value() + microswitch4.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == -3)
        while( 2 != (microswitch5.Value() + microswitch6.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == -4)
        while( 2 != (microswitch7.Value() + microswitch8.Value()) && TimeNowMSec() - startTime <
timeout ) {};
```

```
    // if both of the microswitches are depressed on the specified side
    if (bumpSide == 1)
        while( 0 != (microswitch1.Value() + microswitch2.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == 2)
        while( 0 != (microswitch3.Value() + microswitch4.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == 3)
        while( 0 != (microswitch5.Value() + microswitch6.Value()) && TimeNowMSec() - startTime <
timeout ) {};
    if (bumpSide == 4)
        while( 0 != (microswitch7.Value() + microswitch8.Value()) && TimeNowMSec() - startTime <
timeout ) {};

    if (TimeNowMSec() - startTime > timeout)
        returnValue = 1;

    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilBumpTimeout stop");
    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);

    Sleep(100);
    return returnValue;
} // end driveUntilBumpTimeout function

/* This function drives the robot with a given heading at a given power until the cdsvalue is
reached.
 * If the CdS sensor reads a value less than float cdsvalue, the function stops the robot.
 * Last Modified: 3/25/2016 JKL
 */
void driveUntilCds(int heading, float power, float cdsvalue, int timeout) {

    ratios(heading);

    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilCds start");
    LCD.WriteRC("CdS:", 5, 3);
    LCD.WriteRC(cdsvalue, 5, 8);

    motor1.SetPercent(power * motor1ratio);
    motor2.SetPercent(power * motor2ratio);
    motor3.SetPercent(power * motor3ratio);
    motor4.SetPercent(power * motor4ratio);

    int startTime = TimeNowMSec();
    while( cdscell.Value() > cdsvalue && TimeNowMSec() - startTime < timeout);

    worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
    SD.Printf("driveUntilCds stop");

    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);

    Sleep(150);
} // end driveUntilCds function

/* This function corrects the x-position of the robot using RPS.
```

```
 * The robot will drive until the desiredX coordinate with a given power or until the timeout is
exceeded.
 * If the RPS is -1 or -2 (off the course or dead zone), the robot will pulse in the direction of
faultHeading.
 * Use driveUntilRPSxRange if a custom range/tolerance is needed.
 * Last Modified: 3/28/2016 JKL
 */
void driveUntilRPSx(float desiredX, float power, int faultHeading, int timeout) {

    float startTime = TimeNowMSec();
    // the tolerance for the X coordinate is +/- 0.25 inches
    while((RPS.X() < desiredX - 0.25 || RPS.X() > desiredX + 0.25) && TimeNowMSec() - startTime <
timeout) {

        // pulse toward faultHeading
        if (RPS.X() == -1 || RPS.X() == -2) {
            driveUntilTime(faultHeading, power, 500, true);
        }
        else if (RPS.X() < desiredX) {
            int heading = RPS.Heading() + 90;
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSx");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredX, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            // continuously drive the motors while the robot is greater than 2 inches away the
desiredX
            while( fabs( desiredX - RPS.X() ) >= 2);
            // pulse the motors if the robot is within 2 inches of the desiredX
            if ( fabs( desiredX - RPS.X() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);

        }
        else if (RPS.X() > desiredX) {
            int heading = RPS.Heading() - 90;
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSx");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredX, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);
```

```
            // continuously drive the motors while the robot is greater than 2 inches away the
desiredX
            while( fabs( desiredX - RPS.X() ) >= 2);
            // pulse the motors if the robot is within 2 inches of the desiredX
            if ( fabs( desiredX - RPS.X() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);
        }
    } // end while loop
} // end driveUntilRPSx function

/* This function corrects the y-position of the robot using RPS.
 * The robot will drive until the desiredY coordinate with a given power or until the timeout is
exceeded.
 * If the RPS is -1 or -2 (off the course or dead zone), the robot will pulse in the direction of
faultHeading.
 * Use driveUntilRPSyRange if a custom range/tolerance is needed.
 * Last Modified: 3/28/2016 JKL
 */
void driveUntilRPSy(float desiredY, float power, int faultHeading, int timeout) {
    float startTime = TimeNowMSec();
    // the tolerance for the Y coordinate is +/- 0.25 inches
    while((RPS.Y() < desiredY - 0.25 || RPS.Y() > desiredY + 0.25) && TimeNowMSec() - startTime <
timeout) {

        // pulse toward faultHeading
        if (RPS.Y() == -1 || RPS.Y() == -2) {
            driveUntilTime(faultHeading, power, 200, true);
        }
        else if (RPS.Y() < desiredY) {
            int heading = RPS.Heading();
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSy");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredY, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            // continuously drive the motors while the robot is greater than 2 inches away the
desiredY
            while( fabs( desiredY - RPS.Y() ) >= 2);
            // pulse the motors if the robot is within 2 inches of the desiredY
            if ( fabs( desiredY - RPS.Y() ) < 2)
                Sleep(250);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);
```

```
        }
        else if (RPS.Y() > desiredY) {
            int heading = RPS.Heading() + 180;
            heading-= 360 * (heading < 360);
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSy");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredY, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            // continuously drive the motors while the robot is greater than 2 inches away the
desiredY
            while( fabs( desiredY - RPS.Y() ) >= 2);
            // pulse the motors if the robot is within 2 inches of the desiredY
            if ( fabs( desiredY - RPS.Y() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);
        }
    } // end while loop
} // end driveUntilRPSy function

/* This function corrects the x-position of the robot using RPS.
 * The robot will drive until the desiredX coordinate with a given power or until the timeout is
exceeded.
 * If the RPS is -1 or -2 (off the course or dead zone), the robot will pulse in the direction of
faultHeading.
 * float range is the distance error from the desiredX the robot can be within
 * Last Modified: 4/2/2016 JKL
 */
void driveUntilRPSxRange(float desiredX, float power, int faultHeading, float range, int timeout) {
    float startTime = TimeNowMSec();
    while((RPS.X() < desiredX - range || RPS.X() > desiredX + range) && TimeNowMSec() - startTime <
timeout) {
        if (RPS.X() == -1 || RPS.X() == -2) {
            driveUntilTime(faultHeading, power, 500, true);
        }
        else if (RPS.X() < desiredX) {
            int heading = RPS.Heading() + 90;
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSx");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredX, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);
```

```
            // to pulse or not to pulse
            while( fabs( desiredX - RPS.X() ) >= 2);
            if ( fabs( desiredX - RPS.X() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);

        }
        else if (RPS.X() > desiredX) {
            int heading = RPS.Heading() - 90;
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSx");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredX, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            // to pulse or not to pulse
            while( fabs( desiredX - RPS.X() ) >= 2);
            if ( fabs( desiredX - RPS.X() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);
        }
    } // end while loop
} // end driveUntilRPSxRange function

/* This function corrects the y-position of the robot using RPS.
 * The robot will drive until the desiredY coordinate with a given power or until the timeout is
exceeded.
 * If the RPS is -1 or -2 (off the course or dead zone), the robot will pulse in the direction of
faultHeading.
 * float range is the distance error from the desiredY the robot can be within
 * Last Modified: 4/2/2016 JKL
 */
void driveUntilRPSyRange(float desiredY, float power, int faultHeading, float range, int timeout) {
    float startTime = TimeNowMSec();
    while((RPS.Y() < desiredY - range || RPS.Y() > desiredY + range) && TimeNowMSec() - startTime <
timeout) {
        if (RPS.Y() == -1 || RPS.Y() == -2) {
            driveUntilTime(faultHeading, power, 500, true);
        }
        else if (RPS.Y() < desiredY) {
            int heading = RPS.Heading();
            ratios(heading);
```

```
            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSy");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredY, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            // continuously drive the motors while the robot is greater than 2 inches away the
desiredY
            while( fabs( desiredY - RPS.Y() ) >= 2);
            // pulse the motors if the robot is within 2 inches of the desiredY
            if ( fabs( desiredY - RPS.Y() ) < 2)
                Sleep(250);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);

        }
        else if (RPS.Y() > desiredY) {
            int heading = RPS.Heading() + 180;
            heading-= 360 * (heading < 360);
            ratios(heading);

            worldState(true, heading, power, motor1ratio, motor2ratio, motor3ratio, motor4ratio);
            SD.Printf("driveUntilRPSy");
            LCD.WriteRC("RPS:", 5, 3);
            LCD.WriteRC(desiredY, 5, 8);

            motor1.SetPercent(power * motor1ratio);
            motor2.SetPercent(power * motor2ratio);
            motor3.SetPercent(power * motor3ratio);
            motor4.SetPercent(power * motor4ratio);

            while( fabs( desiredY - RPS.Y() ) >= 2);
            // to pulse or not to pulse
            if ( fabs( desiredY - RPS.Y() ) < 2)
                Sleep(300);

            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);

            Sleep(250);
        }
    } // end while loop
} // end driveUntilRPSyRange function

/* This function turns the robot at a given power for a given time.
 * float power (0-100): positive = clockwise rotation, negative = counter-clockwise rotation
 * Last Modified: 4/2/2016 JKL
 */
void turnUntilTime(float power, int time) {

    worldState(true, 0, power, 1, 1, 1, 1);
```

```
    SD.Printf("turnUntilTime start");
    LCD.WriteRC("Time:", 5, 3);
    LCD.WriteRC(time, 5, 9);

    motor1.SetPercent(power);
    motor2.SetPercent(power);
    motor3.SetPercent(power);
    motor4.SetPercent(power);

    Sleep(time);

    worldState(true, 0, power, 1, 1, 1, 1);
    SD.Printf("turnUntilTime stop");

    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);

    Sleep(100);

} // end turnUntilTime function

/* This function corrects the heading of the robot using RPS.
 * Power should be less than 30.
 * Last Modified: 4/2/2016 JKL
 */
void turnUntilRPS(int desiredHeading, int power, int timeout) {
    int startTime = TimeNowMSec();
    // tolarance/range is +/- 1.5 degrees
    // probably could have implemented a custom range
    while( ( fabs(desiredHeading-RPS.Heading())>1.5 && fabs(desiredHeading-RPS.Heading())<358.5 )
&& (TimeNowMSec() - startTime < timeout)) {
        worldState(true, desiredHeading, power, 1,1,1,1);
        SD.Printf("turnUntilRPS");
        LCD.WriteRC("Desired H:", 5, 3);
        LCD.WriteRC(desiredHeading, 5, 13);

        // if within +/- 10 degree of desiredHeading, start to pulse the motors
        if ( fabs(desiredHeading-RPS.Heading())<10 || fabs(desiredHeading-RPS.Heading())>350) {
            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);
            worldState(true, desiredHeading, power, 0,0,0,0);
            SD.Printf("turnUntilRPS");
            Sleep(150);
        }

        // if within +/- 1.5 degrees, stop everything and break out of the function
        // necessary to check within the loop right after the robot has stopped from the pulsing
        if (fabs(desiredHeading-RPS.Heading())<1.5 || fabs(desiredHeading-RPS.Heading())>358.5) {
            motor1.SetPercent(0);
            motor2.SetPercent(0);
            motor3.SetPercent(0);
            motor4.SetPercent(0);
            worldState(true, desiredHeading, power, 0,0,0,0);
            SD.Printf("turnUntilRPS");
            return;
        }

        if (RPS.Heading() - desiredHeading >= 0 ) {
```

```
            // turn clockwise
            if (RPS.Heading() - desiredHeading <= 180) {
                motor1.SetPercent(power);
                motor2.SetPercent(power);
                motor3.SetPercent(power);
                motor4.SetPercent(power);
            }

            // turn counter-clockwise
            else if (desiredHeading - RPS.Heading() < 180) {
                motor1.SetPercent(-power);
                motor2.SetPercent(-power);
                motor3.SetPercent(-power);
                motor4.SetPercent(-power);

            }
        }

        if (RPS.Heading() - desiredHeading < 0 ) {

            // turn counter-clockwise
            if (desiredHeading - RPS.Heading() <= 180) {
                motor1.SetPercent(-power);
                motor2.SetPercent(-power);
                motor3.SetPercent(-power);
                motor4.SetPercent(-power);
            }

            // turn clockwise
            else if (RPS.Heading() - desiredHeading < 180) {
                motor1.SetPercent(power);
                motor2.SetPercent(power);
                motor3.SetPercent(power);
                motor4.SetPercent(power);
            }
        }
    } // end while loop
    motor1.SetPercent(0);
    motor2.SetPercent(0);
    motor3.SetPercent(0);
    motor4.SetPercent(0);
    worldState(true, desiredHeading, power, 0,0,0,0);
    SD.Printf("turnUntilRPS");
} // end turnUntilRPS function
```

supplyarm.h

```
#ifndef SUPPLYARM_H
#define SUPPLYARM_H

void initializeArm();
void lowerToPickupArm();
void raiseToPickupArm();
void lowerToDepositArm();
void raiseToDepositArm();

#endif // SUPPLYARM_H
```

supplyarm.cpp

```
/* This is part of a custom header that contains the functions to operate the servo
 * for the arm to pickup and deposit the supplies.
 * Last modified: 3/21/2016 JKL
 */

// Required FEH libraries
#include <FEHServo.h>
#include <FEHUtility.h>

// Required custom libraries
#include "constants.h"
#include "worldstate.h"
#include "supplyarm.h"

/* This function is run from void start() at the beginning of the run.
 * It sets the minimum and maximum of the servo and raises it to an upright position.
 * Last modified: 3/21/2016 JKL
 */
void initializeArm() {
    arm_servo.SetMin(SERVO_MIN);
    arm_servo.SetMax(SERVO_MAX);
    arm_servo.SetDegree(175);
}

/* This function is run to lower the arm to contact the supplies.
 * Last modified: 3/21/2016 JKL
 */
void lowerToPickupArm() {
    arm_servo.SetDegree(16);
    Sleep(1000);
}

/* This function is run to pickup the supplies without dropping it.
 * A delay must be added to ensure that the supplies are picked up.
 * Last modified: 3/21/2016 JKL
 */
void raiseToPickupArm() {
    for (int degree = 18; degree <= 175; degree+= 2 ) {
        arm_servo.SetDegree(degree);
        Sleep(10);
    }
}

/* This function is run to lower the supplies into the drop zone.
 * Last modified: 4/2/2016 JKL
 */
void lowerToDepositArm() {
    arm_servo.SetDegree(63);
    Sleep(400);
}

/* This function is run to raise the arm after the supplies are deposited.
 * Last modified: 3/21/2016 JKL
 */
void raiseToDepositArm() {
    arm_servo.SetDegree(160);
}
```

worldstate.h

```
#ifndef WORLDSTATE_H
#define WORLDSTATE_H
```

```
void initializeLog();
void worldState(bool updateLog, int heading, float power, float motor1ratio, float motor2ratio,
float motor3ratio, float motor4ratio);
void closeLog();

#endif // WORLDSTATE_H
```

worldstate.cpp

```
// Required FEH libraries
#include <FEHRPS.h>
#include <FEHIO.h>
#include <FEHLCD.h>
#include <FEHUtility.h>
#include <FEHBattery.h>
#include <FEHSD.h>

// Required custom libraries
#include "constants.h"
#include "worldstate.h"

/* This function initializes the SD card for logging. It closes any open logs, opens a new log, and
prints a log header.
 * From past experiences, it is best to try to close all previous logs in case any are still open.
 * Last modified: 3/14/2016 JKL
 */
void initializeLog(){
    // Must close any remaining log files left over to prevent any SD card issues
    for (int x = 0; x<=100; x++) {
        LCD.WriteRC("Closing Log:", 7, 3);
        LCD.WriteRC(x, 7, 16);
        SD.CloseLog();
    }
    SD.OpenLog();
    SD.Printf("Title: %s - Version: %f\n", TITLE, VERSION);
    SD.Printf("Course: %c - Red: %d - White: %d - Blue: %d\n", RPS.CurrentRegionLetter(),
RPS.RedSwitchDirection(), RPS.WhiteSwitchDirection(), RPS.BlueSwitchDirection() );

SD.Printf("Time\tRPSTime\tBatVolt\tMS1\tMS2\tMS3\tMS4\tMS5\tMS6\tMS7\tMS8\tCDSCell\tMot1\tMot2\tMot
3\tMot4\tH.\tRPS_H\tRPS_X\tRPS_Y");
} // end initializeLog function

/* This function closes the SD log. This allows for the FEHSD.h library to be isolated in this
program.
 * Last Modified: 3/14/2016 JKL
 */
void closeLog() {
    SD.CloseLog();
} // end closeLog function

/* This function prints the world state to the Proteus LCD and, if requested, to a log file
 * Inputs: - bool updateLog - true writes the world state to the log file / false does not
 *         - The rest of the arguments are what is written to the log file
 * Last Modified: 3/14/2016 JKL
 */
void worldState(bool updateLog, int heading, float power, float motor1ratio, float motor2ratio,
float motor3ratio, float motor4ratio) {
    LCD.Clear(BLACK);

    LCD.DrawRectangle(2*12+1, 4*17+1, 22*12-1, 6*16-1);
    //    LCD.WriteRC("*                    *", 5, 2);
```

```
    //    LCD.WriteRC("*                    *", 6, 2);
    //    LCD.WriteRC("*                    *", 7, 2);
    //    LCD.WriteRC("*                    *", 8, 2);

    LCD.SetFontColor(WHITE);

    LCD.WriteRC(TimeNow(), 13, 5);
    LCD.WriteRC(RPS.Time(), 13, 15);

    if (updateLog)
        SD.Printf("\n%f\t%f\t", TimeNow(), RPS.Time());

    if (updateLog)
        SD.Printf("%f\t", Battery.Voltage());

    LCD.WriteRC( (int) microswitch1.Value(), 0, 2);
    LCD.WriteRC( (int) microswitch2.Value(), 0, 23);
    LCD.WriteRC( (int) microswitch3.Value(), 2, 25);
    LCD.WriteRC( (int) microswitch4.Value(), 11, 25);
    LCD.WriteRC( (int) microswitch5.Value(), 13, 23);
    LCD.WriteRC( (int) microswitch6.Value(), 13, 2);
    LCD.WriteRC( (int) microswitch7.Value(), 11, 0);
    LCD.WriteRC( (int) microswitch8.Value(), 2, 0);

    if (updateLog)
        SD.Printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t",
             (int) microswitch1.Value(), (int) microswitch2.Value(), (int) microswitch3.Value(),
(int) microswitch4.Value(),
             (int) microswitch5.Value(), (int) microswitch6.Value(), (int) microswitch7.Value(),
(int) microswitch8.Value());

    if (0.0 < cdscell.Value() && cdscell.Value() < 0.8)
        LCD.SetFontColor(RED);
    if (1.0 < cdscell.Value() && cdscell.Value() < 1.8)
        LCD.SetFontColor(BLUE);
    LCD.WriteRC( cdscell.Value(), 0, 11);

    if (updateLog)
        SD.Printf("%f\t", cdscell.Value());

    LCD.SetFontColor(WHITE);
    LCD.WriteRC("Mot1", 2, 2);
    LCD.WriteRC(power * motor1ratio, 3, 2);
    LCD.WriteRC("Mot2", 2, 20);
    LCD.WriteRC(power * motor2ratio, 3, 18);
    LCD.WriteRC("Mot3", 11, 20);
    LCD.WriteRC(power * motor3ratio, 10, 18);
    LCD.WriteRC("Mot4", 11, 2);
    LCD.WriteRC(power * motor4ratio, 10, 2);

    if (updateLog)
        SD.Printf("%f\t%f\t%f\t%f\t", power * motor1ratio, power * motor2ratio, power *
motor3ratio, power * motor4ratio);

    LCD.WriteRC("H:", 2, 10);
    LCD.WriteRC(heading, 2, 12);

    if (updateLog)
        SD.Printf("%d\t", heading);

    LCD.WriteRC("B:", 11, 9);
    LCD.WriteRC(RPS.Heading(), 11, 11);
```

```
    LCD.WriteRC("X:", 12, 4);
    LCD.WriteRC(RPS.X(), 12, 6);
    LCD.WriteRC("Y:", 12, 14);
    LCD.WriteRC(RPS.Y(), 12, 16);

    if (updateLog)
        SD.Printf("%f\t%f\t%f\t", RPS.Heading(), RPS.X(), RPS.Y());
} // end worldState function
```