Fundamentals of Engineering Robot Project

Engineering 1282H

Spring 2019

Team Galaga 7: Children of the Claw

Craig Fouts

Jodie Lawson

Aditya Vadlamani

Ellie Wong

S. Matthiesen 12:40 PM

Date Submitted: 4/22/19

Executive Summary

The purpose of this project was to complete a variety of maintenance tasks on FEH Planet's new arcades with high consistency using an autonomous robot designed and built by the team. Successful completion of this project would provide FEH Planet with an efficient and cost-effective means of maintaining their arcades. This will allow for the increase in employee morale and enhancement in customer experience which FEH Planet is expecting.

The team's robot approached each task with one of two mechanisms: (1) its large, custom, 3D-printed arm and (2) its chassis and drivetrain. The arm was responsible for carrying and depositing a token, resetting foosball score counters, and moving the claw lever to the down position. The chassis and drivetrain were used for pushing the final button and one of the two DDR buttons. The DDR task required that the robot push one of two buttons corresponding to two lights placed in front of the task. A Cadmium Sulfide (CdS) cell mounted on the underside of the robot allowed it to detect which colored light was on so that it would determine which button to approach and push with its front-right wheel. Navigating the course was accomplished using timing estimations and the Robot Positioning System (RPS) which provided the robot with information regarding its position and heading.

The robot's final design amounted to a trapezoidal prism-shaped chassis mounted on four 3.5-inch Dubro wheels. A QR code for RPS was placed on top of the chassis along with the arm mechanism. This design posed some initial drawbacks such as its weight, which prevented the robot from traveling up one of the two ramps in the arcade. Tasks such as the token, the foosball score counters, and the lever were completed with relatively high consistency, however the robot's method of pushing the DDR buttons proved inconsistent. Even in cases where the robot successfully lined up with and detected the correct light—which was also inconsistent—it often

overshot or undershot the corresponding button. These observations can be corroborated with the results of the individual and final competitions where the points scored came primarily from the token and foosball tasks.

If selected for production by FEH Planet, there are a few changes that should be made to the robot's navigation and task mechanisms to improve performance. To improve consistency in approaching each task, sensors such as bump switches and encoders should be employed to provide more reliable location awareness than timing. In terms of our primary mechanism, the servos which control the arm's horizontal and vertical movement should be mounted in such a way that reduces the strain placed on their joints.

Table of Contents

List of Figures

List of Tables

1. Introduction

When managing a chain of businesses, it is important to not be bogged down by remedial tasks, so finding quick efficient ways to complete those tasks is crucial. The arcade FEH Planet is planning on opening multiple new locations and wants to implement an autonomous robot in each store to complete simple tasks needed to open the store each day. The addition of these vehicles will boost business, morale, and the experience of guests. To start the implementation of the bots, FEH Planet contacted the Ohio Research and Development team (OSURED) to build a scale model of the layout and tasks at the arcade that need to be completed by the robot, such as resetting foosball scoring disks, depositing a coin, and lowering a claw machine lever. Over the months of February and March 2019, this team of Craig Fouts, Jodie Lawson, Aditya Vadlamani, and Ellie Wong crafted and coded a robot to compete in the 2019 FEH Planet Robot competition on Saturday, April 6, as tasked by OSURED. The purpose of this project was to complete the tasks on the FEH Planet course using an autonomous robot designed, built, and coded by the team.

Section 2 details the team's brainstorming process and the team's initial prototype and mockup design for the robot. Section 3 explains the testing of the chosen design in the required performance tests and changes the team made to the design as a result of testing. Section 4 describes the individual competition, including an analysis of the robot's performance on each run and suggested modifications for the final competition. Section 5 describes the final design, including the chassis, drivetrain, and mechanisms. This section also gives information on the team's final budget and design schedule. Section 6 details the performance of the robot at the final competition and an analysis of the results. Finally, Section 7 gives a conclusive summary of the project's results and suggests modifications if the robot is selected for production.

2. Preliminary Concepts

OSURED requires that the robot be able to repeatedly complete specific tasks in a timely manner. Figure 1 below shows a bird's-eye view of the course to be completed. Part of the course is elevated, so the robot must be able to climb one of two ramps to reach the upper level. One ramp is vinyl and has a power cord across the top; the other ramp is acrylic and has stairs at the far end of the ramp.

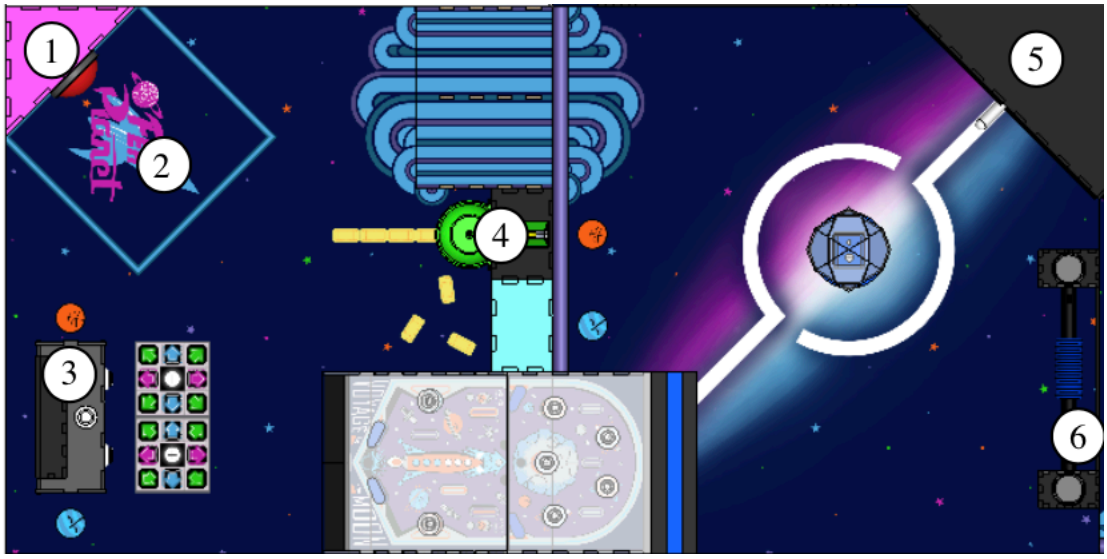

Figure 1: Overhead view of the course and tasks to be completed by the robot [1].

To begin preparing the arcade for opening the next day, the robot must start when a signal light is activated, as shown in Location 2 on Figure 1. In 2 minutes, the robot must complete a variety of tasks. For example, the robot must return the claw machine lever to the lowered position (Location 5) and reset the foosball score counter (Location 6). The robot must also reset the Dance-Dance Robot (DDR) by pressing either the red or blue software buttons (Location 3), which the robot determines based on what color light on the machine turns on. The

clean-up crew found an extra token, which the robot must either drop back into the coin slot or place in the lower coin return holder (Location 4). Once the robot has finished preparing the arcade, it must return to its initial position at the charging station, where it will press the final button that puts the robot in sleep mode (Location 1) [1].

2.1. Guidelines and Specifications

OSURED requires that the robot meet certain specifications to be eligible for the robot competition. In the robot's starting configuration, it must fit in a 9" x 9" square and may not be taller than 12". The robot's QR code, used for the Robot Positioning System, must be completely visible to the RPS technology and mounted 9'' from the ground. RPS is available to the robot on the lower level on the course but not the upper level. In order to activate RPS on the upper level of the course, the robot must press and hold the white button between the two DDR buttons. Activating RPS on the upper section is not required. OSURED also imposes restrictions on the construction of the robot. With a starting budget of $160, the team must acquire all parts through the online Company Store or have proprietary parts verified before use. The robot must be constructed entirely by the team, with no outside help [1]. Finally, no adhesives except Velcro can be used on the Proteus controller.

2.2. Development of Ideas

Before beginning the construction of the robot, the team brainstormed ideas for the robot's chassis, drivetrain, mechanisms to complete tasks, and strategies for completing the course. Each team member brainstormed individually. Then, the team discussed each member's ideas to determine the best designs. From the team's combined brainstorming list, the team screened the

ideas to obtain a smaller subset of ideas by developing a set of criteria used to compare designs to a reference design. Separate matrices were used to determine the best ideas for the chassis, drivetrain, and mechanism for completing each task. These screening matrices are available in Tables A1-A7 in Appendix A. From the top concepts determined from the screening matrices, a second decision matrix was then used to determine the "best idea" from the top concepts. The team defined additional success criteria and weighed the importance of each criterion (0-100%). Then, the team chose a new reference design, rated each design for each criterion (0-5, with 5 as the best), and added the weighted scores to determine each design's ranking. The final decision matrix is shown in Table A8 in Appendix A.

2.2.1. Initial Brainstorming

After discussing and screening individual ideas, the team determined three main designs to rank with a decision matrix, as described in the previous section. Figure 2 on the following page shows detailed sketches of each of the three main designs. The reference design, the top sketch in Figure 2, used erector set pieces for the chassis and four wheels for the drivetrain. This design used a claw mounted on a rotating as a mechanism for completing all the tasks. Design A, the middle sketch, also used four wheels and erector set pieces but included acrylic components in the chassis to improve stability. A claw on a rotating platform would be used to complete the lever, foosball, and token tasks. To press the DDR button and the final button, the robot would collide into the buttons. Finally, Design B, the bottom sketch, used erector set pieces for the chassis and three wheels for the drivetrain. An extrusion from the robot would be used to press the DDR and final buttons. This design also incorporated the claw on the rotating platform to

complete the lever and foosball tasks. To complete the token task, the robot would collide with the token holder, and the token would slide from the robot to the holder.

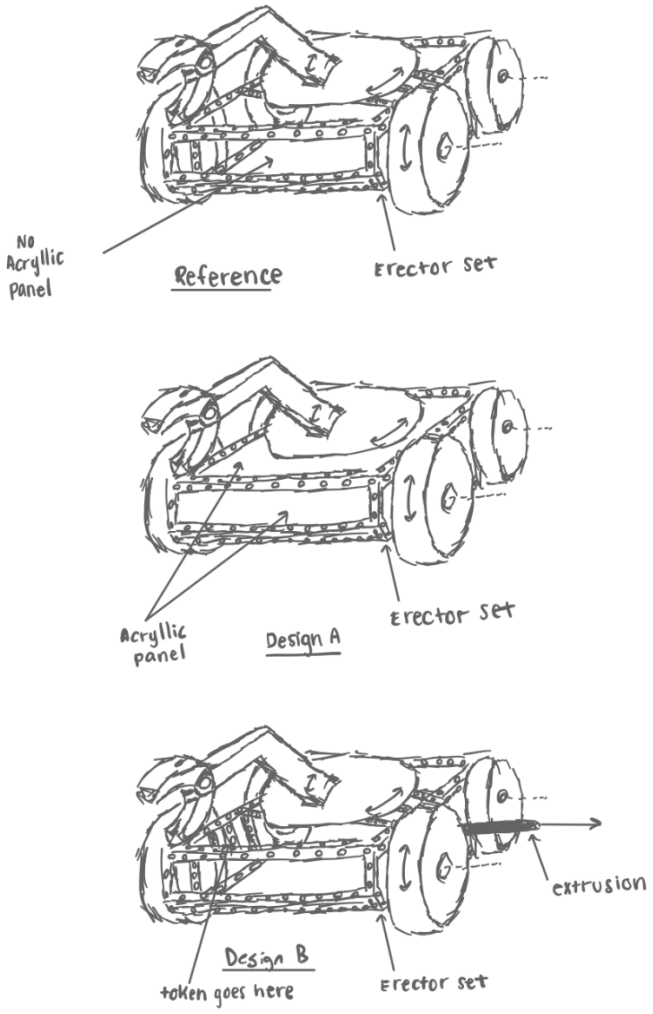

Figure 2: Initial brainstorming sketches.

The team also determined two strategies for completing the course. As seen in Figure 3 on the next page, for the first strategy, the robot would move in straight lines, counterclockwise around the course. The robot would go up the acrylic ramp with the stairs and down the vinyl ramp with the power cord bump. The robot would complete tasks in the following order: (1) DDR, (2) foosball counter, (3) claw machine, (4) token, and (5) final button.

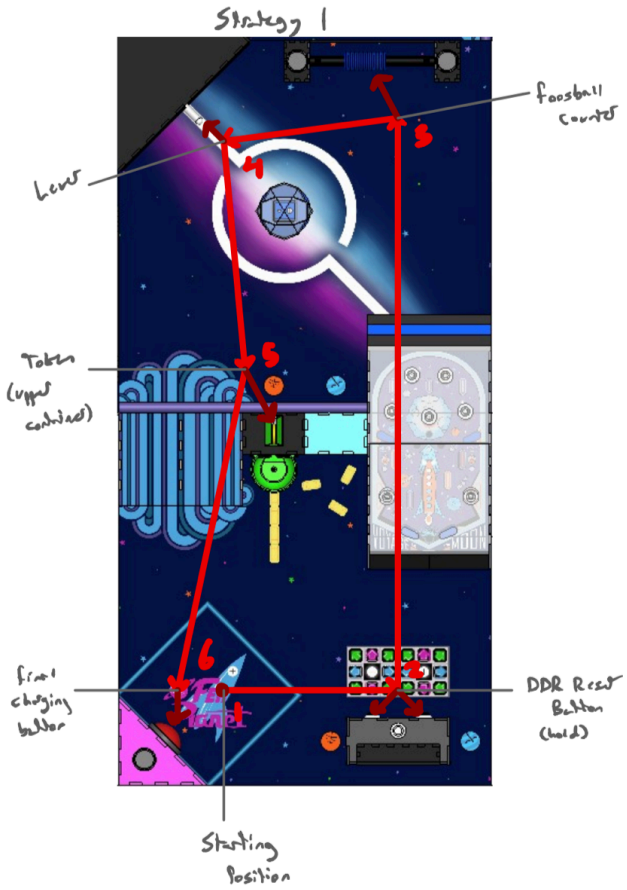

Figure 3: Strategy 1 for completing the course. The robot moves counterclockwise around the course.

As seen in Figure 4 on the following page, for the second strategy, the robot would move in straight lines clockwise around the course, going up the vinyl ramp and down the acrylic ramp. The robot would complete tasks in the following order: (1) token, (2) claw machine, (3) foosball counter, (4) DDR, and (5) final button.

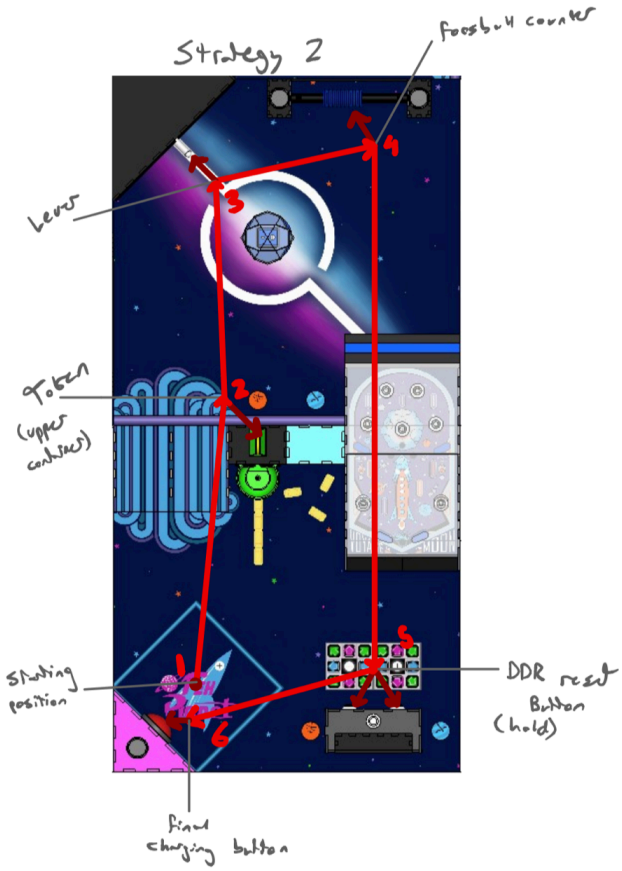

Figure 4: Strategy 2 for completing the course. The robot moves clockwise around the course.

When considering the two strategies, the team determined that the counterclockwise strategy would be the best method. One criterion the team considered was the ability of the robot to go up and down ramps and bumps. It will likely be easier for to robot to go down rather than up the vinyl ramp due to the large bump at the top. Also, when the robot travels counterclockwise, it will be easier for the robot to move the foosball disks since they will need to be moved in the direction the robot is driving.

2.2.2. Mockup and 3D-Model

The team determined that an erector set chassis with acrylic supports would be the best chassis for the robot. For completing the tasks, the team decided to use a claw on a rotating

platform. The team constructed a to-scale mockup of the robot shown in Figure 5 below. The chassis, drivetrain, and Proteus controller were modeled from cardboard. The clear plastic box on the top of the chassis represents the rotating platform, and the two forks coming out of the platform represent the claw mechanism.

Figure 5: Mockup of robot.

After constructing the mockup, the team modeled the robot in SolidWorks using the actual parts, as seen in Figure 6 on the following page.

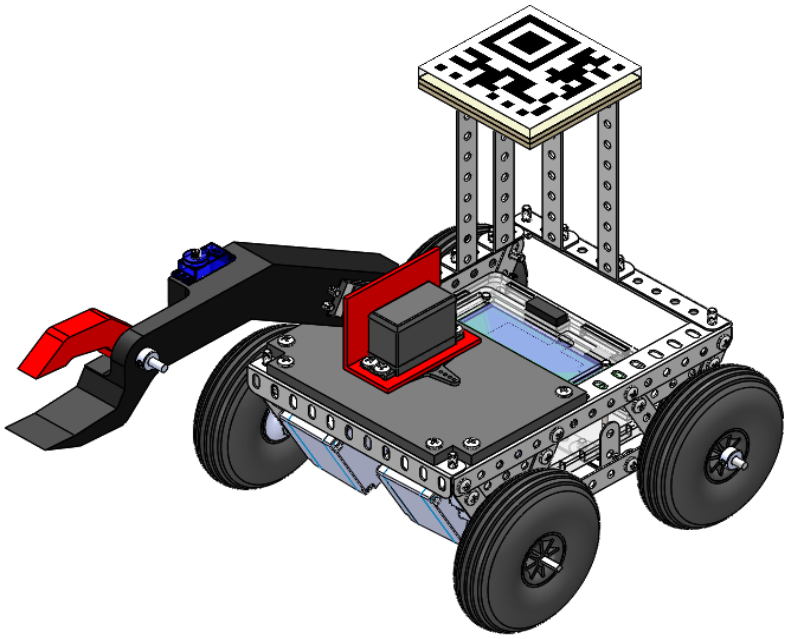

Figure 6: SolidWorks model of the robot.

Figure 6 shows the erector set frame for the robot's chassis. The sides of the robot include triangular supports that give the chassis more stability. An acrylic panel covers half of the top surface to support the rotating platform and the claw. The rotating platform is controlled by a Futaba servo. A Futaba servo also controls vertical motion of the arm. Although not shown in this model, the claw is held together by a rubber band. The top part is pulled back by a string, which is controlled by a micro servo. Inside of the robot sits the Proteus and two Vex motors controlling the two front wheels. The wheels used in this design are 3.5-inch Dubro wheels.

2.2.3. Programming Ideas

When programming the robot, the team designed the code to be modular by having specific methods for specific tasks. This would make identifying sources of errors and testing pieces of code more efficient. The main function would then call on these more general methods,

keeping the main function concise. The main function begins by orienting the robot to have the arm facing towards the back so that it would meet the starting orientation requirements for the robot. Then, using the CdS cell, the robot would detect when the starting light turns red and proceed to driving. If a starting light isn't detected within 10 seconds, the robot will start to drive anyway as a default mechanism. Once the robot starts moving, the arm will rotate to its driving position, which is vertical to reduce strain on the servo motors powering the arm. This will be the arm's state whenever it is driving to any part of the course. To complete the course, the main function calls a `driveTo` method and a `complete` method, which both take in an element of an enumeration consisting of the different task names as a parameter to determine what task to navigate to and compete. Within the `driveTo` method, the robot travels by time and then uses RPS data to cross-check the distance traveled or angle turned. The `complete` method specifies claw movements or position adjustments for completing tasks. The team's initial code representation is available in Figure G1 in Appendix G. Figure G2 in Appendix G shows the team's final code representation.

3. Analysis, Testing, and Refinements

After constructing the proposed design, the team identified numerous issues with both the robot's hardware and software. Analysis of the robot's speed and torque requirements allowed the teach to select an ideal drivetrain. The team's experience with performance tests provided important insights on strengths and flaws of the robot's design, and class explorations provided resources and information the team could use to improve the robot's performance.

3.1. Drivetrain Calculations

To determine the robot's drivetrain, the team determined what motors fit the speed and torque requirements of the robot. After estimating the distance traveled by the robot and the desired time to complete the course, the required linear speed was calculated using Equation E1 in Appendix E. Using Equation E2 in Appendix E, the linear speed was converted to a shaft rotational speed. The team determined the rotational speed to be 17.46 revolutions per minute. The torque requirement was determined by calculating the required motor force, shown in Equation E3, which was used to determine overall torque as shown in Equation E4. The team calculated that each motor required a torque of 27.5 oz-in. Figure B1 in Appendix B shows the speed-torque specifications for each available motor. The area bounded by the motor's line and the team's robot's torque and speed requirements gave the usable performance area for that motor. As seen in the graph, all motors satisfied the robot's speed and torque requirements. However, the three best candidates with the highest usable performance areas were the Acroname, Igwan, and Vex. When determining what motors to use, the team considered cost and reliability. Because the Igwan motors were expensive and the Acroname motors were unreliable, the team decided to use Vex motors.

3.2. Explorations

Each exploration provided concepts and information that affected the design of the robot. Through these explorations, the team was able to experiment with sensors, motors, navigation strategies, and RPS. For each exploration, the team used a Crayola Bot.

3.2.1. Exploration 1: Sensors and Motors

In the first part of the sensors and motors exploration, the team recorded CdS cell values for a red light, a blue light, and no light, filtering each reading with a blue filter, a green filter, a red filter, a yellow filter, or no filter. The red filter provided the greatest disparity between red light and blue light values, so the team decided to use it along with the CdS cell on the robot to make it easier to determine which DDR color was displayed. The second part of the exploration involved using the `TouchCalibrate` function to set minimum and maximum values for a servo motor. This method proved useful when calibrating the servos that control the arm on the robot, allowing the team to precisely set upper and lower bounds for the arm's motion. For the final part of the exploration, the team programmed a robot to use four microswitches to navigate a short maze. While the team members were successful in completing this task, they decided against implementing the switches on their final robot. Since the robot's large wheels extended pass the chassis, there was no location on the robot where a switch could be placed without building an extrusion on the robot.

3.2.2. Exploration 2: Line Following and Shaft Encoding

For the first part of the line following and shaft encoding exploration, the team programmed a robot to use an array of analog optosensors to navigate both straight and curved paths by following strips of colored tape. The team found that since each optosensor differed slightly in its sensitivity, so each had to have its own threshold values for detecting the tape. The team decided not to include optosensors on their final robot, as there seemed to be few opportunities on the course where line following would be beneficial. In the second part of the exploration, the team programmed a robot to use shaft encoders to travel certain distances and turn specified amounts.

Given the inconsistencies faced with the shaft encoders, the team decided against including them on the final robot, opting for timing the distances instead. The team also did not have the sufficient funds to purchase Vex encoders and was forced to use other navigation methods.

3.2.3. Exploration 3: RPS and Data Logging

In the first part of the RPS and data logging exploration, the team programmed a robot to use a combination of shaft encoders and RPS to navigate between four points on the robot course. Some problems the team encountered, however, such as the imprecision of RPS and its delay, indicated that RPS could not be used as the sole method of navigation for the final robot. The team decided to time the distances and turns and then check with RPS afterward to ensure that the robot traveled the correct distance or turned to the correct heading within some threshold. If required, the team programmed the robot to pulse to the position or heading before continuing with the course. Since the team planned on using the RPS methods learned in this exploration for the competition, the team needed to construct a mount for the RPS QR code. For the second part of the exploration, the team programmed a robot to record and log its X and Y coordinates using RPS. This functionality proved instrumental in troubleshooting the final robot, as the team has been able to follow what the robot "sees" through RPS to determine where the robot thought it was when it performed some action incorrectly.

3.3. Performance Tests

Each performance test required the robot to complete a specific task on the course. Preparation for each of the performance tests required the team to analyze the current design and make refinements based on testing the robot on the course.

3.3.1. Performance Test 1

The first performance test required the robot to navigate to the upper level of the course and toggle the claw machine lever. The design met all of the needs to complete the test. The robot was able to align with the bottom of the ramp, drive up the ramp, turn to align with the lever, and hit the lever down with its claw mechanism. Once the robot was able to meet the minimum requirements, the code was edited to complete the bonus task of returning to the bottom level. The team received a score of 23/20. The testing log for Performance Test 1 is shown in Table C1 in Appendix C.

When preparing the robot for Performance Test 1, the team encountered issues with the robot's tires. The front wheels, the ones with the motors, did not have enough traction to drive up either of the two ramps. Adversely, the back tires had too much traction. When the robot turned, the back wheels would not turn with the front wheels. Also, when the robot attempted to drive straight, it would slightly veer to the left due to an issue with the back-left wheel. When the back-left wheel turned, the tire rubbed slightly against the adjacent erector set pieces that made the chassis. This friction caused the left wheel to turn less than the right wheel, resulting in a curved path. The team easily resolved the issues with the wheels by adding materials to either decrease or increase the wheels' traction. Since the front wheels did not have enough traction, the team wrapped a rubber band around each wheel, allowing the wheel to more easily grip the surface of the course. The team then placed electrical tape around the back wheels to reduce the traction of these wheels. After these changes, the robot was able to drive up the ramps as well as turn without the back wheels dragging. The modifications to the wheels are shown in Figure 7 on the following page.

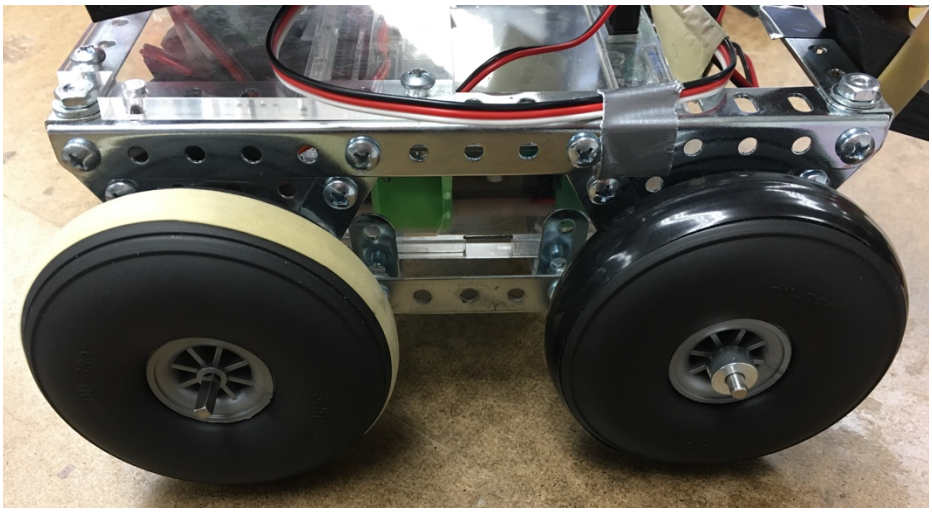

Figure 7: Modification to wheels to improve driving.

Besides issues with the drivetrain, the team faced issues with the physical size of the robot. The team worried that the robot would not meet the dimension specifications. Although the chassis itself fit in specified dimensions, when the claw mechanism was extended, it exceeded the height restriction of 12’’. To account for this, the team programmed the robot arm to have an initial position facing the back-left corner of the robot, as far down as possible. The team accomplished this by setting initial angles for the servos controlling the rotating platform and the vertical motion of the arm. In this starting configuration, the robot fit within the specified dimensions.

Furthermore, the team was concerned about the weight of the robot and how this would affect the ability of the robot to successfully drive up the ramps. After testing driving up the left ramp, which the team planned in the original strategy of completing the course, the team determined that going up this ramp was not feasible as the robot was too large and heavy to get over the bump on top of this ramp. As a result, the team decided to alter the strategy for completing the course by programming the robot to use the right ramp instead of the left ramp with the bump.

The robot still had difficulties getting up the ramp due to its weight, but increasing the motor power slightly when going up the ramp resolved the issue.

Performance Test 1 also brought the issue of navigation to the team's attention. The team originally planned to navigate the course using analog optosensors. The team mounted a sensor on the side of each motor so that the sensors faced the inside of the tires. The team printed black and white pinwheels and glued them to the inside of the tires. Ideally, the sensors would detect changes in reflectivity between the white and black segments of the pinwheel and use this information to determine the number of wheel rotations and thus distance traveled, similar to shaft encoders. The pinwheel set up is shown in Figure 8 below. However, this navigation proved to be inconsistent and unsuccessful. The sensors often did not detect the correct number of wheel rotations or encoder counts, which led the robot to travel incorrect distances.

Figure 8: Pinwheel setup for navigation with analog optosensors.

After many failed tests using the optosensors, the team switched to using navigation by time for this performance test. The robot was programmed to drive and turn for a certain amount

of time, based on how far it needed to drive or what angle it needed to turn. These tests helped the team realize the need for better navigation techniques, like RPS, in future tests.

3.3.2. Performance Test 2

For Performance Test 2, the robot had to read and display the color of the DDR light, press the corresponding DDR button, and hold the button for 5 seconds. All of the specifications for the test were completed. The robot started with the red light, drove to the DDR light at an angle, sensed the correct DDR light, pressed the correct DDR button, and held the button for 5 seconds. The robot did not complete the bonus task of going to the upper level and touching the foosball counters, so the team received a score of 20/20. The testing log for Performance Test 2 is shown in Table C2 in Appendix C.

For the DDR button task, the team originally planned on aligning the robot to directly face DDR and press the correct button using the claw mechanism. However, the height of the claw could not reach and press the buttons effectively in this configuration. Instead, the team determined that the wheels would be a better method of hitting and holding the buttons. The DDR tests also led to many changes to the RPS and timing code. Since this test was the first one in which the team experimented with RPS, the team had to design and construct a mount on which to place the QR code. The mount was built out of erector set pieces with an acrylic plate on top the place the code. The team added a few pieces of cardboard under the acrylic to ensure that the QR code was at the desired height (9’’) from the ground. The QR code mount is can be seen in Figure 9 on the next page. The team’s official QR is shown in Figure B3 in Appendix B.

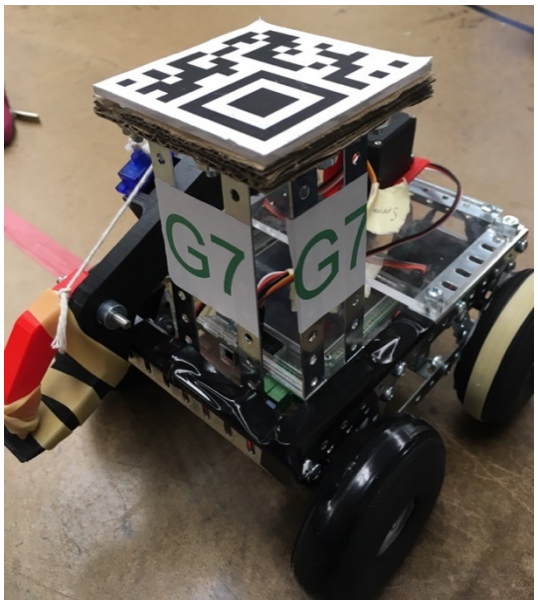

Figure 9: QR code mount constructed out of erector set pieces, acrylic, and cardboard.

The initial tests mainly consisted of checking distances, angles, and motor power using RPS values. Multiple tests helped find the right distances and angles for the CdS cell to align with the DDR light consistently each run. The team struggled to effectively use RPS data to navigate to the DDR task, so the team once again navigated solely using time.

In Performance Test 1, the robot was able to successfully start with the starting light because the team was able to place the robot in the starting box so that the CdS cell was directly over the light. However, when testing the code to complete DDR, the robot did not consistently detect the correct color of the DDR light because the CdS cell was not always directly over the DDR light. At the edge of the light, the light values detected by the CdS cell were inconsistent with the threshold values originally determined by the team, resulting in incorrect color detection. To account for this issue in the robot's design, the team added a paper funnel around the CdS cell. The addition of the CdS cell funnel redefined the CdS cell thresholds and helped the robot

detect the blue and red lights consistently, even when the cell was not perfectly aligned with the lights.

3.3.3. Performance Test 3

To complete Performance Test 3, the robot had to place the token in the slot located on the upper level. The robot met all of the minimum specifications for the test. The robot started with the red light, drove forward and turned to align with the token slot, and used the claw to drop the token through the slot. However, the bonus task of flipping the lever down was not attempted, and the team received 20/20. The testing log for Performance Test 3 is shown in Table C3 in Appendix C.

One of the first issues the team encountered regarding the claw mechanism was its ability to hold the token. The claw material lacked enough friction to hold the token for an extended period of time without dropping it. The token, having been printed using the same material as the claw, would slip from the claw's grip whenever the claw moved. As a result, the team wrapped rubber bands around the separate parts of the claw to grip the token more easily, as shown in Figure 10 on the following page. After testing multiple times with this design change, the team concluded that this was a successful refinement.

Figure 10: Addition of rubber bands to the claw for token task.

The team also discovered that the servo used to open and close the claw was incapable of performing this task with the arrangement of the claw at the time. The rubber band keeping the claw closed proved to be stronger than the servo motor, hindering the claw from operating at all. To fix this issue, the team moved the rubber band farther back on the claw to reduce the torque and make it easier for the servo to lift the claw. In testing, the also team noticed that the claw wobbled back and forth significantly when the robot was driving or when the claw itself was rotating to a new position. The team was concerned that this instability was putting too much strain on the servo controlling the claw. To resolve this issue, the team added additional screws to secure the claw to the acrylic panel.

For the token task, the robot was able to reach the token slot from the bottom level, so the strategy originally planned of going to the top level to deposit the token was not needed. However, testing of the token deposit still provoked changes in the navigation and claw methods. The navigation to the claw used RPS to align with the token slot, therefore it took many attempts to find the right distances to drive forward and turn towards the slot. Adjustments to the claw angle

were also made throughout the testing process to ensure that the token was located directly above the slot when the claw opened. The team also decided to implement a shaking motion through the arm servo when the claw opened. This ensured that the token fell when the claw opened.

Navigation for this performance test relied on RPS. The initial idea surrounding the use of RPS with the `driveStraight` method would be that the robot would use RPS to calculate the distance it has traveled and then the robot would stop once it was within a certain threshold of the desired distance. When the code was implemented, the team noticed that the robot was overshooting the intended distance each run. To debug the issue, the code was modified to log the distance. After viewing the data, the team noticed that by the time the robot got the data it needed from RPS and calculated the distance traveled, it had exceeded the predetermined threshold. After identifying the issue, the team decided that the number of calls made to RPS should be decreased. This then posed the concern of when the robot should crosscheck with RPS and what it should use for navigation seeing that encoders and bump switches were not viable.

The team decided that the robot would drive based on a calculated estimate for time and then crosscheck with RPS to see that the robot had indeed traveled that amount. This decision then led to the creation of the `pulseDriveStraight` (and `pulseTurn`) method. The `pulseDriveStraight` method took in a pointer to the robot's coordinates before it started driving. It then calculated the distance between that point and the robot's current point. Based on the distance, if the robot overshot or undershot the predetermined threshold, the robot would pulse backward or forwards until the robot was within the desired threshold. These changes to the `driveStraight` method made navigating with RPS significantly more reliable.

3.3.4. Performance Test 4

Performance Test 4 required the robot to move the foosball disks to their final position and press the final button. The robot did not meet all of the specifications for the performance test. It was able to navigate to the upper level and move the foosball disks from its initial position to the final position, and the disks stayed in the final position. The team missed 5 points because the robot did not press the final button. The team also did not attempt to get the bonus points by pressing a DDR button, so the team received a score of 15/20. The testing logs for Performance Test 4 are shown in Tables C4-C5 in Appendix C.

The robot completed the task by aligning itself around the middle of the foosball task and grabbing the foosball disks with the claw. Then the arm moved to the left to move the disks. The team realized that when the robot tried to move the disks to the left, the arm got stuck when it moved the disks to the middle of the rod. To resolve this, the team programmed the robot to move backward slightly after gripping the to give the arm a full range of motion. The robot also had difficulties gripping the foosball disks in the first place, so the team added more rubber bands to the top and bottom parts of the claw. Once these two issues were fixed, the robot moved the foosball disks consistently as expected. The claw was consistent in moving the disks even when the robot was located in slightly different positions relative to the task. As a result, the team determined that the mechanism for this task was a strength of the current design and would require minimal modifications for future competitions.

Despite the success with the foosball task, the robot faced difficulties with navigation. The robot consistently got up the ramp, but the position of the robot after it drove off the ramp on the upper level varied between runs. This caused issues when the robot tried to navigate around the obstacle and go down the left ramp to hit the final button. The robot would sometimes run into the

obstacle, stopping it from going down the ramp. To prevent this problem, the team decided to have the robot knock the obstacle out of the way using its arm, which would eliminate any possibility of the robot getting stuck on the upper level. Testing this strategy revealed that hitting the obstacle was an effective solution to the navigation issue. Although the robot did not successfully press the final button for the performance test, the team discovered that the robot could get over the bump to drive down the left ramp. This confirmed the team's initial strategy of using this ramp to return to the lower level to hit the final button.

4. Individual Competition

The individual robot competition took place in class at 12:40 pm on Friday, March 29, 2019, in Hitchcock Hall Room 208. The robot was required to have a QR code and run the RPS software. The order in which teams competed was determined randomly. Before the first run, the instructional team ensured that the robot met the size specifications and was clearly labeled with the team designation on at least 2 sides. For the competition, the team was given three attempts. The robot was given two minutes for each run. If the robot started before the starting light, it was considered a false start. Two false starts in one round resulted in a disqualification for that run. The course and DDR color run were chosen by the instructional team. The second run was fully randomized, and the final run was chosen by the team. Once at the course, the team was given one minute to prepare for the run, which included placing the token on the robot and lining up the robot in its starting position. The seeding for the final competition was determined by each team's best score (primary + secondary points), average score, and best time.

4.1. Robot Performance

The first run took place on Course H with the blue DDR light. On the first attempt, the robot started before the starting light, which was a false start. When the team attempted that run a second time, the robot scored 34 primary points and 20 secondary points for a total of 54 points. The robot successfully placed the token in the top slot, and the token stayed in the bowl. For the foosball task, the robot moved the scoring disks to the final position, and the disks stayed in that position. The team unsuccessfully attempted to complete the DDR task. The CdS cell was not close enough to the light. Although it did not detect the blue light, it still navigated to the blue button as this was the team's default. The robot was not close enough to the button to press it. After completing the foosball task, the robot unsuccessfully attempted to knock out the obstacle with its arm, which blocked it from reaching the lever task or the final button.

For the second run, the team was on Course G with the blue DDR light. The team modified the code in preparation for this run. The angle of the robot arm was lowered after the foosball task to ensure that the robot would hit the obstacle out of the way. The team also decreased the distance that the robot travels backward after the token task so that the robot was closer to the DDR buttons. Once again, the robot scored all primary and secondary points associated with the token and foosball tasks. With the new code, the robot knocked the obstacle out of the way, toggled the claw machine lever, and left the lever in its lowered position. When the robot attempted to navigate to the final button, it drove into the platform for the token task, preventing it from getting down the ramp. Once again, the robot was still too far from the blue DDR button. Overall, the team scored 51 primary points and 20 secondary points for a total of 71 points.

For the final run, the team chose to be on Course G with the blue DDR light once again. To improve the robot's performance with DDR, the team decreased the distance traveled after the

token task once again. The team also modified the code after completing the lever task so that the robot would drive down the ramp to the final button successfully. The robot scored all primary and secondary points for the token task. Unfortunately, the robot traveled too far back after the token task such that it was too close to the DDR structure. The robot pressed the red button but was unable to drive up the ramp to complete tasks on the upper level. The team only scored 29 primary points and 8 secondary points for a total score of 37 points.

The team's second run was used for final competition seeding. This run had a time of 1 minute and 14 seconds, and the average score across the three runs was 54 points. For the final competition, the team was seeded 5th out of 8 teams in the class.

4.2. Analysis and Modifications

Although the robot performed better in the individual competition than the team expected it to, the team noticed aspects of the robot's design that needed significant improvements. The robot did not complete the DDR task successfully in any of the three runs. Even during test runs, the team's strategy for the DDR task was inconsistent. The team discussed changing the strategy altogether by approaching the DDR light while driving parallel to the DDR task. In the third run, the team did not get to test to new code for navigating to the final button since the robot did not reach the upper level. For the final competition, the team needed to modify the code to ensure that the robot could get down the left ramp without driving into the token task. Certain aspects of the robot's designs performed extremely well and did not require modifications for the final competition, such as the methods of completing the token, foosball, and lever tasks. The testing logs for the individual competition are available in Tables C6-C9 in Appendix C.

5. Final Design

Over the course of this project, the team made many changes to the robot's hardware and software. The following sections describe the robot's final design and code as well as the team's final budget and schedule.

5.1. Features and Strengths

The results of performance tests, explorations, and the individual competition resulted in changes in the team's initial robot design, including changes in the chassis, drivetrain, mechanisms, and methods. The robot's design for the final competition can be seen in Figure 11 below.

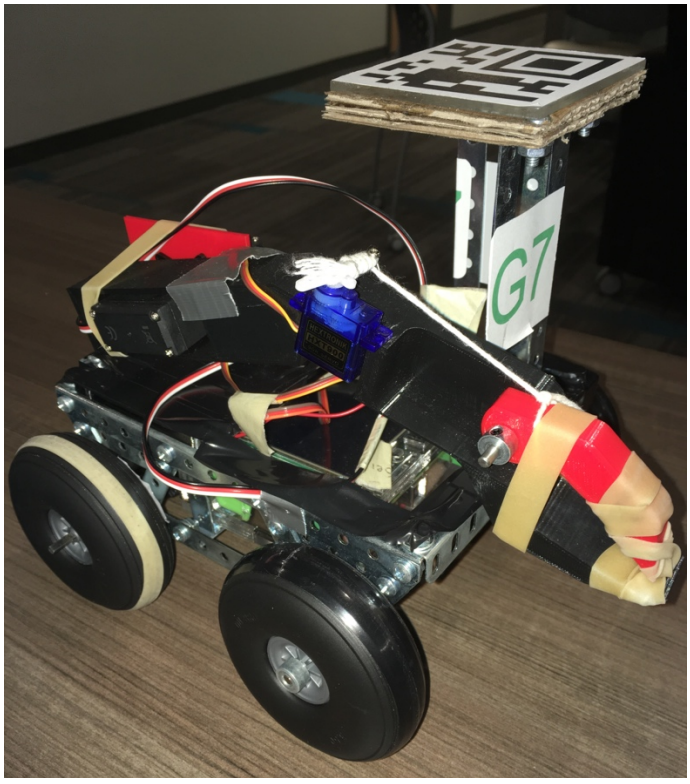

Figure 11: Final robot design.

5.1.1. Chassis and Drivetrain

The chassis of the robot consisted of an erector set frame. The Proteus controller rested inside the robot on erector set pieces along the base of the chassis. To improve the stability of the chassis, the team used erector set pieces to make triangular supports and placed them behind the wheels. The team laser-cut an acrylic panel and attached it to the top of the erector set frame. The robot's arm mechanism was screwed into this panel. The panel was covered in black electrical tape to prevent glare off the acrylic that could interfere with RPS, as seen in Figure 12 below.

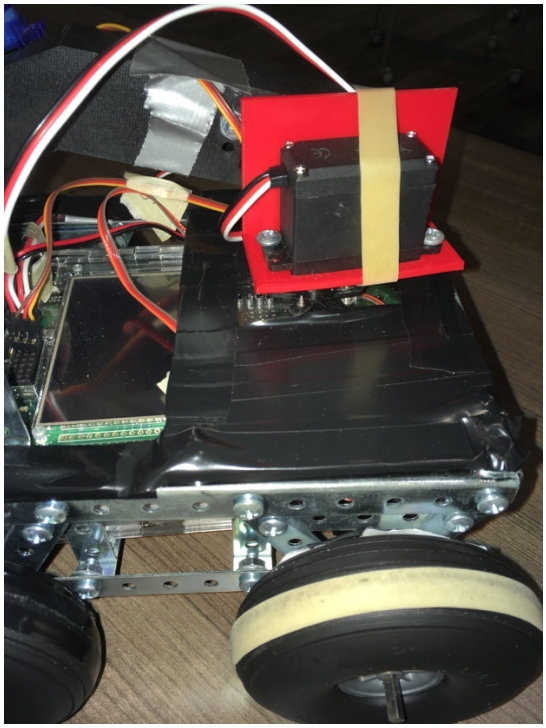

Figure 12: Acrylic panel used to mount robot arm. Panel is covered in electrical tape.

For the drivetrain, the team used four 3.5'' Dubro wheels. The robot required such large wheels to ensure that the chassis was high enough off the ground that it would be able to clear the bump on the left ramp on the course. Rubber bands were wrapped around the front wheels to

increase traction, and electric tape was wrapped around the back wheels to decrease traction. The front two wheels were powered by Vex motors, which were placed on the inside wheels as shown in Figure 13 below. The black-and-white paper pinwheels shown in figure were used with analog optosensors as an initial navigation technique as described previously, but the team did not use the pinwheels past Performance Test 1. Since the robot did not use shaft encoders or bump switches, the robot navigated the course using time and checked its location and angle using RPS.

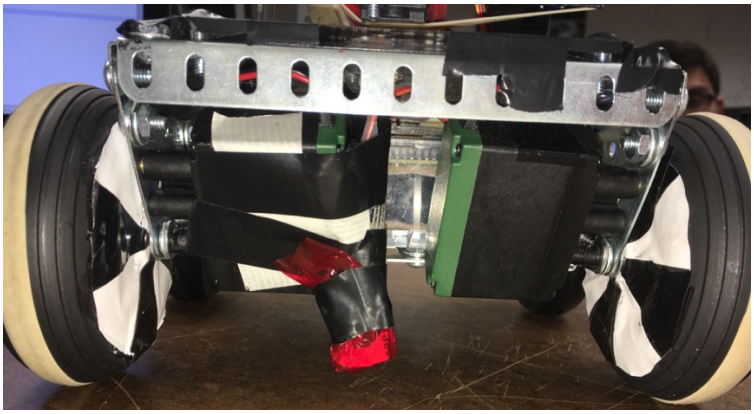

Figure 13: Set up of Vex motors.

5.1.2. Electrical System and Sensors

The robot's electrical system setup with the Proteus controller can be seen in Figure B2 in Appendix B. The robot used two Vex motors for the front two wheels. A CdS cell was taped to the inward-facing surface of the right motor as seen in Figure 13 above, and the sensor was wired to one of the input/output pins on the Proteus. Three servo motors were used to control the motion of the robot's arm and claw: One Futaba servo controlled the movement of the rotating platform, a second Futaba servo controlled the vertical motion of the arm, and a micro servo controlled the opening motion of the claw. The wires for the CdS cell and servo motors were tagged with

descriptive names to make rewiring the Proteus easier. Tables B1-B3 in Appendix B summarize the electrical system for the drivetrain motors, servo motors, and input/output pins.

5.1.3. Mechanisms and Methods

The robot used the same mechanism—a claw—to complete the token, foosball, and lever tasks. The mechanism consisted of a rotating platform, an arm that could move up and down, and a claw that could open and close, as shown in Figure 14 below. Each of the three aspects of the mechanism was controlled by its own servo motor.

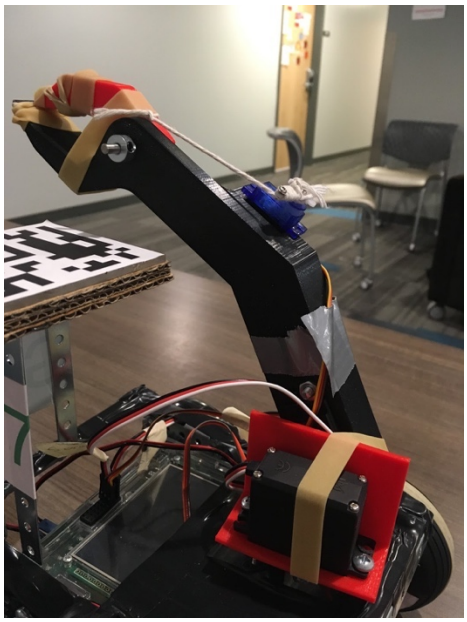

Figure 14: Robot claw and arm set up.

As shown in Figure 15 on the following page, the robot held the token vertically in its claw. The token was placed slightly offset in the claw to ensure that it would fall the correct direction when the claw released it. The robot completed the token task from the lower level by positioning

the claw above the token slot and opening the claw to release the token, as shown in Figure 16 below.

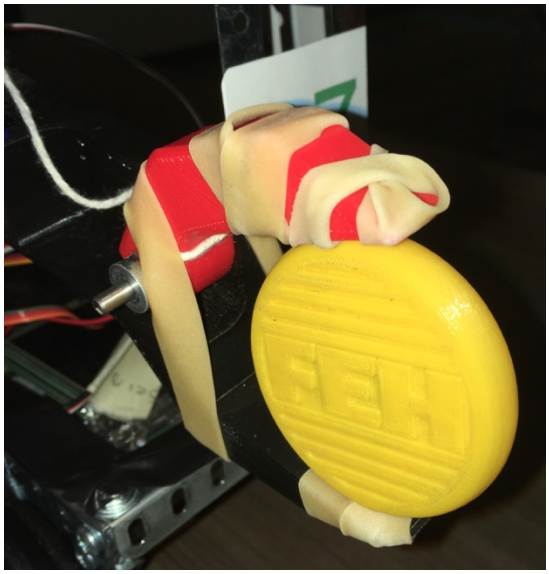

Figure 15: Positioning of the token in the claw.

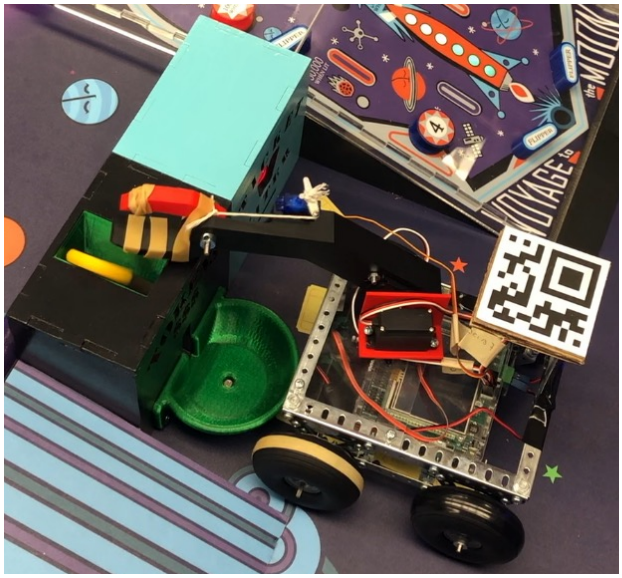

Figure 16: Method of releasing the token into the slot.

To complete DDR, the robot positioned the CdS cell over the left DDR light and collected data from the sensor to determine what color the light was. The robot displayed the color detected on the Proteus screen and, depending on what color it was, drove to the corresponding button at an angle and pressed it with its front right wheel. During testing, the robot could not hold the button down for 5 seconds when it hit the button straight on, so the robot had to approach the button from an angle. Figure 17 below shows the robot detecting a red color and driving to press the red button.

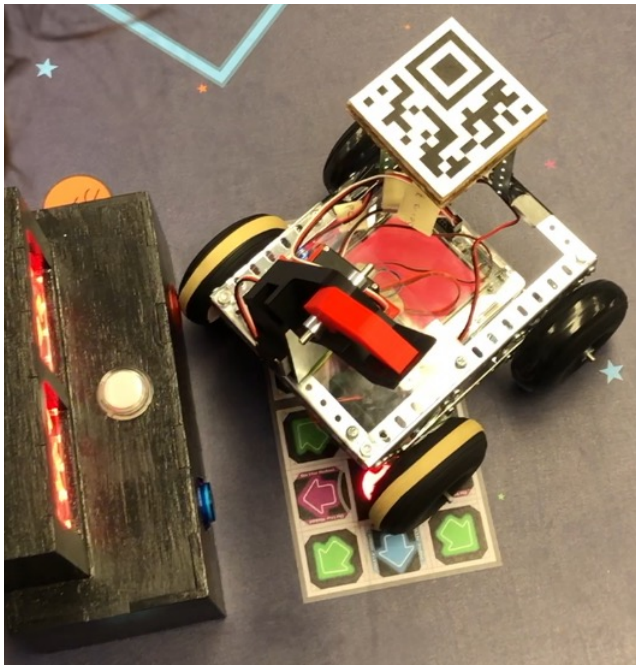

Figure 17: Method of completing DDR task using the robot's wheels.

The robot also used its claw to complete the foosball tasks. Once aligned near the center of the rod, the robot extended its arm to the disks' final position and opened its claw. The robot arm then moved as far right as possible, and the claw closed to grip the foosball disks. The robot had to go to the left first rather than grabbing the disks initially to ensure that the claw always reached the disks. If the robot arm moved to the right initially, the team found that it would sometimes get stuck to the right of the foosball structure and would be unable to complete the task. Figure 18

below shows the claw gripping the disks in their initial position. While gripping the disks, the arm moved to the left to move the disks to their final position. When moving the disks, the robot drove backwards slightly to give the arm a full range of motion.

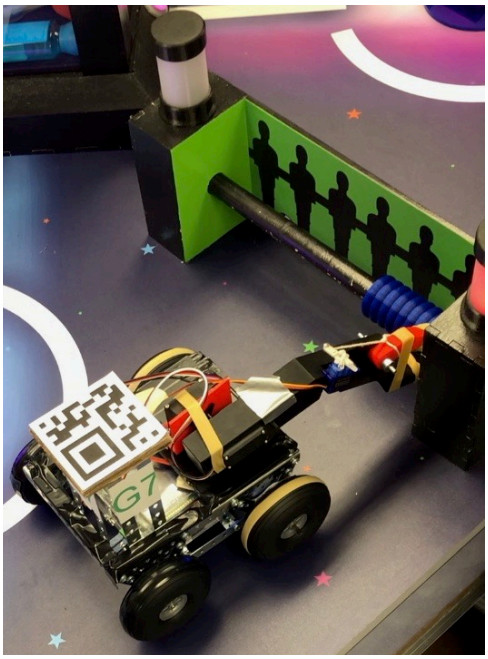

Figure 18: Method for moving the foosball disks.

To complete the lever task, the robot lifted its arm and then lowered it to push down the lever. To account for variations in the robot's position relative to the lever task, the arm moved up and down repeatedly at different angles on the platform to ensure that the robot toggled the lever. Figure 19 on the following page below shows the robot completing the lever task.

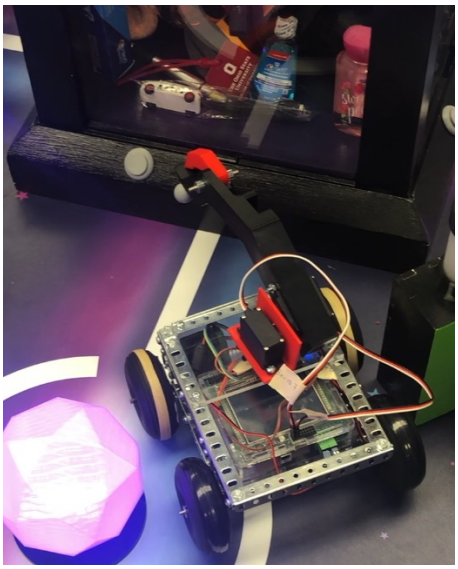

Figure 19: Method of completing the lever task.

To press the final button, the robot drove down the left ramp and drove straight until its front right wheel collided with the final button, as depicted in Figure 20 below.

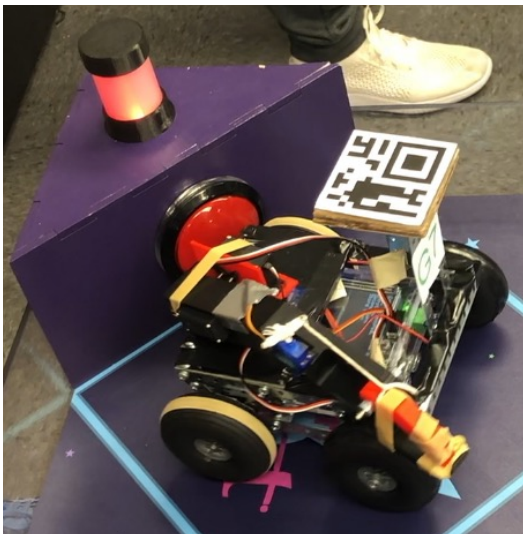

Figure 20: Method of pressing the final button.

5.2. Final Code

The final code for the robot followed the modular design that the team had hoped for. The different actions that the robot took to perform the different tasks on the course were packaged into two different void functions, `driveTo(Task)` and `complete(Task)`. These functions took in the name of the task which would then determine the set of actions the robot will perform. The reason for consolidating the driving and completion of the different tasks into two methods instead of having a method for each task was purely a design decision and could easily be modified to meet the preferences of those working on the code. The set of actions in the two methods utilized both existing methods, such as `setDegree()` in the `FEHServo` library, and newly created void methods such as `driveStraight()` and `turn()`. By having the code broken into multiple methods, the code can be modified accordingly to what the group wants, especially when changes to the robot's design and the team's strategy change. The final code is shown in Appendix G. Figures G2-G3 in Appendix G also show a flowchart representation of the final code.

While completing the coding portion of the project, the major issue that occurred was dealing with the inconsistencies with RPS. The robot relied heavily on RPS to navigate the course, so there were times when testing on a specific course would cause the robot to behave erratically. It was later determined to be an issue with the QR code, and many of the RPS issues were resolved when the team received a replacement QR code.

5.3. Final Budget

The team spent $151.08 of the allocated $160 for a final budget of $8.92. The chart in Figure 21 on the following page shows the breakdown of the team's budget.

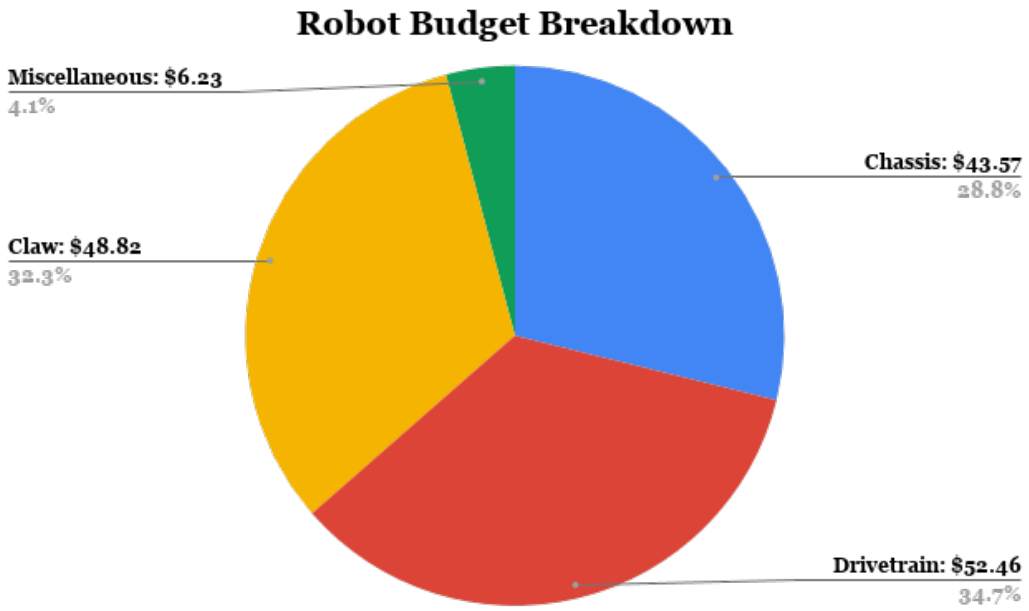

Figure 21: Breakdown of the team's budget.

The budget is split into four categories: (1) Drivetrain, which includes the wheels, axles, and the Vex motors; (2) Chassis, which includes the erector set pieces and acrylic plate; (3) Claw, which includes 3D-printing fees and servo motors; and (4) Miscellaneous, which includes sensors, rubber bands, and solder. As seen in the diagram, the highest fraction of the budget was spent on the drivetrain, followed by the claw, chassis, and miscellaneous parts.

The change in the budget over time was also tracked as purchases were made. Figure 22 on the following page shows the progression of the team's budget over time. The budget decreased the most from the first purchase made which included most of the erector set pieces and motors to build the chassis and drivetrain. Other major decreases in the budget occurred in the first weeks of the project as the 3D printed parts and laser cut acrylic were designed. Past mid-February, the changes in the budget were minor. The team made small miscellaneous purchases like rubber

bands for the final competition and construction fees for the cardboard that was used in constructing the QR code mount.

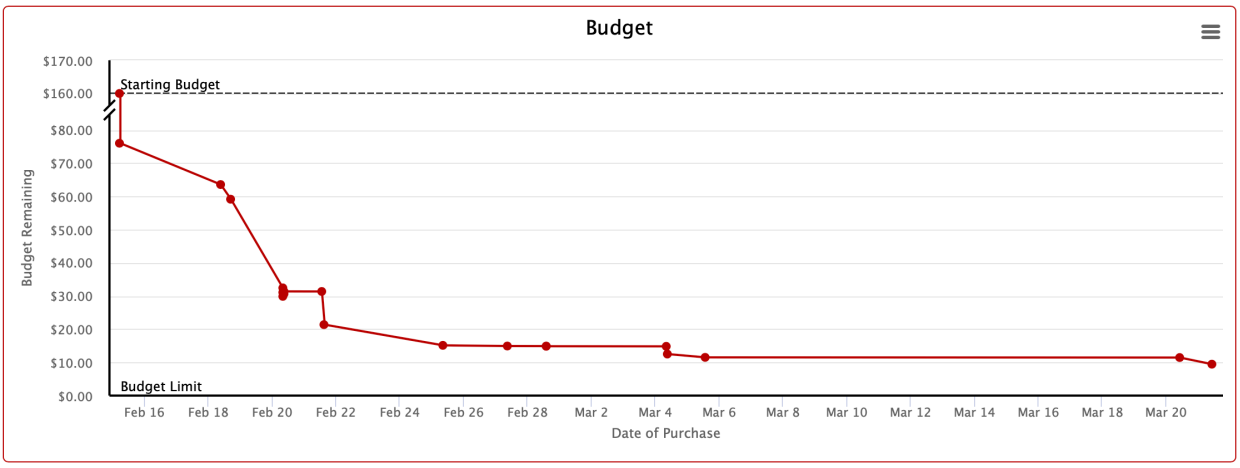

Figure 22: Budget changes over the course of the project.

5.4. Final Schedule

The team planned out each task to be completed for the project and assigned each group member to a primary (P), secondary (S), or contributor (C) role for each task. The schedule, as shown in Table D1 in Appendix D, gave estimated and actual start and finish dates for tasks involving brainstorming, robot construction, performance test preparation, documentation. Some tasks were completed earlier than expected like the building and finalization of the robot and claw. These items had to be finished to complete certain performance tests; therefore, the overall design and claw were completed early on in the project. Other tasks were put off until later like the website and code for performance tests. The website was not started until mid-March when there was sufficient information to add to the pages. Code for the performance tests was often not started until a couple of days before the test. Originally, the team thought writing the code for tests would only take a few hours, but the actual testing and revision process made this time closer to five

hours for each test. Overall, all assignments were completed by the due date, but some assignments took longer than anticipated, and starting other assignments was delayed because of this.

Table D2 in Appendix D shows the weekly time breakdown for time spent on documentation, project management, code, building, testing, and CAD. The team spent a total of 408 hours on the project, with 61 hours spent on documentation, 29 hours spent on project management, 40 hours spent on CAD, 55.5 hours spent on coding, 184.75 hours spent on testing, and 37.5 hours spent on building. Figures D1-D7 in Appendix D summarize the percentage of hours that each team member spent on each of these categories.

6. Final Competition

The final robot competition occurred on Saturday, April 6, 2019, in the Davis Gymnasium at the Recreation and Physical Activity Center (RPAC). The round-robin tournament began at 12:00 pm. There were 3 rounds of round robin play. The head-to-head competition began at 4:15 pm. For each run in the single-elimination tournament, the robot that earned the most points was declared the winner. In case of a tie, the faster robot won. If multiple robots tied in points and time, the round was re-run with only those robots. For each run, the team was given 1 minute to set up the robot. After the start light was activated, the robot was given 2 minutes to complete the run. Four robots competed simultaneously each run. The testing log for preparation for the final competition is shown in Figure C10-C13 in Appendix C.

6.1. Team Strategy

The team attempted to score as many primary and secondary points as possible. After starting on the starting light, the robot navigated to the token task from the lower level and released

the coin into the top token slot using its claw. Then, the robot traveled to the DDR task, used the CdS cell to determine if the light was red or blue, and pressed the corresponding button with its front right wheel. The robot then turned and drove up the right ramp until it aligned with the center of the foosball task, where it used its claw to move the disks from their starting position to their final position. On its way to the lever task, the robot used its arm to knock the center obstacle out of the way. The robot's large size made navigation around the obstacle difficult, so hitting it out of the way ensured that the robot would not get stuck on the upper level. With the obstacle out of the way, the robot pressed down the lever using the top part of its arm. Finally, the robot drove down the right ramp with the bump and traveled straight until either its front right wheel or its chassis hit the final button. For navigation, the robot relied on time and used RPS data when available to check its location and heading, pulsing the motors until the position was within a desired threshold.

6.2. Results and Analysis

The first three rounds the robot competed in were a part of the round robin competition. The first run was on Course G with the red DDR light. This run was successful except for the token deposit. The token fell off to the left side of the claw as opposed to the right, so the token did not fall into the slot. The robot scored 73 total points the first run. The second round robin run was on Course C with a blue DDR light. This run was a complete failure and the robot only scored 8 points from starting with the red light. The robot missed the token deposit and was a couple of inches off from every task. The third round robin run was the best run of the day. The robot scored 92 points with a time of 1:35. The robot successfully deposited the token for the first time, and the only task the robot did not complete was the final button. The robot got stuck on the black wall after

completing the lever, so the team decided to use the kill switch and end the run. Overall, no changes in the code were made after any of the round robin runs. There were no obvious changes that would make the robot perform more consistently. The token issues from the first two rounds were fixed by changing the way the token was placed in the claw at the beginning of the run. The token was placed at more of angle to ensure the it would fall to the right when the claw opened.

Next, the robot competed in the head to head competition. The seeding for the bracket was determined from the robot's performance in the individual competition. In the round of 64, the robot completed the token and foosball tasks, but the robot got stuck on the dodecahedron obstacle when attempting to navigate to the lever. The robot scored 54 points which was more than the other three robots in this round, meaning the robot moved onto the next round. In the round of 16, the robot scored 73 points, missing the DDR and part of the foosball points. The robot finished this round with a time of 1:27. The other robots in this round performed better, and the robot did not move onto the final four.

Overall, the performance of the robot in the final competition was better than expected. The robot had one bad run, one good run, and three average runs that were comparable to the individual competition. The main problems that the robot faced at the final competition were inconsistency in the token task and not completing the course in a competitive time. However, the robot did complete the tasks in less than two minutes and showed improvement in completing the DDR task from the individual competition. The robot was also awarded second place for innovation out of all robots that competed.

7. Summary and Conclusions

The task assigned to the team was to create a robot that could complete the tasks asked by the FEH Planet for their new locations. Through a long period of brainstorming the team came up with multiple design concepts and performed extensive analysis on the different parts which would constitute the final robot, looking at the pros and cons of each idea. Along with brainstorming design ideas, the team members underwent three exploration tasks which allowed them to familiarize themselves with the different tools at their disposal for completing this task. The brainstorming and extensive analysis yielded the final robot design which featured a sturdy chassis made from erector set pieces, a four-wheel drivetrain in which both wheels were powered by Vex motors, and a 3D-printed arm-like mechanism which was able to complete many of the tasks. The immense amount of testing completed also lead to well-documented modularized code. While the work put into the robot project was very intensive, the team was able to produce a robot following FEH Planet's timeline.

Over the course of the project, the team underwent four performance tests, the individual competition, and the final competition. During the first three performance tests, the robot met or exceeded the expectations of the team as it completed the required tasks as well as some additional tasks. In the fourth performance test, however, the robot failed to travel back down to the lower level resulting in an imperfect run. During the individual competition, the robot was able to complete the token and lever task with a high success rate, but was unable to complete the lever with such proficiency and unable to complete the DDR and final button at all. Overall, the robot scored an average of 54 points over the three runs leading the team to seeded 5th out of the 8 teams in the class section for the final competition. For the final competition, the robot competed in five rounds, 3 round-robin rounds and 2 head-to-head. In two of the round-robin rounds, the robot

exceeded in performance compared to the individual competition. The robot also was successful in moving past the round of 64 into the round of 16 during the head-to-head competition. The team was awarded 2nd place in innovation for the arm mechanism that the robot used. Overall, the team believes the robot was successful in completing the token and foosball task with very high accuracy which FEH Planet could incorporate into the final robot that they will be using for their arcades.

If FEH Planet selected the team's robot for production, the team recommends some changes to the robot. First, the team recommends utilizing as many of the navigational tools provided instead of heavily relying on one or two techniques. For example, the team would have found a way to more navigation techniques such as microswitches to help keep the robot parallel with a side of the course and the use of encoders to be able to drive fixed distances more accurately. The latter would also require better budgeting early in the project. The second thing the team would change would be to redesign the arm-like mechanism to cause less strain on the servos controlling the arm. Finally, the team would have been more proactive in charging the Proteus ensuring that it was at 100% so that they could eliminate any small discrepancies between runs. Future work that can be done on the robot would be to have it perform different tasks by changing the code to see if the robot could be modified or if the robot could adapt to new tasks that FEH Planet may want it to do in the future.

References

[1] Robot Scenario. 2019, February 24. www.carmen.osu.edu.

[2] Drivetrain Calculations. 2019, March 31. www.carmen.osu.edu.

APPENDIX A

Brainstorming

Table A1: Decision matrix used to determine the chassis.

Chassis					
Success Criteria	**Reference (Acrylic)**	**Design A (Laminated wood)**	**Design B (PVC)**	**Design C (Aluminum)**	**Design D (Erector Set Piece)**
Modular ability	0	0	0	-	+
Maintenance	0	-	-	0	+
Weight	0	-	0	-	+
Durability	0	+	0	+	-
Sum +	0	1	0	1	3
Sum 0	5	0	3	1	0
Sum -	0	2	1	2	1
Net Score	0	-1	-1	-1	2
Continue?	Combine	No	No	No	Combine

Table A2: Decision matrix used to determine the drivetrain.

Drivetrain				
Success Criteria	**Reference (Threads)**	**Design A (Four wheels)**	**Design B (Three wheels)**	**Design C (Two wheels)**
Ability to go uphill	0	+	0	-
Maintenance	0	+	+	+
Weight	0	0	+	+
Durability	0	+	+	+
Ability to go over bumps	0	+	+	-
Sum +	0	4	4	3
Sum 0	0	0	0	0
Sum -	0	0	0	2
Net Score	0	4	4	1
Continue?	Combine	Combine	Combine	No

Table A3: Decision matrix used to determine the mechanism to complete the task with the foosball disks.

Foosball Scoring Disks				
Success Criteria	**Reference (Claw Method 1)**	**Design A (Claw Method 2)**	**Design B (Claw Method 3)**	**Design C (Claw Method 4)**
Time	0	+	0	-
Consistency	0	0	0	0
Realistic	0	0	0	0
Ease of Attachment	0	+	0	0
Sum +	0	2	0	0
Sum 0	0	2	4	3
Sum -	0	0	0	1
Net Score	0	2	0	-1
Continue?	Combine	Combine	Combine	No

Table A4: Decision matrix used to determine the mechanism to complete the task with the DDR buttons.

DDR Buttons				
Success Criteria	**Reference (Claw)**	**Design A (Projectiles)**	**Design B (Collision)**	**Design C (Extruded pointy object)**
Time	0	0	+	+
Consistency	0	-	0	0
Realistic	0	-	+	0
Ease of Attachment	0	-	+	+
Sum +	0	0	3	2
Sum 0	0	1	2	2
Sum -	0	3	0	0
Net Score	0	-3	3	2
Continue?	Combine	No	Yes	Combine

Table A5: Decision matrix used to determine the mechanism to complete the task with the claw machine lever.

Lever				
Success Criteria	**Reference (Claw)**	**Design A (Hook)**	**Design B (Hydraulics)**	**Design C (Projectile)**
Time	0	0	-	0
Consistency	0	0	0	-
Realistic	0	0	-	-
Ease of Attachment	0	0	-	-
Sum +	0	0	0	0
Sum 0	4	4	1	1
Sum -	0	0	3	3
Net Score	0	0	-3	-3
Continue?	Combine	Combine	No	No

Table A6: Decision matrix used to determine the mechanism to complete the task with the token.

Token				
Success Criteria	**Reference (Claw)**	**Design A (Collision)**	**Design B (Hydraulics)**	**Design C (Projectile)**
Time	0	+	-	+
Consistency	0	-	0	-
Realistic	0	0	-	-
Ease of Attachment	0	+	-	-
Sum +	0	2	0	1
Sum 0	0	1	1	0
Sum -	0	1	3	3
Net Score	0	1	-3	-2
Continue?	Combine	Combine	No	No

Table A7: Decision matrix used to determine the mechanism for pressing the final button.

Final Button				
Success Criteria	**Reference (Claw)**	**Design A (Projectiles)**	**Design B (Collision)**	**Design C (Extrusion)**
Time	0	0	+	+
Consistency	0	-	+	+
Realistic	0	-	0	0
Ease of Attachment	0	0	+	0
Sum +	0	0	3	2
Sum 0	0	2	1	0
Sum -	0	2	0	0
Net Score	0	-2	3	2
Continue?	Combine	No	Yes	Combine

Table A8: Decision matrix used to determine the robot design.

		Reference		**Design A**		**Design B**	
Success Criteria	**Weight**	**Rating**	**Weighted Score**	**Rating**	**Weighted Score**	**Rating**	**Weighted Score**
Modular ability	10%	5	0.5	5	0.5	5	0.5
Durability	10%	2	0.2	2.5	0.25	2	0.2
Consistency	50%	4	0.2	3.5	0.35	4	0.2
Time	25%	1	0.25	2	0.5	3	0.75
The effort needed to build	5%	3	0.15	3	0.15	3	0.15
		Total Score	1.3		1.75		1.8
			No		No		Develop

APPENDIX B

Electrical Systems and QR Code

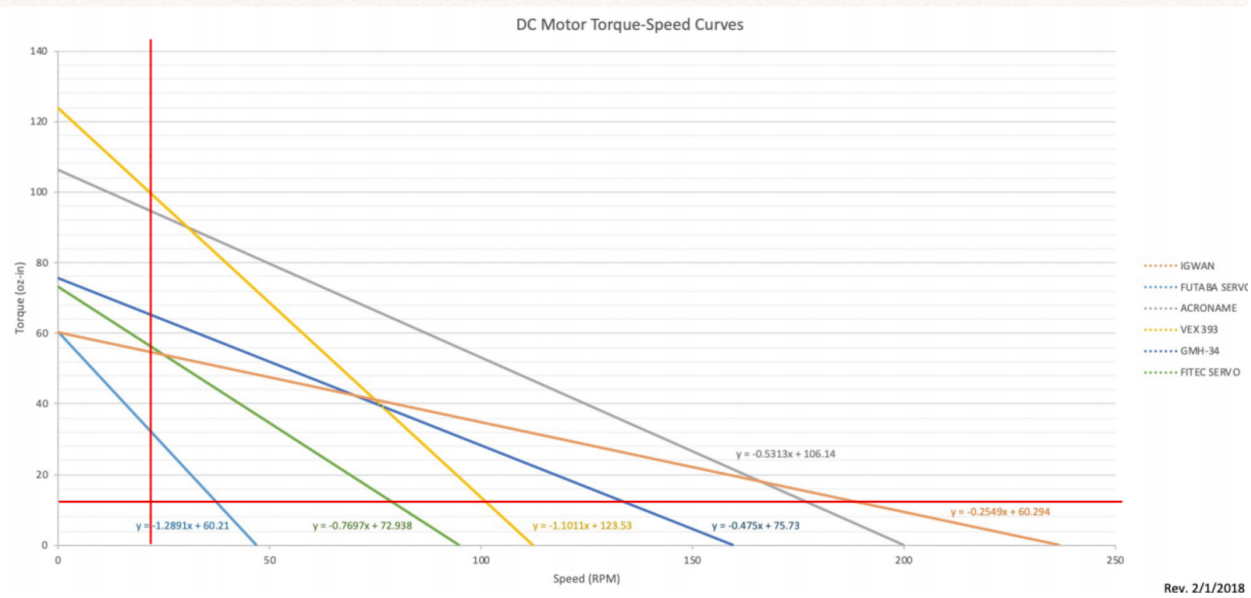

Figure B1: Speed-torque graph used in determining the drivetrain [2].

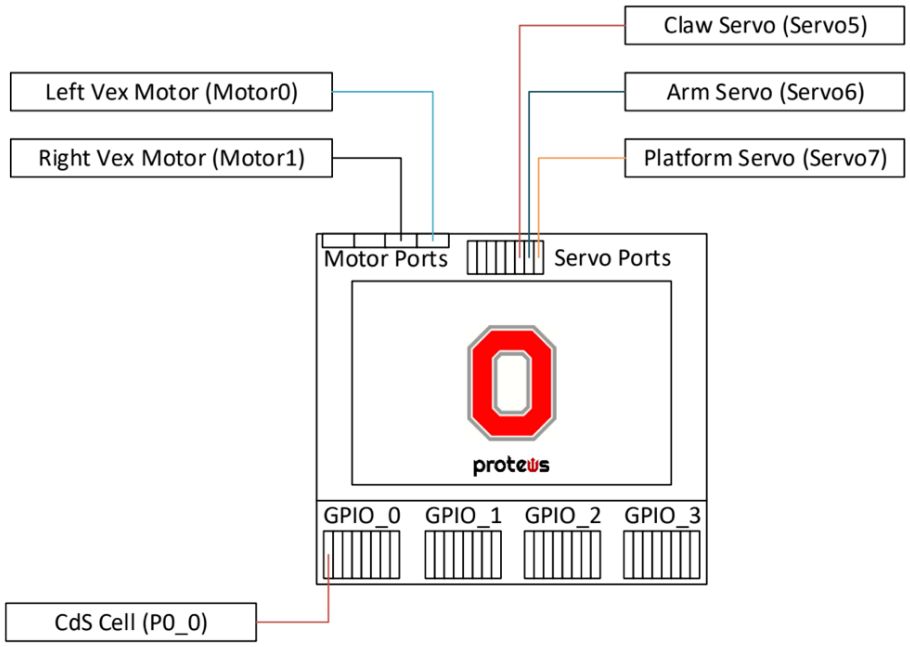

Figure B2: Diagram of the robot's electrical system.

Table B1: Description of input/output pins (sensors) on the robot.

Port ID	Wire ID	In-Code Name (Object Name)	Type of Sensor	Purpose	Additional Notes
P0_0	P0_0	CdS_Cell	CdS Cell	Detects the start light and DDR light	Orange, gray, and red wires

Table B2: Description of servo motors on the robot.

Port ID	Wire ID	In-Code Name (Object Name)	Type of Servo	Purpose	Additional Notes
Servo5	Servo 5	claw_servo	Micro Servo Motor	Controls the opening and closing of the claw, will be used for the token task	Orange and yellow wires
Servo6	Servo 6	arm_servo	Futaba Servo Motor	Controls the up and down movement of the arm, will be used for putting the claw in the starting position, the token, lever, and foosball tasks	Red, white, and black wires
Servo7	Servo 7	platform_servo	Futaba Servo Motor	Controls the rotation of the claw platform, will be used for putting the claw in the starting position and the foosball, lever, and token tasks	Red, white, and black wires

Table B3: Description of motors on the robot.

Port ID	Wire ID	In-Code Name (Object Name)	Type of Motor	Purpose	Additional Notes
Motor0	No flag	left_motor	Vex	Turn the left wheel	Red and black wires
Motor1	No flag	right_motor	Vex	Turn the right wheel	Red and black wires

Figure B3: QR code used for RPS technology.

APPENDIX C

Testing Logs

Table C1: Testing log for Performance Test 1.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
2/21	7:00	Determine if the robot could go up the left ramp and get over the power cord bump	JE	F	15	The robot is too heavy to get over the bump...we will focus on getting the robot up the other ramp
2/21	7:13	Find the initial turning angle of the robot to align with the ramp	JE	F	45	Incorrect values, change time for turning
2/21	7:15	Find the initial turning angle of the robot to align with the ramp and the distance of the first straight away	JE	F	50	Incorrect values, change time for turning and driving straight
2/21	7:20	Find the initial turning angle of the robot to align with the ramp and the distance of the first straight away	JE	H	30	Incorrect values, change time for turning and driving straight
2/21	7:21	Find the initial turning angle of the robot to align with the ramp and the distance of the first straight away.	JE	F	45	The correct angle was determined so that the robot was perfectly aligned with the bottom of the ramp
2/21	7:22	Finding the motor power and duration of straight drive up the ramp	JECA	F	100	Motor power must be increased slightly when going up the ramp
2/21	7:25	Finding the turn the robot can make after leaving the ramp surface	JE	F	45	Robot ran into the right wall when turning after going up the ramp. Change distance traveled to the base of the ramp to avoid this
2/21	7:30	Finding the turn the robot can make after leaving the ramp surface and how far to drive before making its next left turn	JE	F	45	Robot ran into center obstacle, so the turn must be decreased
2/21	7:32	Finding the turn the robot can make after leaving the ramp surface and how far to drive before making its next left turn	JE	A	30	The CdS cell did not sense the red start light
2/21	7:35	Finding the turn the robot can make after leaving the ramp surface and how far to drive before making its next left turn	JE	A	30	The robot was dangerously close to hitting the side of the wall while going up the ramp.
2/21	7:36	The straight drive time was increased so that the robot should reach the lever after making the appropriate turns	JE	F	45	This run was perfect! The robot ended up right in front of the lever without hitting the obstacle
2/21	7:52	The claw was added to the robot to see if the weight would impact the climbing of the ramp	JECA	B	30	Good results, but the robot turned unexpectedly and hit the obstacle before aligning with the lever
2/21	7:56	The angle of the claw was tested once the robot was aligned with the lever.	JECA	F	30	The claw can reach the lever but it turned too far to the right and missed it
2/21	8:03	The angle of the claw was tested once the robot was aligned with the lever.	JE	F	45	The robots turns are getting more inconsistent and the robot ran into the foosball wall
2/22	12:40	Prepare for the test 1	JECA	F	120	The robots times were adjusted and the motor percentage going up the hill was increased. The very first run the robot was successful
2/22	1:00	Performance Test 1	JECA	H	30	The robot worked on the first official run and we got 20 pts
2/22	1:15	Performance Test 1 with the bonus	JECA	H	45	The robot went backwards down the ramp and successfully earned 23 points
				Total time (min)	13	

Table C2: Testing log for Performance Test 2.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
2/27	5:30-9:00	To get the familiarize with RPS and make progress with Performance test 2	A	F	2400	Was able to get the RPS to work with the robot and solve issues from class.
2/28	1:39	Test out performance test 2	EJ	A	16	The robot successfully determined the color of the light, and the code for hitting the blue button actually perfectly hit the red button
2/28	1:44-1:54	Heavily RPS dependent code was tested	EJA	A	120	The robot moved forward but then was very choppy and moved backwards for some reason
2/28	2:02	Test out performance test 2, not the heavily RPS dependent	EJ	D	10	Sometimes the RPS doesn't work and the robot just drives straight and never stops.
2/28	2:03	Test out performance test 2, not the heavily RPS dependent	EJ	A	10	The robot was over a red light but detected blue. Successfully hit blue button when it was not supposed to
2/28	2:04	Test out performance test 2, not the heavily RPS dependent	EJ	A	10	Right wheel got stuck on the corner of the DDR and robot did not get the light detection
2/28	2:05	Test out performance test 2, not the heavily RPS dependent	EJ	B	10	got red light, turned correctly, but did not stop when it was going straight
2/28	2:11	Test out performance test 2, not the heavily RPS dependent	EJ	A	10	FIRST SUCCESSFUL RED RUN!!!!
2/28	2:13	Test out performance test 2, not the heavily RPS dependent	EJ	A	10	Aligned with the light, went a little but past the red button when it tried to hit it, also the robot kept turning after the hitting the button, angle needs to be decreased.
2/28	2:14	Test out performance test 2, not the heavily RPS dependent	EJ	D	30	Sometimes the robot goes a little bit past the red button, time driving straight need to be edited
2/28	2:16	Test out performance test 2, not the heavily RPS dependent	EJ	B	10	Robot goofed, did not align with the light so it did not do anything
2/28	2:17	Test out performance test 2, not the heavily RPS dependent	EJ	A	10	got the red button to work
2/28	2:18:-2:21	Tested on blue	EJ	A	120	did not align with the light, got the default and barely hit the red button (because of the default), once that was fixed, the robot did sense blue but when it turned to hit the blue the right wheel got stuck on the corner of the DDR machine
2/28	2:27	Tested on red	EJ	A	10	The robot aligned with the light and sensed red but didn't drive forward enough to hit the red button
2/28	2:37-2:38	Tested blue	EJ	B	20	Goofed and did not read blue, but successfully did the default
2/28	2:42	Tested on blue	EJ	F	10	The robot goofs often. The RPS also is faulty.
2/28	2:43-3:06	Tested on blue	EJ	A	200	Because the RPS was acting strange, we decided to hard code the driving and turning by time. This was a long process. Another problem was the robot was not successfully detecting some of the blue lights. So the light values were retaken and the thresholds were adjusted
2/28	3:14-3:20	Tested on blue	EJ	A	30	Problems were faced with coding to make the robot stop sensing light after the first two lights were sensed. The robot would stop when turning to hit the light because it sensed the light again
2/28	3:22-3:28	Tested on blue	EJA	H	200	The robot hit the blue button but did not hold it down. As the wheel spun, it didn't propell the robot into the button but only spun the button in a circle. The robot is still inconsistent with the RPS. Sometimes it misses the button.
2/28	3:31-3:39	Tested on red	EJ	A	30	We went back to the original code that was using RPS because timing was more inconsistent. The robot almost worked. It didn't quite press down the button. Also we are having problems with detecting the light more than once and the robot stops when it is supposed to drive into the button
2/28	3:46-4:38	tested on blue	EJA	A	250	We are having problems with holding the button down. The robot is coming at the blue button straight on and it does not seem to be working even when the motors are stopped close to when the robot runs into the button. The red (default) works great so now we are considering coding the robot to run into the blue button at an angle as well.
2/28	4:50-5:05	testing with red and blue	EJ	A	100	The robot sometimes turns way too far backwards because RPS is not working.
2/28	5:25-5:40	testing with hard time coding	EJA	D	200	The timing was difficult and inconsistent. Difficult to hit the button
2/28	5:42-5:55	went back to RPS and time coding	EJ	D	80	The robot works better this way. The mechanism for hitting the blue button was changed to come at the button at an angle
2/28	6:00-6:09	testing on red	EJCA	A	100	The robot was struggling to sense the red light. Light values were retaken.
2/28	6:10-6:20	Final testing on Thursday	EJCA	F	100	2 successful trials for each color
3/1	12:40-1	Performance Test 2	EJCA	F	200	2 successful trials for each color were done before the the test. The robot worked on the performance test (the random color was red)

Table C3: Testing log for Performance Test 3.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/5	5:48	Test straight RPS code	ECJ	A	30	The robot went straight well 10 inches... but it did not go backwards like it was supposed to. We are going to add write statements to see what the robot thinks it is doing
3/5	5:50	Test straight RPS code	ECJ	A	15	The proteus is not rebooting. It was determined that the USB part is broken. We bought another one for $1
3/6	1:50-1:57	Test straight RPS code	ECJA	B	20	We can tell that the RPS is trying to work. The robot drive straight and then starts pulsing but never stops pulsing.
3/6	2:06	Test straight RPS code and a turn to align with the slot	ECJA	F	60	The straight RPS did work. The robot turned and even went straight a little bit. We just need to increase the distance it went straight
3/6	2:08	Test straight RPS code and a turn to align with the slot	ECJA	B	10	Aditya accidently coded the robot to pulse all the way to 7 inches when it should've driven 7 inches and then checked with pulses/RPS
3/6	2:11-2:22	First test with the token	EJ	F	30	The robot drove straight, turned well, but did not drive far enough to the slot. The token was placed in the claw and did not fall out while being moved! The code was written for the arm angle to change but it did not work for some reason.
3/6	2:29	test with token and arm servo movement at the end	EJA	A	45	The robot still needs to go farther into the slot. The arm did rotate this time, but the claw opened at the same time which caused the token to go flying. The plan to fix this is to add sleep statements in between the arm and claw movements.
3/6	2:33	test with token and arm servo movement at the end	EJA	F	15	The arm angles need to be adjusted so that the arm is closer to the slot, and the platform needs to turn a tiny bit more. The robot also needs to drive further to be right up against the slot machine
3/6	2:36-2:40	PPT 03 test	EJAC	F	45	The arm angle is better. Now the problem is that the token does not fall when the claw is opened.
3/6	2:41	PPT 03 test	EJAC	A	60	FIRST SUCCESSFUL RUN! The solution was to offset the token a little more so that the token's weight will cause it to fall out of the claw's grasp
3/6	2:42	PPT 03 test	EJAC	A	60	Sometimes the token is not offset enough and the coin does not fall, it is lined up but the token stays balanced on the claw
3/7	1:50	PPT 03 test	EJA	F	60	The RPS was a little off from the day before the coin did not drop into the slot but it was dropped
3/7	1:54	PPT 03 test	EJ	A	60	the RPS was fixed. however the token did not fall on this run. This could be fixed by shaking the arm back and forth a little but once it is at the slot
3/7	1:56	PPT 03 test	EJ	B	100	The robot would be more successful if it drove forward a little farther
3/7	2:00	PPT 03 test	EJA	F	50	Sometimes the token is not offset enough and the coin does not fall, it is lined up but the token stays balanced on the claw
3/7	2:01	PPT 03 test	EJA	F	100	The coin was not perfectly aligned, but it did fall from the grip
3/7	2:03	PPT 03 test	EJA	F	50	The angle of the arm was shaken when the claw was open and the coin did fall out into the slot
3/7	2:04	PPT 03 test	EJA	F	300	The angle for shaking might be too much because the token was thrown which is too aggressive and inconsistent
3/7	2:07-2:17	PPT 03 test	EJA	F	60	This test did work because the angle of shaking was decreased.
3/7	2:20	PPT 03 test	EA	F	80	The code was edited to include a backwards turn after the coin is deposited.
3/7	2:25-2:30	PPT 03 test	EA	F	30	The robot is not always consistent in depositing the coin so the angle and the distance the robot drive was edited. Eventually the robot worked three times in a row
3/8	12:40	PPT 03 test	EJAC	F	300	the robots initial tests were not successful. The coin sometimes does not fall into the slot and the starting angle really impacts whether the robot is successful
3/8	12:50-1:20	PPT 03 test	EJAC	F	1200	Many aspects of the code were edited. Angles turned and distances driven were changed
3/8	1:30	Final PPT 03 test	EJAC	F	40	The robot worked on the fourth try!!! The extra task of hitting the lever was not attempted

Table C4: Testing log for Performance Test 4.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/20	12:45	Get the robot to press and hold the white DDR button down	AC	F	30	The arm is not able to hold down the button when the rubber bands are around the claw pieces
3/20	12:50	press the wifi button	AC	F	30	The RPS works for aligning the robot with the ddr but the claw is not perfectly aligned to hit the button. There are also still concerns as to whether the arm will hold it down
3/20	12:55	Press the WiFi button, back up and go to the foosball	ACE	F	30	The RPS is not always consistent. (Probably because of the starting position). Also the robot goes towards one side when traveling up the ramp so the rubber bands are going to be replaced to hopefully create more traction on the acrylic ramp
3/20	1-2:00	Work on pressing the WiFi button and getting up the ramp	AC	F	1000	The robot can't consistently press and hold the button. Tomorrow we will focus on just getting up to the foosball task without hitting the WiFi button
3/21	3:15	Find distances to get to foosball	AE	F	15	The distance to the bottom of the ramp is less than 20 in
3/21	3:21	Get to foosball	AJE	A	15	the robot turns too much at first and for never made the second turn
3/21	3:27	Get to foosball	AJE	D	30	First time didn't work...second time the turns looked great but the robot went too far in the second straight away. We are also adding a small turn to align the robot better with the foosball
3/21	3:34	Get to foosball	AJE	A	30	did not make second turn two times
3/21	3:43	Get to foosball	AJE	H	60	the robot works sometimes but the first distance is too long and RPS sometimes just does not work
3/21	3:51	Get to foosball	ACEJ	A	60	robot and RPS are inconsistent
3/21	3:59	Get to foosball	ECJ	B	30	Distance straight towards the ramp needs to be increased, team is trying to figure out why the pulsing method is not working
3/21	4:09	Get to foosball	ACEJ	D	15	The robot started without the light??? Something is off
3/21	4:14	Get to foosball	ACEJ	E	15	going to retake CdS cell values at some point. The robot for some reason is not making the second turn
3/21	4:21	Get to foosball	ACEJ	A	15	The robot is not making the second turn.. something to do with the pulsing method, Aditya added the 0.5 linear term back into the code
3/21	4:27	Get to foosball	ACJ	E	15	The robot turned immediately which is not even in the code. Craig is going to make sure that both of the motors are plugged in properly
3/21	4:34	Get to foosball	CE	C	30	Robot went around in circles.
3/21	4:43	Get to foosball	JEC	B	15	robot went backwards when it got to the ramp. large pulses
3/21	4:49	Get to foosball	ACEJ	B	30	pulsing too far forward in first straight away
3/21	4:55	Get to foosball	ACEJ	A	20	successfully fixed the turning and driving problems, some distances need to be changed
3/21	4:59	Get to foosball	ACEJ	A	15	The robot made it up the ramp! But it almost hit the DDR when turning so some distances will be changed
3/21	5:03	Get to foosball	ACEJ	E	30	Robot did not work on course b, but on course e it did work except the distance straight needs to be increased

Table C5: Testing log for Performance Test 4, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/21	5:09	Get to foosball	ACEJ	D	60	The distances are off. The first turning angle needs to be decreased. Also the robot acts unpredictability during the second turn
3/21	5:44	get to foosball	EJ	E	60	Two successful runs up the ramp
3/21	5:51	Get to foosball	EJ	E	30	Successful run that was not close to the DDR this time, now we just need to edit the distance traveled on the top level
3/21	6:03	Get to foosball	CJ	F	30	Worked! Need robot to be more centered
3/21	6:08	Get to foosball	CJ	E	30	Worked
3/21	6:11	Get to foosball	CJ	B	30	Worked
3/22	12:45	PT4	ACEJ	F	30	The Proteus was fully charged and went too close to the foosball
3/22	12:50	PT4	ACEJ	F	30	Aditya changed one of the turns so that it would not have to pulse as much, but for some reason it did not pulse to the correct location.
3/22	12:52	PT4	ACE	G	100	The robot went crazy, did weird pulsing things that it is not supposed to do, the robot is better on the other courses
3/22	1:03	PT4	ACJ	H	30	The robot got to the foosball and almost grabbed the slider. It was a little bit off. Robot is still not able to detect the red light
3/22	1:10	PT4	ACEJ	F	120	The robot RPS on the second turn was acting weird. Aditya is going to change that turn amount a little bit. It is so close to holding the sliders.
3/22	1:18	PT4	ACEJ	G	30	The robot needs to go forward a little bit to grab it. We are going to try move backwards bit by bit as the claw moves to the left holding the sliders
3/22	1:24	PT4	ACEJ	B	30	the robot did it when ellie pulled it backwards a little bit. There are multiple methods we are considering to have it pulse and move the slider at the same time
3/23	1:23	PT4	AJ	H	30	Robot did not turn to correct position on the top level
3/23	1:28	PT4	AJ	E	30	Arm is not reaching the sliders
3/23	1:38	PT4	AJ	F	120	Need to center the robot in front of the foosball
3/23	2:00	PT4	ECJ	H	60	Multiple methods were tested to get the robot to grab the slider. It did work once, but it is not consistent
3/23	2:25	PT4	CJ	H	60	A different method was tested where the claw opens and the platform rotates to wrap around the sliders. The angle of the arm needed to be adjusted so that it can actually fit around the slider. The issue is that the robot is not able to move them. So rubber bands were added to hopefully increase traction
3/25	12:40	PT4	JEA	F	120	Worked 3/5 times
3/25	1:13	PT4	ACEJ	F	600	The RPS is acting a little funny sometimes. The robot did work 2 times. the team is now working on getting back down the ramp
3/25	1:30	PT4	ACEJ	B	120	The robot was too powerful on the course. the slider bounced off the wall
3/25	1:45	PT4	ACEJ	F	30	Official Test 1: the robot got 15 points
3/25	1:57	PT4	ACEJ	H	60	the robot is working well on this course! The robot can do the task and is almost able to go down the ramp
3/25	2:03	PT4	ACEJ	E	50	the robot needs to turn a little bit less to get down the ramp. It is so close

Table C6: Testing log for individual competition.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/25	6:00	Complete foosball task by holding the token vertically	AE	F	30	The coin falls out of the claw onto the token platform, but the robot needs to be closer to the task for the token to go through the slot
3/25	6:32	Complete token task	AE	F	60	The robot is close enough to the task but the angle of the arm needs to decrease slightly
3/25	6:52	Complete token task	AE	B	120	RPS is inconsistent...the distance that the robot ends up from the token slot varies each time
3/25	7:00	Complete token task	AJE	F	30	The robot worked 3 times in a row! The first time it did not work because the robot was not far enough forward. That distance will be increased a little.
3/25	7:10	Go to DDR	AJE	E	30	A sleep statement needs to be added so that the robot does not begin moving backwards while the token is being released. The angles to get to DDR need to be adjusted
3/25	7:31	Go to DDR from token	AJE	F	30	Token worked and the robot turned to 315 but the robot needs to turn backwards first and drive forward
3/25	7:40	Go to DDR from the token	AJE	C	15	On this course the robot did not get close enough to the slot. The pulses were large backwards. The angle going to the DDR is almost right.
3/25	7:53	Testing RPS angles	AJE	E and B	45	The angles are a little bit off from the expected.
3/25	8:32	Token and DDR	AJ	E	300	The claw sometimes does not open enough for the token to come out
3/26	11:34	Token and DDR	A	B	25	pulseTurn was still an issue
3/26	6:10	Token and DDR	AC	F	30	The robot drove too close to the token deposit
3/26	6:27	Token and DDR	AC	E	25	The robot read blue when the red DDR light was shining and still missed the blue button
3/26	6:36	DDR	AC	B	55	The robot correctly read red at the DDR buttons but barely missed the red button
3/26	6:40	Up to foosball task	AE	E	55	The token didn't go into the top slot. When heading to the foosball task the robot hits the obstacle with the arm
3/26	7:00	Token, DDR, foosball	AE	C	60	Token missed the slot again but the DDR and foosball tasks were completed successfully
3/26	7:41	Token, DDR, foosball	AJ	B	60	The token worked both times. The red button worked but the blue button goofed. The foosball did work 1/2 times (the other time the robot goofed at the button and couldn't find its way up to the ramp)
3/26	7:50	DDR	AE	C/E/F	270	The robot didn't work on course C...RPS is acting up.The robot was most successful on F
3/26	8:22	Token, DDR, foosball	ACEJ	F	200	Inconsistent with the DDR
3/26	8:45	Token, DDR, foosball	ACEJ	E	200	Worked 2 times on red button, works best on E
3/27	1:00	Token, DDR, foosball	ACEJ	G	100	The new QR code did work on the course that our robot usually does not work with. Some of the DDR blue angles and distances need to be changed, and based on that we will change the distance and turns that are made on the second level

Table C7: Testing log for individual competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/27	1:20	Token, DDR, foosball	AJ	F	100	The robot does fit in the starting box. The robot misread the DDR light but still hit a button. The foosball is consistent but the lever is difficult to reach
3/27	1:30	Token, DDR, foosball, lever	AJE	G	100	Working well besides the fact that the Cds cell sometimes reads blue when the light is actually red. The token is literally perfect, the DDR is a sometimes, the foosball always gets primary points but sometimes bounces out of the final position. The lever hasn't been put down yet but we are working on it
3/27	1:45	Token, DDR, foosball, lever	AJE	E	150	Highest run yet! 92 points. The only thing we have not done is hit the final button. Need to retake the light color. Might use the red filter to make the light values more different. The time of the 92 run was 58 seconds so we have plenty of time to get down to the bottom level to hit the button. We are still considering which ramp will easier to get down.
3/27	2:03	Token, DDR, foosball, lever	ACEJ	H	150	The robot worked well the 3 times we tested it. The top level is sometimes off because there is no RPS
3/27	2:21	Token, DDR, foosball, lever	ACEJ	A	120	Worked well and barely missed the blue button. We are going to try the red filter to differentiate between red and blue
3/27	2:26	Token, DDR, foosbal, lever	ACEJ	A	60	The robot read blue again when it was red. The lever times are still being figured out. The obstacle needs to be removed by hitting it out of the way
3/27	2:36	Token, DDR, foosbal, lever	ACEJ	H	120	red filter was tested out and data logging was used
3/27	7:00	Token, DDR, foosbal, lever	AJE	H	120	The robot is still struggling to read the light color. The robot is hitting the obstacle out of the way more consistently
3/27	7:14	Full course	AJE	F	120	Token and DDR worked 1/2 times. The foosball was good both times. The lever is missed sometimes. The obstacle was successfully moved both times. Going down the ramp is still being worked on.
3/27	7:38	Full course	AJE	G	130	Successful run when it was red light. The only thing that failed was the robot hit the lever back up when it was driving to go down the ramp. The robot almost hit the blue button but when going up the ramp it fell off the side.
3/27	7:52	Full course	AJE	H	120	The robot now fits in the starting box. The robot did not complete the DDR task for either color so we are not going to use RPS checks for DDR.
3/27	8:21	Full course	AJE	E	120	Kill switch did not work on this course. But the robot was able to almost hit the DDR buttons
3/27	8:29	Full course	AJE	H	120	Did not hit the buttons. Did not align with the ramp to go up it
3/27	8:37	Full course	AJE	G	120	Both buttons worked! Going up ramp needs to be changed
3/27	8:46	Full course	AJE	G	120	Blue button worked but the red button was a little off. The top tasks are a little inconsistent depending on the positioning of the robot
3/28	11:30	Full course	A	F	2000	Aditya worked on polishing the code and working on the robot getting down to the bottom level while the other team members were in class
3/28	1:47	Full course	AJ	H	60	Token is good, DDR is close, and the top ramp needs work: will be better when DDR is consistent
3/28	1:49	Full course	AJ	H	60	Lever and foosball need work
3/28	1:57	Full course	AJ	F	60	The top level is very inconsistent. DDR red did work this time.
3/28	1:59	Full course	AJ	F	60	DDR red did not work this time. The timing to get to the lower level needs to be edited

Table C8: Testing log for individual competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/28	2:07	Full course	AJ	D	60	The token sometimes just falls horizontal and remains in the claw when the robot opens the claw. Ellie needs to be the person to always set up the token
3/28	2:08	Full course	AJ	D	60	Everything failed on this run. The token fell the wrong direction and the robot was not close enough to the foosball to grab it which is unusual
3/28	3:45	Full course	AJE	H	60	Something with RPS went wrong and the robot got stuck after the token
3/28	3:47	Full course	AJE	H	60	The robot is almost aligned parallel with the DDR (a little bit different of an approach as the other code)
3/28	4:05	Full course	AJE	E	60	The robot missed the blue DDR and then ran straight into the foosball and the kill switch did not stop the robot
3/28	4:16	Full course	EJ	D	60	The robot missed the blue DDR but Ellie positioned it as if it hit it. Then the robot did not hit the lever but it did hit the final button for the first time.
3/28	4:18	Full course	EJ	D	60	The robot missed the blue DDR but Ellie positioned it as if it hit it. Then the robot did not hit the lever but it did hit the final button for the first time.
3/28	4:27	Full course	EJ	B	60	The robot was way too close to the foosball for some reason and missed it
3/28	4:28	Full course	EJ	B	60	Platform angle of the foosball task was too little to start and the claw rammed right into the wall
3/28	4:36	Full course	EJ	F	45	Blue DDR almost worked, failed at the foosball
3/28	4:37	Full course	EJ	F	45	Platform angle of the foosball task was too little to start and the claw rammed right into the wall
3/28	4:50	Full course	EJA	A	30	The CdS cell is consistently not aligned with the DDR light. This is causing issues
3/28	4:52	Full course	EJA	A	30	Robot got stuck on the ramp on the token side.
3/28	4:58	Full course	EJA	F	45	Bue DDR did work, the robot got stuck when it tried to hit the obstacle out of the way
3/28	5:00	Full course	EJA	F	45	Everything failed but the robot did make it back down to attempt to hit the final button
3/28	5:03	Full course	EJ	C	60	Robot worked surprisingly well. The red DDR worked and the lever part even worked. It was not aligned to get down the ramp however
3/28	5:09	Full course	EJ	C	60	Robot worked surprisingly well. The red DDR worked and the lever part even worked. It was not aligned to get down the ramp however
3/28	5:11	Full course	EJA	B	45	The robot got stuck on the obstacle when it tried to knock it over.
3/28	5:27	Full course	EJ	A	45	Missed the DDR but the top level was almost perfect. The new technique of moving the claw up and down repeatedly worked well
3/28	5:29	Full course	EJA	A	60	DDR is not detecting the light. Everything else worked except the new lever method hit the lever back up on accident
3/28	6:07	Full course	JAC	C	40	Not close enough to the foosball. Ran into the wall when trying to go down the ramp
3/28	6:09	Full course	JC	C	40	The arm skimmed the obstacle and it was not knocked over. Then the robot got stuck

Table C9: Testing log for individual competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/28	6:27	Full course	JAC	F	30	New method DDR is tested to try and get the robot parallel with the DDR instead of at an angle
3/28	6:29	Full course	JAC	F	30	New method DDR is tested to try and get the robot parallel with the DDR instead of at an angle
3/28	6:36	Full course	JAC	H	30	the RPS is not aligning the robot parallel with the DDR well
3/28	6:37	Full course	JAC	H	30	the RPS is not aligning the robot parallel with the DDR well, the turn is too late
3/28	6:48	Full course	JAC	F	15	The robot is not aligned with the DDR light
3/28	6:49	Full course	JAC	F	15	The robot is not aligned with the DDR light. The robot also might need to turn more to hit the button
3/28	6:57	Full course	JAC	A	15	The robot is acting strange around the DDR because it cannot detect the light
3/28	7:08	Full course	JAC	A	20	It is so close to hitting a button, but it is not necessarily the correct button
3/28	7:09	Full course	JAC	A	20	Better aligned this time but it went to the blue button while it was the red light. It almost hit the button to get some primary points
3/28	7:21	Full course	JC	C	20	Got too close to the DDR and hit the black wooden box and got stuck
3/28	7:30	Full course	JC	C	20	RPS fail. The robot went back and forth forever after depositing the token
3/28	7:32	Full course	JC	D	20	Robot went to the red light when it was supposed to be blue
3/28	7:46	Full course	AJ	D	20	Needs to drive forward to actually hit the button. Also the code after the DDR needs to be edited so that the robot actually goes up the ramp
3/28	7:54	Full course	AJ	E	20	The robot actually detected blue but the robot drove too far and did not hit the button.
3/28	7:55	Full course	AJ	E	30	Red button was pressed but when the robot backed up to go up the ramp it started driving in circles. Something with the RPS code is probably off
3/28	8:10	Full course	AJ	E	30	Red button was pressed but when the robot backed up to go up the ramp it started driving in circles. Something with the RPS code is probably off
3/28	8:11	Full course	AJ	E	30	DDR is not perfect. Also the robot needs to drive forward more before going up the ramp.
3/28	8:20	Full course	AJ	H	60	Everything went wrong. The token, DDR, foosball, and lever all failed. The arm got stuck under the lever and did not press it down
3/28	8:22	Full course	AJ	H	60	Lever and foosball actually worked this time, DDR is no good
3/28	8:35	Full course	AJ	A	30	Robot did not detect the light and ran straight into a wall.
3/28	8:36	Full course	AJE	E	30	Almost hit the red button. The robot turns while aligning with the bottom of the ramp and the alignment is not perfect
3/28	8:44	Full course	AJE	A	30	Not aligned well with the bottom of the ramp which caused the robot to drive off the edge. However the robot did successfully hit the red button
3/28	8:45	Full course	AJE	A	30	Hit the DDR and got up the ramp but it missed the foosball
3/28	8:53	Final tests	AJE	E	30	The DDR missed and the robot ran off the side of the ramp. Not a great way to end the day

Table C10: Testing log for final competition.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
3/30	3:30	Full course	EAC	C	120	Token works. Robot cannot detect red color but does detect blue. Robot needs to be closer to foosball (arm does not reach) and it needs to line up more parallel to the foosball in order to hit the obstacle.
3/30	3:40	Full course	EAC	D	120	Token and DDR worked, but the robot was too far left so it fell off the ramp when it tried to go up. The pulsing with RPS seems to be causing this issue so we will comment out the pulsing.
3/30	4:00	Full course	EAC	B	120	Robot is now close enough to foosball and hits the obstacle. The arm missed the lever but the robot navigated down to the final button!
3/30	4:15	Full course	EAC	A	120	Without the pulsing before the ramp, the robot is aligned properly. All tasks completed except final button...the angles were off and the robot ran into the wall before going down the ramp. 87 points!
3/30	4:30	Full course	EAC	D	120	The robot was too close to the DDR task, so when it tried to turn to go up the ramp it got stuck
3/30	4:40	Full course	EAC	F	120	Another turn was added after the lever task to line up the robot better with the ramp. Token, DDR with blue, foosball, and lever are all consistent!! (primary and secondary points)
4/1	1:15	Full course	EAJ	B	60	Robot had some RPS issue and started to turn in circles after attempting the DDR. The robot was too cose to the DDR and the axel got stuck
4/1	1:16	Full course	EACJ	B	60	Missed token, hit blue DDR, foosball bounced off, lever got hit, did not make it down the ramp
4/1	1:19	Full course	ECAJ	B	60	Token was good, missed blue DDR, missed foosball (too far away), almost got the lever
4/1	1:22	Full course	EAJ	B	60	Token was good, got the blue DDR, not close enough to the foosball, did not hit the lever, ran into the wall when it tried to go down the ramp
4/1	1:31	Full course	EAJ	G	60	Missed the blue, the default did not work, when the robot tried to turn and go up the ramp it failed. The distance between the DDR box and the robot needs to be increased
4/1	1:33	Full course	EAJ	G	45	Too far to hit the blue button, back wheels still get stuck on the DDR buttons consistently
4/1	1:34	Full course	EAJ	G	45	Missed blue (too far) and the back wheels got stuck on the DDR when it was turning to go up the ramp
4/1	1:39	Full course	EAJ	F	60	Token was a little off, the blue DDR worked, not close enough to the foosball, the arm was not close enough to the lever to hit it, and the robot ran straight into the wall when attempting to go down the ramp
4/1	1:41	Full course	EAJ	F	60	Blue button was almost hit. The foosball was too short, and the lever was hit on the last try
4/1	1:58	Full course	EAJ	F	30	RPS mistake, the robot detected blue and proceeded to run right into the wall
4/1	1:59	Full course	EAJ	F	60	Foosball worked
4/1	2:00	Full course	EAJ	F	60	token good, hit blue ddr, too close to the foosball to move it, claw got stuck on the glass by the lever and we had to kill it
4/1	2:03	Full course	JC	G	30	axel got stuck on the red button, tried to hit the blue button but could not go forward to hit
4/1	2:09	Full course	JAC	F	20	Did not back up enough after the token so the light was nowhere near the DDR light to detect it
4/1	2:17	Full course	ACEJ	G	30	The axel got stuck on the red DDR button and the robot got stuck
4/1	2:18	Full course	ACEJ	G	60	Token and blue DDR worked well, the robot made it up the ramp but could not do the foosball
4/1	2:28	Full course	ACEJ	D	40	Detected blue when the light was red, did not hit either button though.
4/1	2:30	Full course	ACEJ	G	40	Token and DDR worked, foosball bounced out of final position, the robot ran straight into the obstacle so we had to stop it
4/1	2:32	Full course	ACEJ	G	50	token failed, got DDR and foosball, the robot missed hitting the obstacle off of its mark.
4/1	2:41	Full course	ACEJ	E	40	went too far and did not hit the blue button, too close to get the foosball, but the token did work, hit the obstacle and got stuck on the glass when trying to hit the lever.
4/1	2:43	Full course	ACJE	E	40	Detected the wrong color light, did go up the ramp and get the foosball, did get the lever, could not make it down the ramp
4/2	7:24	Full course	AJE	F	50	Stopped over blue light and hit the blue light! Not close enough to the foosball to grab it
4/2	7:26	Full course	AJE	F	45	Token did work, the DDr detected the red light correctly but did not drive far enough to actually press the button, the foosball worked but the lever was too far away to be hit
4/2	7:36	Full course	AJE	F	60	Token failed, the red DDR worked, the foosball worked and the lever worked too! The robot did not make it back down the ramp

Table C11: Testing log for final competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
4/2	7:38	Full course	AJE	B	20	Axel got stuck on the red button, tried to hit the blue button but could not go forward to hit
4/2	7:39	Full course	AJE	D	45	Token fell the wrong way, blue DDR is becoming more consistent, way too close to the foosball and did not grab the sliders
4/2	7:41	Full course	AJE	D	60	Token and red DDR were good, foosball was perfect, the claw got stuck on the lever so the team had to kill the run so that the servos would not get hurt from the humming
4/2	7:52	Full course	AJE	C	60	Token and blue light were perfect, not close enough to the foosball, touched the lever but it was at a weird angle so it did not go down. The robot got stuck while trying to get down the ramp and the wheel got stuck on the corner of the token box
4/2	7:54	Full course	AJE	G	60	Missed red DDR, did foosball well, hit the lever down perfectly, got stuck on the wall before it could check RPS and go down the ramp
4/2	7:57	Full course	AJE	E	15	The robot got the coin but was aligned wrong up near the green coin bucket. The robot was too far off the DDR to be able to hit any button or detect the light
4/2	7:58	Full course	AJE	E	60	Pretty good run! The robot got the token, narrowly missed the red light, got up the ramp, moved the foosball (but it did not stay in the final position), hit the obstacle, and hit the lever down. We still need to work on getting down the ramp
4/2	8:03	Full course	AJE	B	30	Token was ok, the light was detected red when it was actually blue, but the robot still got up the ramp because of the RPS checks that were added. Almost did the foosball
4/2	8:05	Full course	AJE	B	50	Almost hit the red button, not close enough to the foosball, the robot was at a weird angle and could not get the lever, final button was successful though, the RPS check on the top level is helping a lot with that
4/2	8:10	Full course	AJE	F	40	Token missed, blue light was detected but the robot missed the light barely, we are still working on getting the robot the perfect distance from the foosball so that it can get it everytime. (Considering trying an RPS check on the top level in the undead half of the course)
4/2	8:11	Full course	AJE	F	50	Token missed, light missed, not close enough to the lever or foosball
4/2	8:18	Full course	AJE	G	60	Token, button, and lever worked, foosball was too close. The final button was almost successful! The robot needs to turn to the right and then drive forward to get it everytime
4/2	8:20	Full course	AJE	G	60	Token missed- we need to find a way to make this more consistent (DDR too), foosball too far away, lever too far away, had to stop it before the end of the run
4/2	8:26	Full course	AJE	G	80	token was good, touched the button but did not press, way too far from the foosball, too close to lever, got stuck on token when driving to final button
4/2	8:28	Full course	AJE	G	80	Missed everything except for the final button! Great way to end the day
4/3	12:49	Full course	ACEJ	F	10	False start, the robot did not do anything after it drove forward a couple of inches
4/3	12:50	Full course	ACEJ	F	10	False start, the robot did not do anything after it drove forward a couple of inches
4/3	12:58	Full course	ACEJ	E	40	Token was thrown by the claw! Robot did a weird unexpected pulsing near the DDR and while aligning with the ramp
4/3	1:06	Full course	ACEJ	E	30	Missed blue DDR, the robot kept driving directly into the foosball-something was wrong in the code that Aditya added to check the position before driving to the foosball on the top level
4/3	2:35	Full course	ACEJ	E	45	Token went the wrong way, small edits are made to the DDR and it is still very inconsistent, got the foosball but was too far from the lever
4/3	2:37	Full course	ACEJ	E	45	The robot did not pulse where it usually does and the robot ran into the red button on accident- but it did hit it!
4/3	2:38	Full course	ACEJ	D	90	Hit the blue button! Perfect run except for the lever!!
4/3	7:00	Full course	ACEJ	D	20	Token was lovely, robot was stopped at the DDR
4/3	7:03	Full course	ACEJ	G	45	Token and blue button worked, way too close to the foosball
4/3	7:07	Full course	ACEJ	H	60	Token was good, almost hit the blue, foosball was perfect, turned wrong because the back wheels hit the wall
4/3	7:09	Full course	ACEJ	H	45	Token and red are good, too close to the foosball and the robot seems to be too far to the right side of the course when it is reaching for the foosball
4/3	7:15	Full course	ACEJ	A	45	Ran straight into the wall after doing DDR. RPS goof? somehow still got some foosball points
4/3	7:16	Full course	ACEJ	A	50	Token, foosball, obstacle is as far as it got
4/3	7:33	Full course	ACEJ	H	50	Not close enough to the foosball, also the token is getting more inconsistent
4/3	7:34	Full course	ACEJ	F	45	hit blue, too close to the foosball, it is hard to know what changes to make at this point because the robot is different on every course and every run.
4/3	7:46	Full course	ACEJ	F	30	Wayyy to far from the foosball so the run was killed

Table C12: Testing log for final competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
4/3	7:48	Full course	ACEJ	A	60	The token was good, the blue light was very close, but the robot was not aligned properly with the foosball and it drove too far
4/3	8:03	Full course	ACEJ	D	60	An RPS check was added on the top of the ramp to make the robot align with the foosball but it did not work and the robot got stuck checking and pulsing over and over again
4/3	8:05	Full course	ACEJ	D	60	Missed blue light, smooth turn to align with the ramp, but it was like 2 inches from grabbing rhe foosball sliders.
4/3	8:26	Full course	ACEJ	F	45	Everything on the bottom floor worked, but the foosball was missed and the robot just started backing up after the foosball attempt and never stopped
4/3	8:28	Full course	ACEJ	D	50	Tried to change the code so that the robot does not have to go forward to reach the sliders, but that it would drive directly to the correct position
4/3	8:38	Full course	ACEJ	B	50	Blue DDR is consistently missed, doesn't drive far enough forward. The foosball is really messed up at this point
4/3	8:40	Full course	ACEJ	B	40	Red DDR was a little bit short, foosball did the same thing twice, we need to change that code.
4/3	8:49	Full course	ACEJ	H	50	Token did not work, foosball was still a little bit off
4/3	8:51	Full course	ACEJ	H	40	Red and token did work, foosball needs to got farther
4/3	8:58	Full course	ACEJ	H	30	Did not detect the correct light color, but still made it up the ramp, the foosball was closer this time
4/4	2:17	Full course	ACEJ	A	60	Red DDR worked well! The foosball was perfect, the rest of the code was not tested
4/4	2:18	Full course	ACEJ	A	30	Weird pulsing is happening in the run early on near the token deposit
4/4	3:52	Full course	ACEJ	F	10	Did a weird fast turn that it never normally does, ran its wheel right into the green bucket and got stuck
4/4	3:52	Full course	ACEJ	F	20	Ran into the wall while trying to get the DDR button
4/4	3:57	Full course	ACEJ	H	30	Missed the token, keeps running into the wall for some reason while doing the DDR task
4/4	3:59	Full course	ACEJ	H	10	Wheel got stuck on the DDR black box when trying to drive parallel to it
4/4	4:00	Full course	ACEJ	D	20	Got stuck while trying to turn and go up the ramp. way too close to the wall
4/4	4:07	Full course	ACEJ	D	30	Weird pulsing near the bottom of the ramp caused the robot to not align with the ramp, we killed the robot then

Table C13: Testing log for final competition, continued.

Date	Time	Purpose of Testing	Engineers	Course	Time on Course (s)	Important Notes
4/4	4:10	Full course	ACEJ	D	40	Blue DDR looked great! robot was close to the wall when going up the ramp, the robot did a weird pulse backwards when aligning with the foosball
4/4	4:12	Full course	ACEJ	D	45	Axel got stuck on the red button, turned too soon to go up the ramp
4/4	4:15	Full course	ACEJ	C	20	Did a weird turn thing and threw the token, probably something to do with the starting position?
4/4	4:15	Full course	ACEJ	C	40	Turned too soon to go up the ramp and ran into the blue token box with its left wheel
4/4	4:17	Full course	ACEJ	C	20	Cds was way off the light and the robot ran into the wall
4/4	4:18	Full course	ACEJ	C	20	Cds was way off the light and the robot ran into the wall
4/4	7:42	Full course	ACEJ	A	96	foosball and DDR were perfect, the lever was hit, and the final button was good!!!!
4/4	7:44	Full course	ACEJ	A	60	Missed the token-has been inconsistent recently, thankfully foosball looks good. Lever was missed and the final button was not reached
4/5	1:05	Full course	ACEJ	F	90	Missed everything, had to stop it at the lever
4/5	1:07	Full course	ACEJ	F	50	Barely got token, barely missed blue, too close to the wall when driving up the ramp, nowhere near the foosball
4/5	1:09	Full course	ACEJ	B	30	DDR is so inconsistent, turned too close to the wall when trying to align with the ramp
4/5	2:08	Full course	ACEJ	B	45	Detected the wrong color light, foosball bounced off, could not drive to the lever
4/5	2:10	Full course	ACEJ	D	60	Most things failed, the token did not drop and the foosball did not make it all the way, bu the lever worked!!
4/5	2:18	Full course	ACEJ	D	90	Pressed red, got foosball, and lever, did go down the ramp and hit the button!
4/5	2:20	Full course	ACEJ	E	90	Foosball is looking much better but on some courses it is more likely to bounce off of the final position
4/5	2:22	Full course	ACEJ	F	50	Robot did not get close enough to the token deposit, CdS cell did not detect the light, missed the foosball
4/5	2:24	Full course	ACEJ	F	45	Detected the wrong light again, have to figure out technique for getting values before the competition tomorrow
4/5	2:35	Full course	ACEJ	D	30	Detected the wrong light again, still made it up the ramp which is good
4/5	2:39	Full course	ACEJ	D	20	Ran straight into the wall after DDR
4/5	2:40	Full course	ACEJ	G	30	Got too close to the wall and got stuck while trying to turn and go up the ramp
4/5	2:42	Full course	ACEJ	G	30	Detected the wrong light again, did not make it up the ramp
4/5	2:43	Full course	ACEJ	G	45	Red DDR worked! a little too far away from the foosball to get it
4/5	2:45	Full course	ACEJ	G	60	Token is making us worried! Does not always work. Red worked, barely missed the foosball

APPENDIX D

Schedule and Time Breakdown

Table D1: Team design schedule.

Team Task Schedule

P - Primary, responsible for doing and knowing all about a task
S - Secondary, strong backup, takes over if necessary
C - Contributor, helps with task, may not have in depth understanding

No.	Task	Sched. Start	Actual Start	Sched. Finish	Actual Finish	Craig	Aditya	Jodie	Ellie	Est. Hours	Actual Hours	% Done
1	Team Contract	2/1	2/1	2/5	2/5	P	P	P	P	0.5	0.5	100%
2	Brainstorming											
	a. Sketches	2/1	2/1	2/5	2/5	P	S	S	S	1	1	100%
	b. Strategy	2/1	2/1	2/5	2/5	P	P	P	P	1	1	100%
3	Design Schedule	2/6	2/6	2/6	2/6	P	S	S	P	1	1	100%
4	Prototype Chassis Design	2/8	2/10	2/8	2/10	P	C	S	C	2	8	100%
5	Mock-Up/Solid Model	2/8	2/10	2/8	2/10	P	P	P	P	1	2	100%
6	Drivetrain Analysis	2/11	2/11	2/11	2/12	P	P	P	P	1	2	100%
7	Project Portfolio	3/1	3/17	4/21	4/21	S	S	P	P	10	12	100%
8	Agenda and Notes	2/12	2/11	2/12	2/13	P	P	P	P	0.5	1	100%
9	Prototype Chassis Built	2/13	2/15	2/20	2/21	P	C	P	C	5	3	100%
10	Write Code for PT1	2/13	2/10	2/20	2/22	C	P	C	P	3	7	100%
11	Final Report First Draft	2/15	2/23	3/5	3/5	P	P	P	P	2	5	100%
12	Final Report Outline	2/18	2/17	2/19	2/19	P	P	P	P	2	1	100%
13	Review Design Schedule	2/19	2/25	2/19	2/25	P	P	P	P	0.5	0.25	100%
14	Code Representation	2/20	2/22	2/20	2/24	C	P	C	P	2	1	100%
15	Prototype Chassis Test	2/21	2/21	2/21	2/22	P	C	P	C	1	5	100%
16	Write Code for PT2	2/21	2/25	2/27	2/27	C	P	C	P	2	5	100%
17	Finalize Chassis Design	2/22	2/22	2/23	2/22	P	C	P	C	0.5	2	100%
18	Finalized Chassis Built	2/25	2/20	3/6	2/22	P	S	P	C	2	8	100%
19	Budget and Testing Log	2/26	2/26	2/26	2/26	C	C	P	S	1	1	100%
20	Write Code for PT3	2/28	3/4	3/6	3/8	C	P	C	P	2	4	100%
21	Prototype Claw Design	3/7	2/20	3/8	3/6	P	C	S	C	1	5	100%
22	Write Code for PT4	3/7	3/18	3/23	3/25	C	P	C	P	2.5	5	100%
23	Prototype Claw Built	3/11	2/22	3/13	3/8	P	S	P	C	2	2	100%
24	Final Report Second Draft	3/12	3/18	3/30	3/20	P	P	P	P	2	4	100%
25	Prototype Claw Test	3/15	2/20	3/15	3/25	P	P	C	C	1	5	100%
26	Finalize Claw Design	3/18	3/25	3/18	3/29	P	C	P	C	1	1	100%
27	Electrical Documentation	3/18	3/6	3/19	3/18	S	C	P	P	1.5	1	100%
28	Finalize Claw Built	3/20	3/25	3/22	3/29	P	S	P	C	2	1	100%
29	Isometric for Display	3/27	3/27	4/2	4/3	P	C	P	C	2	3	100%
30	Working Drawing Set	4/3	4/14	4/6	4/14	P	C	P	C	4	6	100%
31	Final Written Report	4/8		4/21		P	P	P	P	3	5	100%

Revisions

Name	Date	Changes
Ellie	2/6	Document Created
Team	2/6	Initial Plan
Jodie	2/13	Updating progress and completions.
Aditya	2/17	Updating progress and completions.
Ellie	2/25	Updating progress and completions.
Jodie	3/18	Updating progress and completions.
Jodie	3/31	Updating progress and completions.
Ellie	4/21	Updating progress and completions.

Table D2: Weekly time breakdown by team member.

Name	Week	Total Hours	Documentation	Project Management	CAD	Coding	Testing	Building	Other
Aditya	2/3/2019 - 2/9/2019	4	0	2	0	1	1	0	
Craig	2/3/2019 - 2/9/2019	6	2	2	0	1	1	0	0
Jodie	2/3/2019 - 2/9/2019	4.25		3		1	0.25		
Ellie	2/3/2019 - 2/9/2019	7	2	2		1		2	
Craig	2/10/2019 - 2/16/2019	21	3	1.5	10	1.5	2	3	0
Aditya	2/10/2019 - 2/16/2019	10	0	1	2	5	2	0	0
Ellie	2/10/2019 - 2/16/2019	9	2	1	2	2		2	
Jodie	2/10/2019 - 2/16/2019	6	2	1	1			2	
Craig	2/17/2019 - 2/23/2019	13	0	1	2	0	4	6	
Jodie	2/17/2019 - 2/23/2019	12	1.5				4.5	6	
Aditya	2/24/2019 - 3/2/2019	19	2	2	3	4	5	3	
Ellie	2/24/2019 - 3/2/2019	24	3	1	3	2	10	5	
Ellie	3/3/2019 - 3/9/2019	11	4	1		3	3		
Jodie	2/24/2019 - 3/2/2019	11	3	1		1	5	1	
Aditya	3/3/2019 - 3/9/2019	7	0		0	3	4	0	
Ellie	3/10/2019 - 3/16/2019	6.5	2	0.5		2	2		
Jodie	3/3/2019 - 3/9/2019	8	4	1			3		
Aditya	3/10/2019 - 3/16/2019	0.25	0	0	0	0	0	0	0.25
Craig	3/17/2019 - 3/23/2019	14	0	1	2	1	9	1	0
Craig	3/10/2019 - 3/16/2019	0	0	0	0	0	0	0	0
Craig	3/3/2019 - 3/9/2019	12	1	1	2	1	6	1	0
Craig	2/24/2019 - 3/2/2019	15	1	1	2	1	8	2	0
Aditya	3/17/2019 - 3/23/2019	11.75	0.5	0	0	6	5	0.25	0
Jodie	3/17/2019 - 3/23/2019	13	5	1			6	1	
Ellie	3/17/2019 - 3/23/2019	18	6	1		1	10		
Aditya	3/24/2019 - 3/30/2019	21	0	0	0	6	15	0	0
Craig	3/24/2019 - 3/30/2019	14	1	0	3	1	8	1	0
Jodie	3/24/2019 - 3/30/2019	15	3				12		
Ellie	3/24/2019 - 3/30/2019	25	3	1		1	20		
Craig	3/31/2019 - 4/6/2019	16	2	1	6	0	6	1	0
Craig	4/7/2019 - 4/13/2019	6	0	0	2	0	4	0	0
Ellie	3/31/2019 - 4/6/2019	14	4				10		
Aditya	3/31/2019 - 4/6/2019	22.25	0	0	0	10	12	0.25	0
Jodie	3/31/2019 - 4/6/2019	12	4	1			7		

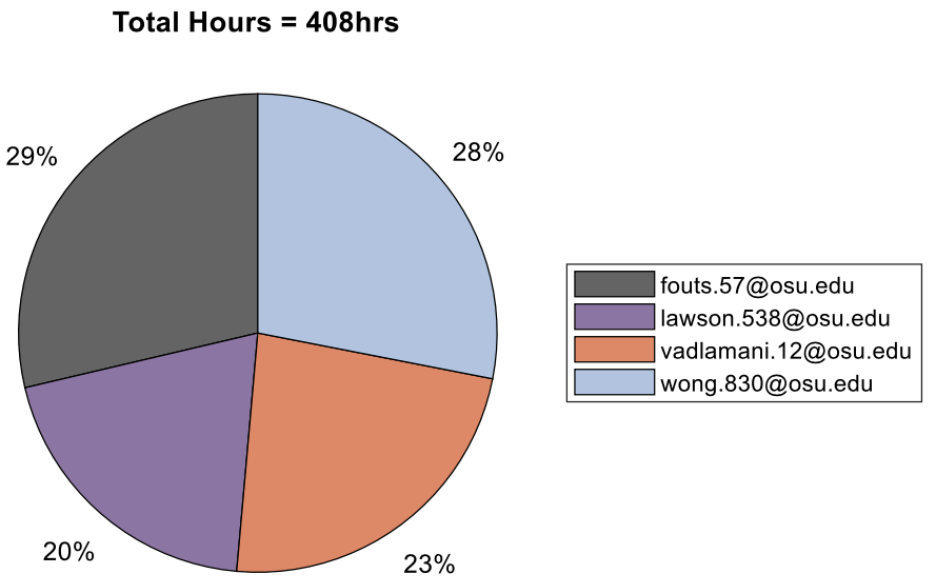

Figure D1: Total hours spent on robot project by team member.

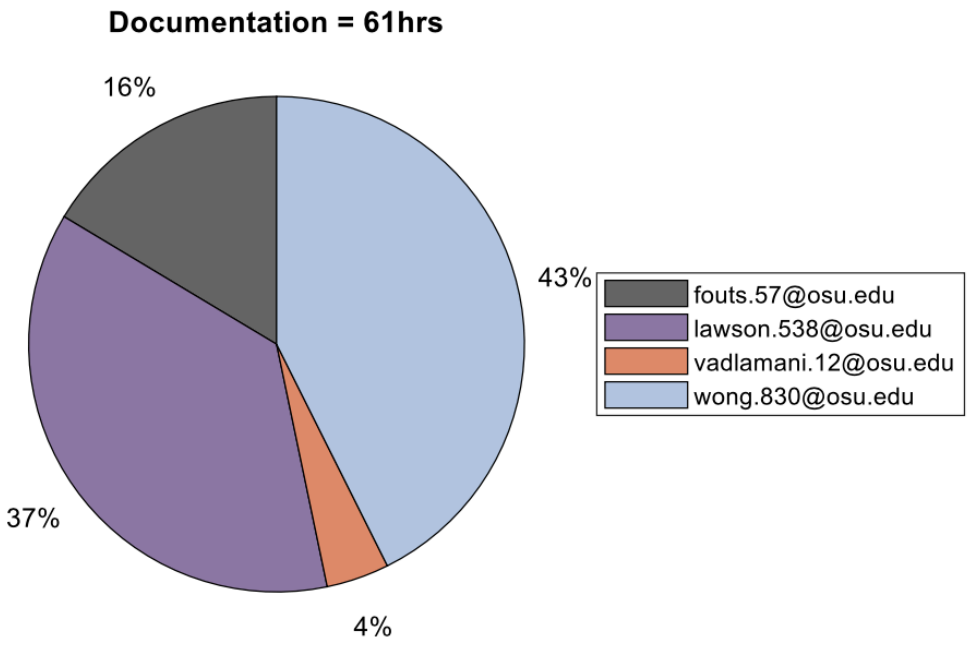

Figure D2: Total hours spent on documentation by team member.

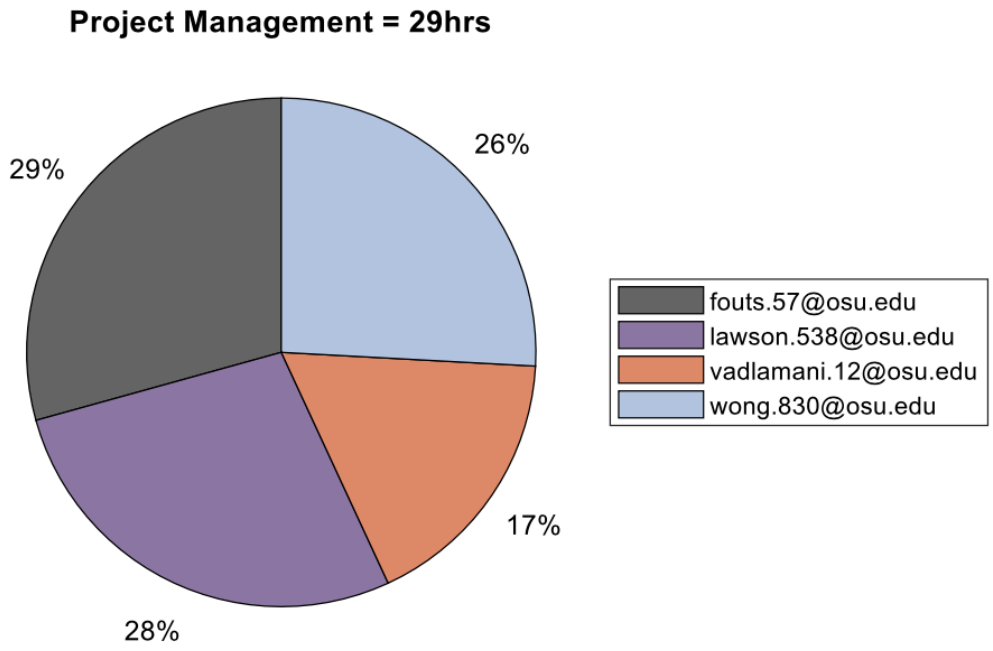

Figure D3: Total hours spent on project management by team member.

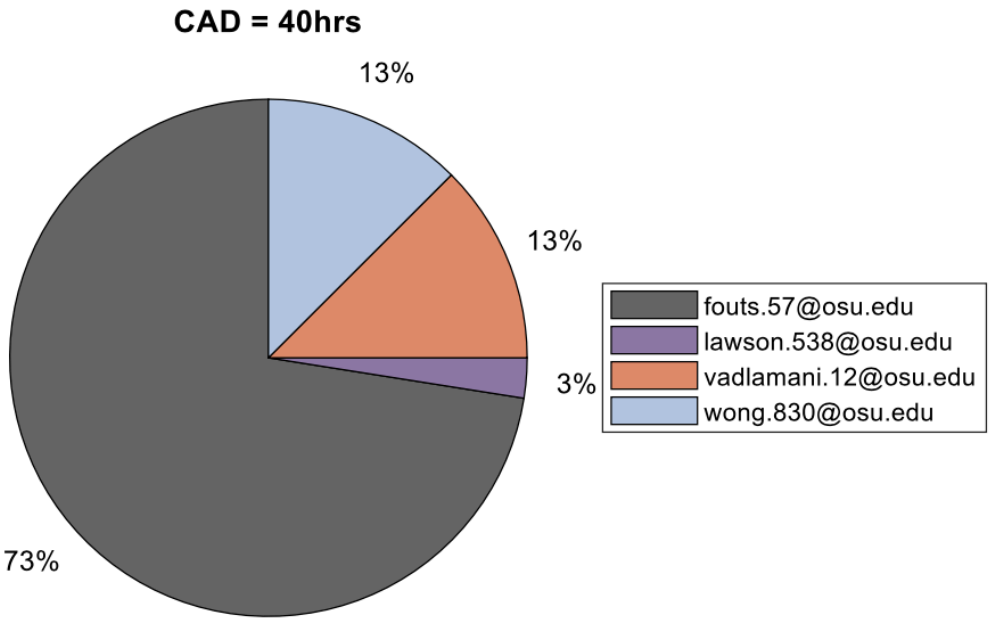

Figure D4: Total hours spent on CAD by team member.

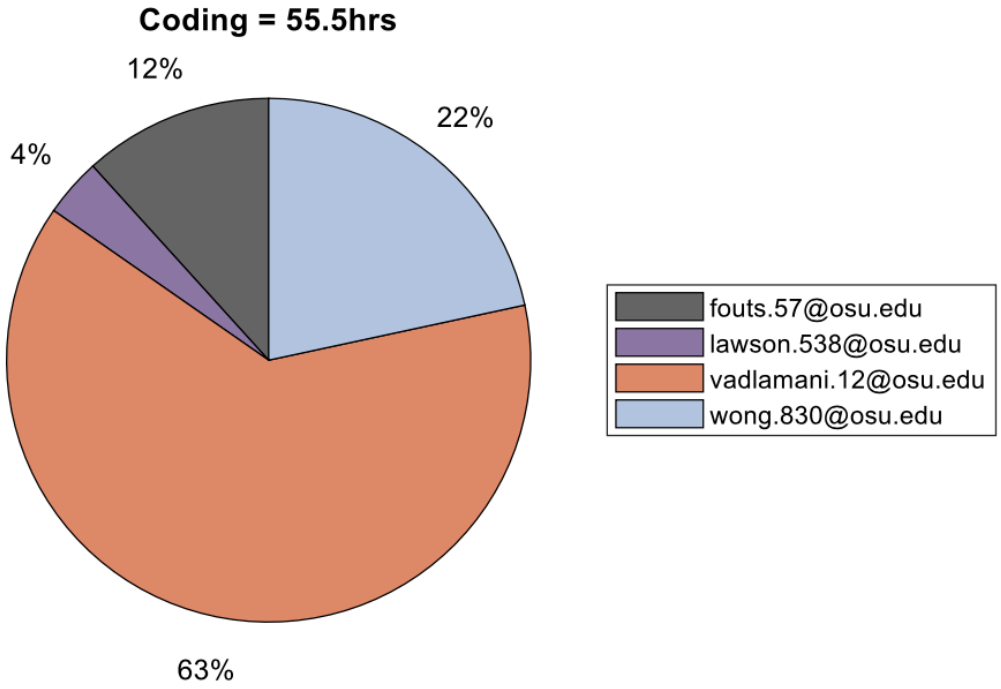

Figure D5: Total hours spent on coding by team member.

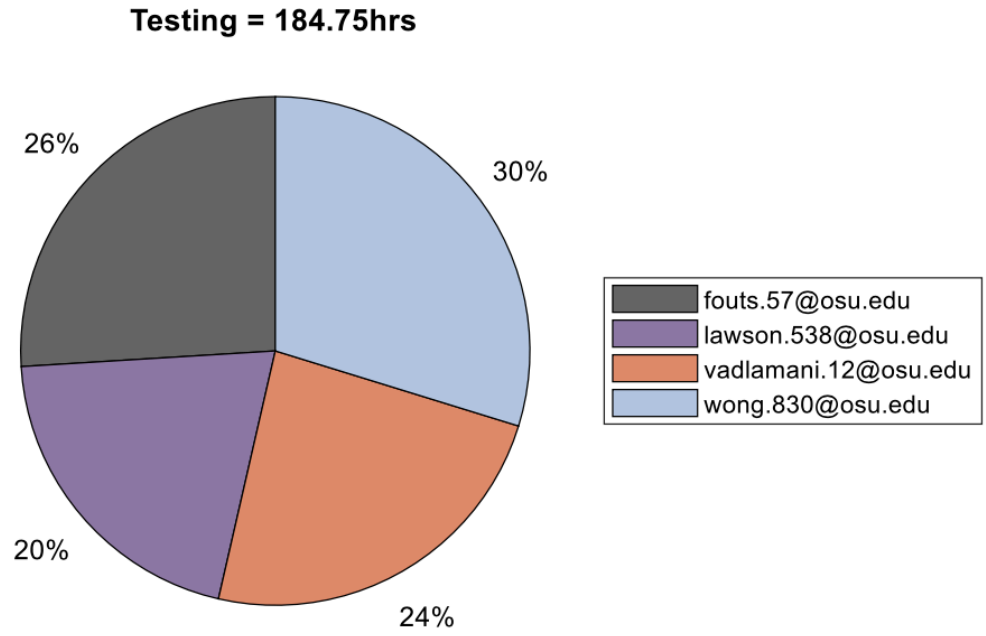

Figure D6: Total hours spent on testing by team member.

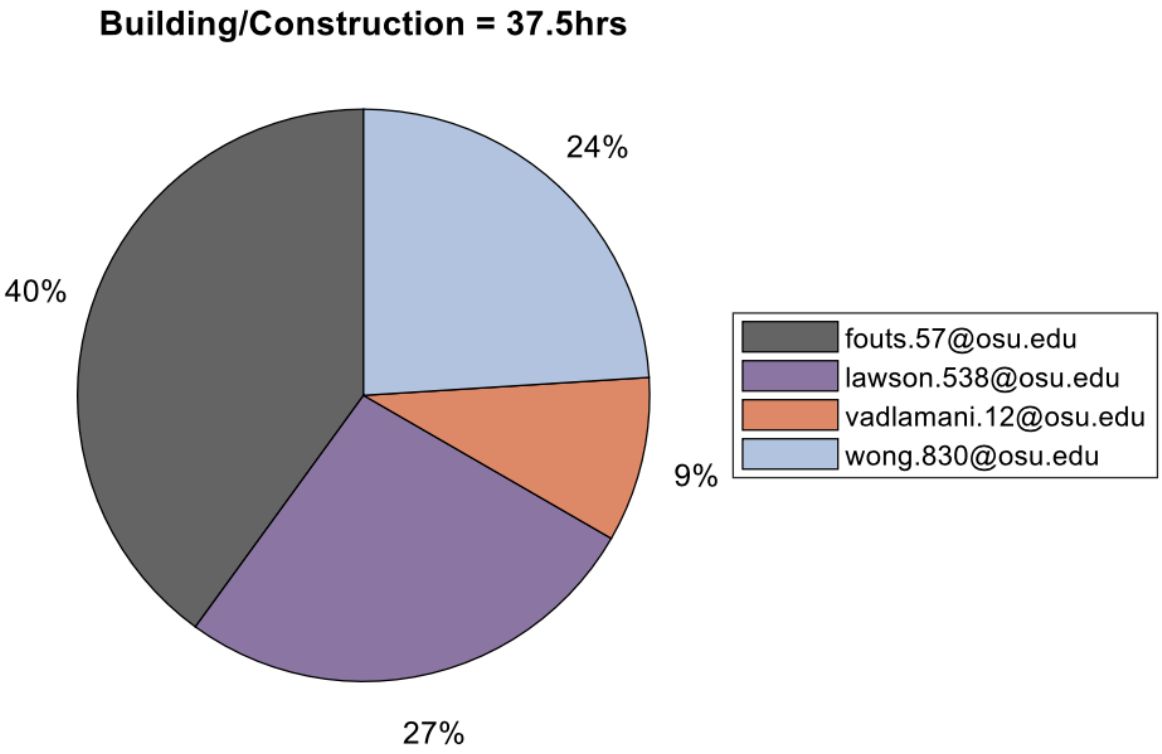

Figure D7: Total hours spent on building by team member.

APPENDIX E
Sample Calculations

$$\boldsymbol{s_{linear} = \frac{d}{t}} \tag{E1}$$

$$\boldsymbol{s_{rot} = \frac{s_{linear}}{C}} \tag{E2}$$

$$\boldsymbol{F_{motors} = W sin\theta + F_{friction}} \tag{E3}$$

$$\boldsymbol{\tau_{motors} = F_{motors} + r_{wheel}} \tag{E4}$$

Sample Calculation for Equation E1

$\boldsymbol{s_{linear}} = \frac{201.64\ in}{63\ s} = 3.20\ in/s$

Sample Calculation for Equation E2

$s_{rot} = \frac{3.20\ in/s}{10.99\ in} = 0.29\ revolutions/s = 17.46\ revolutions/min$

Sample Calculation for Equation E3

$F_{motors} = 24.3\ oz\ sin(18.5°) + 8\ oz = 15.7\ oz$

Sample Calculation for Equation E4

$\tau_{motors} = 15.7\ oz * 1.75\ in = 27.5\ oz - in$

APPENDIX F
Symbols

s_{linear}	Linear robot speed (in/s)
d	Distance (in)
t	Time (s)
s_{rot}	Shaft rotational speed (revolutions/min)
F_{motors}	Motor force (oz)
W	Weight (oz)
θ	Angle of ramp (degrees)
$F_{frction}$	Frictional force (oz)
τ_{motors}	Required motor torque (oz-in)
r_{wheel}	Radius of wheel (in)

APPENDIX G
Programming

Main

1. Initialize the robot by setting the angles for the robot's arm in its starting position and having the token in the claw
    a. Call a method that sets the robot's initial state (initialState)
2. Have the robot wait until the CdS cell detects the red light in the start box indicating that the clock has started or until 5 seconds have passed since the robot turned on.
    a. Call a method that outputs a true/false value depending on what the value of the CdS cell is (isRed)
3. Once the light is detected have the arm go into driving position (180 degrees vertical)
    a. Call a method that sets the angle of the arm's servo (setDegree)
4. Drive to the token machine on the bottom floor and drop the token into the top slot
5. Drive towards the DDR buttons and use the CdS cell to detect the color of the light and based on the light hit the appropriate button
6. Drive up the ramp to the foosball task and complete move the disk
7. Drive toward the lever and push it down
8. Drive back to the starting button using the other ramp and press it
9. For tasks 4-9, call a specific method for each task. Within those functions, the program will call other functions that allow the motor to turn and drive straight, using information from RPS to determine how the robot should move.

Figure G1: Initial code representation for the main function.

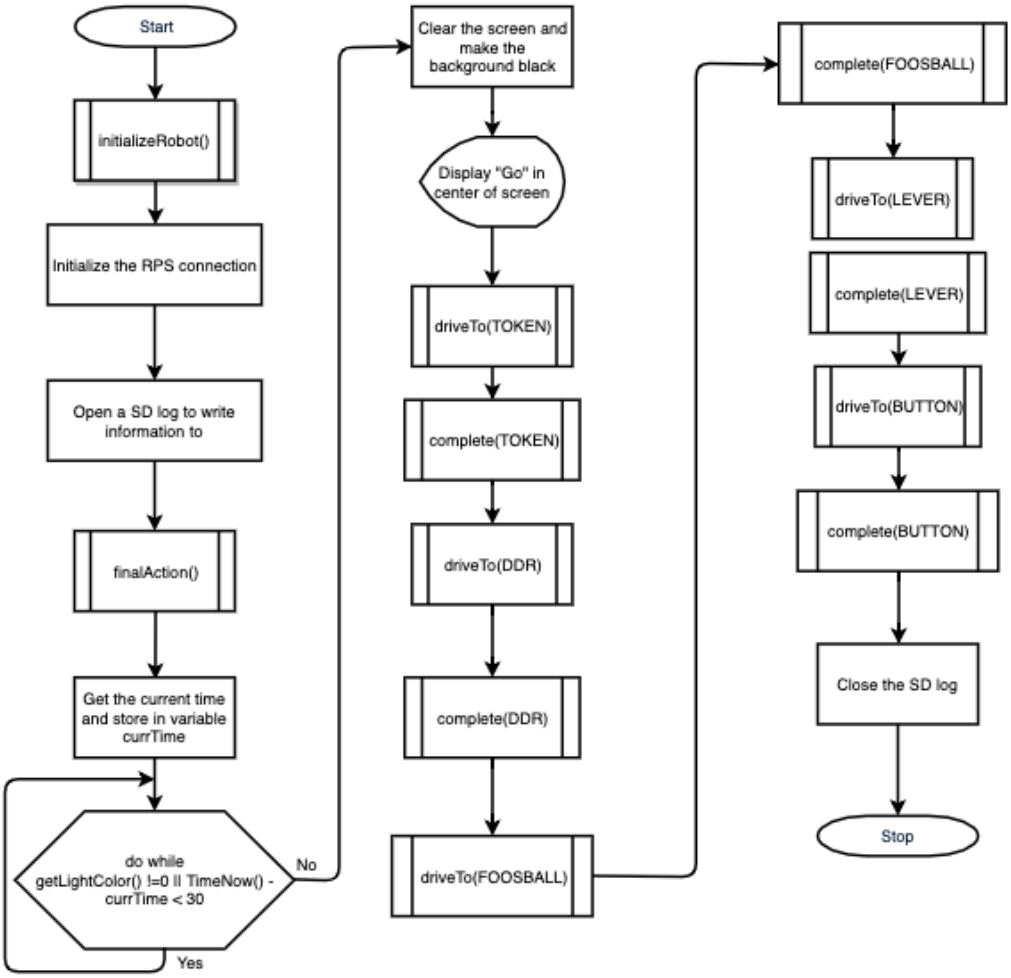

Figure G2: Flowchart representation of the main program.

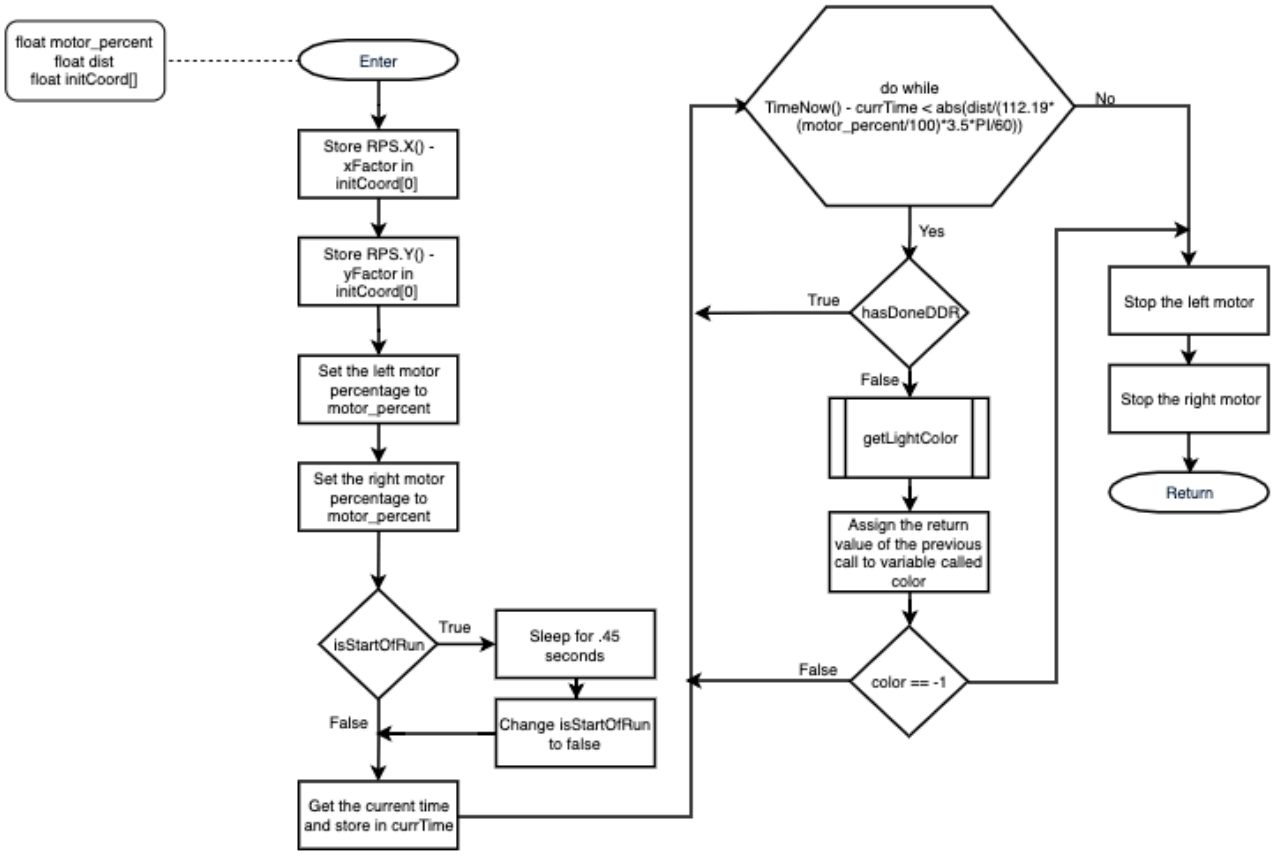

Figure G3: Flowchart representation of the driveStraight method which is a prominent method.

Final Code

```
#include <FEHLCD.h>
#include <FEHIO.h>
#include <FEHUtility.h>
#include <FEHMotor.h>
#include <FEHServo.h>
#include <FEHRPS.h>
#include <FEHSD.h>
#include <math.h>
#include <string.h>
#define PI 3.141592653
#define DISTANCE_TOLERANCE .35
#define ANGLE_TOLERANCE 1.5
#define ARM_SERVO_MIN 1620
#define ARM_SERVO_MAX 2450
#define CLAW_SERVO_MIN 560
#define CLAW_SERVO_MAX 1950
#define PLATFORM_SERVO_MIN 560
#define PLATFORM_SERVO_MAX 2500
#define INITIAL_PLATFORM_ANGLE 170.0
#define INITIAL_ARM_ANGLE 115.0
#define INITIAL_CLAW_ANGLE 0.0
#define DRIVING_STATE
platform_servo.SetDegree(180);Sleep(.5);arm_servo.SetDegree(35);Sleep(.5);cla
w_servo.SetDegree(0);
#define MOTOR_PERCENT_ON_PULSE 30
#define MOTOR_PERCENT_ON_LEVEL 30
#define MOTOR_PERCENT_ON_INCLINE 45
#define CW false
#define CCW true

using namespace std;

/**
* Declaring and initializing analog input pin for cds cell.
*/

AnalogInputPin CdS_Cell(FEHIO::P0_0);

/**
* Declaring and initializing the two motors in the front of the robot.
*/

FEHMotor left_motor (FEHMotor::Motor0, 9.0);

FEHMotor right_motor (FEHMotor::Motor1, 9.0);

/**
* Servo motors for the rotating platform mount, the arm, and the claw.
*/

FEHServo platform_servo(FEHServo::Servo7);

FEHServo arm_servo(FEHServo::Servo6);
```

```
FEHServo claw_servo(FEHServo::Servo5);

/**
* Enumerations of task.
*/

enum Task{TOKEN,DDR,FOOSBALL,LEVER,BUTTON};

/**
* Boolean to determine if it is the start of the run
*/
bool isStartOfRun = true;

/**
* Boolean to determine if light has been detected for DDR and an integer to
store the color
*/

bool hasDoneDDR = false;
int color = -1;

/**
* Correction factors for RPS
*/

float xFactor;
float yFactor;

/**
* Correction factor for ambient light differences
*/
float redThreshold = .25;
float blueThreshold = .85;

/**
* @brief distance
*              helper function to calculate distance between two points
* @param x
*              x-coordinate of first point
* @param y
*              y-coordinate of first point
* @param z
*              x-coordinate of second point
* @param w
*              y-coordinate of second point
* @return
*              distance between the points (x,y) and (z,w)
*/
float distance(float x, float y, float z, float w){

    return pow(pow(z - x,2) + pow(w - y,2),.5);
}

/**
* @brief getLightColor
*                  helper function to determine light color
```

```
* @return
*          integer value corresponding to a color (Red -> 0, Blue ->1, and No
light -> -1)
*/
int getLightColor(){

    if (CdS_Cell.Value() <= redThreshold + .5){

        return 0;

    }else if(CdS_Cell.Value() > redThreshold + .5 &&CdS_Cell.Value() <
blueThreshold + .35){

        return 1;
    }

    return -1;
}

/**
* @brief turn
*              General function for the robot's turning.
* @param isLeft
*              boolean parameter to dictate if the robot turn's left or not
* @param motor_percent
*              the motor power percentage
* @param angle
*              the amount the robot needs to turn in degrees
* @param expHeading
*              an array which stores the robot's expected heading
*/
void turn(bool isLeft, float motor_percent, float angle, float expHeading[]){

    //Get current orientation and determine the expected heading

    float initHeading = RPS.Heading();

    expHeading[0] = isLeft ? fmod((initHeading + angle+360), (float)360.0) :
fmod((initHeading - angle+360), (float)360.0);

    //Have the wheels move in opposite directions for a calculated amount of
time

    left_motor.SetPercent(motor_percent * pow(-1,(int)isLeft));

    right_motor.SetPercent(motor_percent * pow(-1,(int)!isLeft));

    float currTime = TimeNow();

    while(TimeNow() - currTime <
abs(((8.58*PI/4)/(112.19*(motor_percent/100.)*3.5*PI/60)) * (angle/90.0))){
        SD.Printf("Time: %f\n\t%f\n",TimeNow(), CdS_Cell.Value());
    }

    //Stop both motors.

    right_motor.Stop();
```

```
    left_motor.Stop();
}

/**
* @brief pulseTurn
*              function pulses the robot to the correct heading if it is not
within a certain thresholds
* @param expHeading
*              pointer to a float where the expected heading is stored
*/

void pulseTurn(float* expHeading){
    if(RPS.Heading() >= 0){
        //Counters for the number of pulse in a (counter)clockwise direction
        int numPulsesCW = 0;
        int numPulsesCCW = 0;
        float angleOff;
        float currTime = TimeNow();

        //Temporary float array for the calls to the turn method
        float temp[1];

        //Consider the case where the expected heading is close to 0 degrees
separately
        if(*expHeading <= ANGLE_TOLERANCE || *expHeading >= 360 -
ANGLE_TOLERANCE){
            while ((RPS.Heading() > *expHeading + 1 || RPS.Heading() <
*expHeading - 1) && !(numPulsesCCW > 2 && numPulsesCW > 2)){
                //Depending on which side of 0 degrees the robot is facing,
turn in a certain direction
                if(RPS.Heading() < 360 - ANGLE_TOLERANCE && RPS.Heading() >
270){
                    angleOff = abs((360*(*expHeading <= ANGLE_TOLERANCE &&
*expHeading >= 0) + *expHeading) - RPS.Heading());
                    turn(CCW,MOTOR_PERCENT_ON_PULSE, angleOff/2 +
ANGLE_TOLERANCE,temp);
                    numPulsesCCW++;
                }else if(RPS.Heading() > ANGLE_TOLERANCE  && RPS.Heading() <
90){
                    angleOff = abs((360*(*expHeading >=0 && *expHeading >=
360 - ANGLE_TOLERANCE) + RPS.Heading()) - *expHeading);
                    turn(CW,MOTOR_PERCENT_ON_PULSE, angleOff/2 +
ANGLE_TOLERANCE,temp);
                    numPulsesCW++;
                }
                if(TimeNow() - currTime >= 2){
                    break;
                }
                Sleep(.4);

            }

        }else{

            //Loop until the robot is within a certain threshold or if the
robot pulses back and forth more than twice
```

```
            while ((RPS.Heading() > *expHeading + ANGLE_TOLERANCE ||
RPS.Heading() < *expHeading - ANGLE_TOLERANCE) && !(numPulsesCCW > 2 &&
numPulsesCW > 2))
            {

                //Determine how much the turn was off by and assign the
smaller of the two values to a variable
                 angleOff = fmod(RPS.Heading() - *expHeading + 360.0,360.0) <
abs(RPS.Heading() - *expHeading) ? fmod(RPS.Heading() - *expHeading +
360.0,360.0) : abs(RPS.Heading() - *expHeading);

                 /* Determine which direction to turn in and pulse a fraction
of the angleOff
                * The amount pulse will decrease for each pulse to reduce
pulse count
                * Increment the counters*/

                 if(fmod(RPS.Heading() +angleOff+360.0,360.0) < *expHeading +
.1 && fmod(RPS.Heading() +angleOff+360.0,360.0) > *expHeading - .1){
                     turn(CCW,MOTOR_PERCENT_ON_PULSE, (angleOff/2.) +
1.9,temp);
                     numPulsesCCW++;
                 }else{
                     turn(CW,MOTOR_PERCENT_ON_PULSE,(angleOff/2.) + 1.9,temp);
                     numPulsesCW++;
                 }
                 if(TimeNow() - currTime >= 2){
                     break;
                 }
                 Sleep(.4);

            }
        }
    }
}

/**
* @brief driveStraight
*              general function for the robot driving straight
* @param motor_percent
*              the motor percentage
* @param dist
*              distance robot needs to travel
* @param initCoord
*              array which stores the initial coordinates of the robot
*/
void driveStraight(float motor_percent, float dist, float initCoord[]){

    //Get initial x and y coordinates from RPS and store in initCoords
    initCoord[0] = RPS.X() - xFactor;
    initCoord[1] = RPS.Y() - yFactor;

    //Turn both motors on at given percent motor power.
    left_motor.SetPercent(motor_percent);
    right_motor.SetPercent(motor_percent);
```

```
    //Test to allow the robot to drive over the initial red light rather than
stopping at the light
    if(isStartOfRun){
        SD.Printf("Time: %f\n\t%f\n",TimeNow(), CdS_Cell.Value());
        Sleep(.45);
        isStartOfRun = false;
    }

    //Get the current time and loop for a calculated amount of time
    float currTime = TimeNow();
    SD.Printf("%f: In loop",TimeNow());
    while(TimeNow() - currTime <
abs(dist/(112.19*motor_percent*3.5*PI/6000))){
        SD.Printf("Time: %f\n\t%f\n",TimeNow(), CdS_Cell.Value());
        //If the DDR task isn't done, this will stop the robot once it
detects a light
        if(!hasDoneDDR){
            color = getLightColor();
            if(color != -1){
                break;
            }
        }
    }
    SD.Printf("%f: Out of driving loop",TimeNow());
    LCD.WriteLine("Out of driving loop");
    //Stop both motors if it hasn't already
    right_motor.Stop();
    left_motor.Stop();
}

/**
* @brief pulseDrive
*              function pulses the robot to the correct distance traveled if
it is not within a certain thresholds
* @param initCoord
*              the array that points to the coordinates of the robot before
it started driving
* @param dist
*          distance robots was supposed to have traveled
* @param isForward
*          boolean which stores whether the robot was driving in the positive
x or y-direction
*/
void pulseDrive(float initCoord[], float dist,bool isForward){
    if(RPS.X() >= 0){
        //Counter that keeps track of how many times the robot pulses
forwards or backwards
        int numPulsesForward = 0;
        int numPulsesBack = 0;

        //Temporary array for driveStraight calls
        float temp[2];

        //Stores the how much the robot's distance traveled was off
        float distOff = dist - distance(initCoord[0],initCoord[1], RPS.X()-
xFactor,RPS.Y()-yFactor);
```

```
        //Loop until the robot is within a certain threshold or if the robot
pulses back and forth more than once
        while(abs(distOff) >  DISTANCE_TOLERANCE && !(numPulsesForward >= 2
&& numPulsesBack >= 2) && !(numPulsesBack + numPulsesForward > 5)){
            LCD.WriteLine("In pulse method");
            /* Determine which direction to drive in and pulse a fraction of
the distOff
           * The amount pulse will decrease for each pulse to reduce pulse
count
           * Increment the counters*/

            if(distOff > 0){
                driveStraight(MOTOR_PERCENT_ON_PULSE * pow(-
1,(int)!isForward),abs(distOff/2.) + (2*DISTANCE_TOLERANCE - .01),temp);
                numPulsesForward += 1*isForward;
                numPulsesBack += 1*!isForward;

            }else if(distOff < 0){
                driveStraight(-MOTOR_PERCENT_ON_PULSE * pow(-
1,(int)!isForward),abs(distOff/2.) + (2*DISTANCE_TOLERANCE - .01), temp);
                numPulsesForward += 1*!isForward;
                numPulsesBack += 1*isForward;
            }
            Sleep(.4);
            //Stores the how much the robot's distance traveled was off
            distOff = dist - distance(initCoord[0],initCoord[1], RPS.X()-
xFactor,RPS.Y()-yFactor);
        }
    }
}

void pulseDriveToCoordinate(float expCoord[]){

    int numPulsesForward = 0;
    int numPulsesBack = 0;

    //Temporary array for driveStraight calls
    float temp[2];

    float distOff = distance(expCoord[0],expCoord[1], RPS.X(),RPS.Y());

    float intendedHead[1] = {(atan((expCoord[1] -RPS.Y())/(expCoord[0] -
RPS.X())) * 180.0/PI)};

    pulseTurn(intendedHead);

    //Loop until the robot is within a certain threshold or if the robot
pulses back and forth more than once
    while(abs(distOff) > .5 && !(numPulsesForward >= 2 && numPulsesBack >=
2)){

        /* Determine which direction to drive in and pulse a fraction of the
distOff
       * The amount pulse will decrease for each pulse to reduce pulse count
       * Increment the counters*/
```

```
        driveStraight(MOTOR_PERCENT_ON_PULSE * pow(-1,(int)(expCoord[1] <
RPS.Y())),abs(distOff/2.) + .5,temp);

        if(expCoord[1] < RPS.Y()){
            numPulsesForward++;
        }else{
            numPulsesBack++;
        }

        Sleep(.4);

        //Stores the how much the robot's distance traveled was off
        distOff = distance(expCoord[0],expCoord[1], RPS.X(),RPS.Y());
    }
}

/*
* Navigate with time purely
*/

void driveStraightWithTime(float motor_percent, float t){
    left_motor.SetPercent(motor_percent);
    right_motor.SetPercent(motor_percent);

    Sleep(t);

    right_motor.Stop();
    left_motor.Stop();
}

void turnWithTime(bool isLeft, float motor_percent, float t){
    left_motor.SetPercent(motor_percent * pow(-1,(int)isLeft));
    right_motor.SetPercent(motor_percent * pow(-1,(int)!isLeft));

    Sleep(t);

    right_motor.Stop();
    left_motor.Stop();
}


/**
* @brief driveTo
*          contains the directions that the robot needs to follow to drive to
a certain task location
* @param task
*          an element of the Task enumerations
*/
void driveTo(Task task){
    switch(task){

    case TOKEN:
    {
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];
```

```
        //Lower the arm
        arm_servo.SetDegree(80);
        Sleep(500);

        //Drive straight to get in front of token task and pulse
        driveStraight(MOTOR_PERCENT_ON_LEVEL*2,2,initCoord);
        pulseDrive(initCoord,7.5,true);

        //Turn towards the token task
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,90 - RPS.Heading(),expHeading);
        pulseTurn(expHeading);

        //Drive to the token task
        driveStraight(MOTOR_PERCENT_ON_LEVEL*2,7.75,initCoord);
        pulseDrive(initCoord,9.75,true);

    }
        break;

    case DDR:
    {
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];

        //Pulse so that the robot is straight when it drives
        expHeading[0] = 90.0;
        pulseTurn(expHeading);

        //Drive backwards to get next to the DDR task and pulse
        float y = RPS.Y();
        driveStraight(-MOTOR_PERCENT_ON_LEVEL*2,/*14.25*/ y - yFactor -
1,initCoord);
        pulseDrive(initCoord,/*13.75*/y - yFactor - .6,false);

        //Turn towards the DDR task and pulse
        turn(CW,MOTOR_PERCENT_ON_LEVEL,RPS.Heading() + 15,expHeading);
        expHeading[0] = 0;
        pulseTurn(expHeading);

        //Drive to the first DDR light
        driveStraight(MOTOR_PERCENT_ON_LEVEL*1.25,/*23*/13.25 - RPS.X() +
xFactor,initCoord);
    }
        break;

    case FOOSBALL:{
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];

        //Check the color detected by the DDR
        if(color == 0|| color == -1){

            //Reverse actions to press red buttton
            driveStraight(-MOTOR_PERCENT_ON_LEVEL*.75,1.75,initCoord);
            driveStraightWithTime(-MOTOR_PERCENT_ON_LEVEL*.75, .35);
```

```
            turn(CCW,MOTOR_PERCENT_ON_LEVEL,33.5,expHeading);
            driveStraight(-MOTOR_PERCENT_ON_LEVEL,5,initCoord);
        }else if(color == 1){

            //Reverse actions to press blue button
            driveStraight(-MOTOR_PERCENT_ON_LEVEL*.75,.5,initCoord);
            driveStraightWithTime(-MOTOR_PERCENT_ON_LEVEL*.75, .55);
            turn(CCW,MOTOR_PERCENT_ON_LEVEL,35,expHeading);
            driveStraight(-MOTOR_PERCENT_ON_LEVEL*1.25,8.75,initCoord);
        }


        //Turn towards the ramp
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,45,expHeading);
        expHeading[0] = 45.0;
        pulseTurn(expHeading);

        //Drive forward til the y-coordinate reaches a certain value and
pulse
        float y = RPS.Y();
        driveStraight(MOTOR_PERCENT_ON_LEVEL*1.5,3,initCoord);
        pulseDrive(initCoord, 1.5 + ((/*12*/3 - y +
yFactor)/sin(RPS.Heading() * PI/180.)),true);

        //Turn to get perpendicular to the ramp and pulse
        turn(CW,MOTOR_PERCENT_ON_LEVEL, 60,expHeading);
        expHeading[0] = 0;
        pulseTurn(expHeading);

        //Drive to get in front of the ramp and turn to face the ramp
        driveStraight(MOTOR_PERCENT_ON_LEVEL*1.5,.75*(/*27.75*/ 18.5-
RPS.X()+xFactor),initCoord);
        pulseDrive(initCoord,23.5 - RPS.X() + xFactor,true);


        turn(CCW,MOTOR_PERCENT_ON_LEVEL,90,expHeading);
        expHeading[0] = 85.0;
        pulseTurn(expHeading);


        //Drive up the top of the ramp
        driveStraight(MOTOR_PERCENT_ON_INCLINE*1.5,43.5,initCoord);

        turn(CCW,MOTOR_PERCENT_ON_LEVEL,25,expHeading);

        //Drive straight and turn back to 90 degrees to drive towards
foosball
        driveStraight(MOTOR_PERCENT_ON_LEVEL,3.5,initCoord);
        turn(CW,MOTOR_PERCENT_ON_LEVEL,25,expHeading);
    }
        break;

    case LEVER:
    {
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];
```

```
        //Drive away from foosball and turn towards the right side of the
obstacle
        driveStraight(-MOTOR_PERCENT_ON_LEVEL,3.15,initCoord);
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,70,expHeading);

        //Get arm ready for knocking down the obstacle and drive towards the
obstacle
        //and turn to get on the right side of the obstacle

        arm_servo.SetDegree(100);
        Sleep(.50);
        driveStraight(MOTOR_PERCENT_ON_LEVEL,7.5,initCoord);
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,30,expHeading);

        //Knock down the obstacle
        arm_servo.SetDegree(80);
        Sleep(.50);
        platform_servo.SetDegree(50);
        Sleep(.5);
        arm_servo.SetDegree(30);
        Sleep(.5);
        driveStraight(MOTOR_PERCENT_ON_LEVEL,1,initCoord);
        platform_servo.SetDegree(130);
        Sleep(.5);

        DRIVING_STATE;

        //Drive to the lever
        driveStraight(MOTOR_PERCENT_ON_LEVEL,4.25,initCoord);

    }
        break;

    case BUTTON:
    {
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];
        //driveStraight(-MOTOR_PERCENT_ON_LEVEL,1.5,initCoord);

        //Perform a three point turn to face the final button
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,40,expHeading);

        driveStraight(MOTOR_PERCENT_ON_LEVEL,3.5,initCoord);
        turn(CCW,MOTOR_PERCENT_ON_LEVEL,50,expHeading);

        driveStraight(MOTOR_PERCENT_ON_LEVEL,12,initCoord);

        if(RPS.Heading() > 0){
            turn(CCW,MOTOR_PERCENT_ON_LEVEL,abs(270 -
RPS.Heading()),expHeading);
            expHeading[0] = 270.0;
            pulseTurn(expHeading);
        }else{
            turn(CCW,MOTOR_PERCENT_ON_LEVEL,20,expHeading);
        }
```

```
        while(RPS.Heading() < 0){
            driveStraight(MOTOR_PERCENT_ON_LEVEL,1,initCoord);
        }
        Sleep(.35);

        //Perform RPS check before going down the ramp
        if(RPS.Heading() > 0){
            expHeading[0] = 270.0;
            pulseTurn(expHeading);
        }

        //Drive to final button and face the button
        driveStraight(MOTOR_PERCENT_ON_LEVEL*1.5,40,initCoord);
        turn(CW,MOTOR_PERCENT_ON_LEVEL,45,expHeading);

    }

        break;

    default:

        LCD.WriteLine("This message should not appear. You goofed.");

    }
}

/**
* @brief complete
*          functions which tells the robot what to do to complete the task
* @param t
*          an element of the Task enumeration
*/
void complete(Task t){

    switch(t){

    case TOKEN:
        //Move the arm above the slot and open claw to drop the coin
        platform_servo.SetDegree(52);
        Sleep(500);
        claw_servo.SetDegree(110);
        Sleep(500);
        claw_servo.SetDegree(0);

        break;

    case DDR:
    {
        //Initialize arrays to store values in drive and turn methods
        float initCoord[2];
        float expHeading[1];

        //Get color of light twice to ensure that the color is right
        color = getLightColor();
        driveStraightWithTime(20,.1);
        color = getLightColor();
```

```
        //Do an RPS check to make sure the robot is parallel to side of the
course
        expHeading[0] = 0.0;
        pulseTurn(expHeading);

        //Press the button based on color detected
        if(color != 1){
            if(color == 0){
                LCD.Clear(RED);
            }else{
                LCD.Clear(BLACK);
                LCD.WriteLine("CdS cell didn't read anything. It goofed.");
            }

            driveStraightWithTime(-20,.2);

            turn(CW,MOTOR_PERCENT_ON_LEVEL,34,expHeading);
            driveStraightWithTime(MOTOR_PERCENT_ON_LEVEL*.75, .75);
            driveStraight(MOTOR_PERCENT_ON_LEVEL*.75,4.,initCoord);

            Sleep(5.0);

        }else {
            LCD.Clear(BLUE);

            driveStraightWithTime(MOTOR_PERCENT_ON_LEVEL,.3);
            driveStraight(MOTOR_PERCENT_ON_LEVEL,1.85,initCoord);
            turn(CW,MOTOR_PERCENT_ON_LEVEL,35,expHeading);
            driveStraightWithTime(MOTOR_PERCENT_ON_LEVEL*.75, .5);
            driveStraight(MOTOR_PERCENT_ON_LEVEL*.75,3,initCoord);

            Sleep(5.0);
        }
        hasDoneDDR = true;

    }
        break;

    case FOOSBALL:
    {
        float initCoord[2];
        float expHeading[1];
        //Get the foosball scoring disk in the grasp of the robot's claw
        arm_servo.SetDegree(60);
        Sleep(250);
        platform_servo.SetDegree(75);
        Sleep(500);
        arm_servo.SetDegree(0);
        Sleep(500);
        claw_servo.SetDegree(100);
        Sleep(500);
        if(color != 1){
            driveStraight(MOTOR_PERCENT_ON_LEVEL,.75,initCoord);
        }else{
            driveStraight(MOTOR_PERCENT_ON_LEVEL,.75,initCoord);
        }
        Sleep(500);
```

```
        platform_servo.SetDegree(30);
        Sleep(500);
        claw_servo.SetDegree(30);
        Sleep(500);

        //Back the robot up a bit so the arm can extend and then have the arm
complete the task

        for(int i = 4; i < 9;i++){
            platform_servo.SetDegree(10+10*i);

        }
        Sleep(250);
        driveStraight(-MOTOR_PERCENT_ON_LEVEL,.5,initCoord);

        claw_servo.SetDegree(90);
        driveStraight(-MOTOR_PERCENT_ON_LEVEL,1,initCoord);
        arm_servo.SetDegree(35);
        Sleep(500);
    }
        break;

    case LEVER:
    {
        float initCoord[2];
        arm_servo.SetDegree(60);
        Sleep(.5);
        platform_servo.SetDegree(10);
        Sleep(.5);
        driveStraight(MOTOR_PERCENT_ON_LEVEL,2.5,initCoord);
        arm_servo.SetDegree(15);
        Sleep(.5);
        arm_servo.SetDegree(60);
        for(int i = 3; i < 8;i++){
            platform_servo.SetDegree(5*i);
            Sleep(.5);
            arm_servo.SetDegree(30);
            Sleep(.5);
            arm_servo.SetDegree(50);
            Sleep(.5);
            platform_servo.SetDegree(15);
            Sleep(.5);
            arm_servo.SetDegree(60);
            Sleep(.5);
        }
    }
        break;

    case BUTTON:
    {
        float initCoord[2];
        driveStraight(MOTOR_PERCENT_ON_LEVEL*1.5,5,initCoord);
        LCD.WriteLine("CONGRATS!!! The robot reached the end of the
course.");
    }
        break;
```

```
    default:

        LCD.WriteLine("This message should not appear. You goofed.");

    }
    Sleep(500);
    DRIVING_STATE
}


/**
* @brief initialRobotState
*                  initialize the robot so that it meets the requirements of
the starting position
*/
void initialRobotState(){

    /*
   * Set up servos and have them in starting state
   */

    arm_servo.SetMin(ARM_SERVO_MIN);
    arm_servo.SetMax(ARM_SERVO_MAX);

    platform_servo.SetMin(PLATFORM_SERVO_MIN);
    platform_servo.SetMax(PLATFORM_SERVO_MAX);


    claw_servo.SetMin(CLAW_SERVO_MIN);
    claw_servo.SetMax(CLAW_SERVO_MAX);

    arm_servo.SetDegree(INITIAL_ARM_ANGLE);
    Sleep(.5);
    claw_servo.SetDegree(INITIAL_CLAW_ANGLE);
    Sleep(.5);
    platform_servo.SetDegree(INITIAL_PLATFORM_ANGLE);
}



/**
* @brief finalAction
*             start menu that will act as final action
*/
void finalAction(){

    LCD.Clear(BLACK);

    FEHIcon::Icon start_button;

    start_button.SetProperties("READY", 120, 95, 80, 50, WHITE, WHITE);

    start_button.Draw();


    float xTouch, yTouch;
```

```
    LCD.Touch(&xTouch,&yTouch);

    while(!start_button.Pressed(xTouch,yTouch,1)){
        LCD.Touch(&xTouch,&yTouch);

    }

    LCD.Clear(BLACK);
    LCD.WriteAt("Set",120, 95);

    /*
 * Set up correction factors
 */
    xFactor = RPS.X();
    yFactor = RPS.Y();
}

/*
* Testing methods
*/

/**
 * @brief testingServo
 *          specific testing method to test a servo
 * @param s
 *          servo to be tested
 */
void testingServo(FEHServo s){

    float deg = 0.00;

    int i = 0;

    s.SetDegree(deg);

    Sleep(1.0);

    while(deg <= 180.00){

        LCD.WriteAt(deg,175,10 + i*10);

        s.SetDegree(deg);

        deg+=10;

        Sleep(1.0);

    }

}
/**
 * @brief testMethods
 *          menu which allows for different parts of robot to be tested
 */
void testMethods(){

    LCD.Clear(BLACK);
```

```
FEHIcon::Icon turn_button;
turn_button.SetProperties("TURN", 10, 20, 80, 50, WHITE, WHITE);
turn_button.Draw();

FEHIcon::Icon arm_button;
arm_button.SetProperties("ARM", 10, 80, 80, 50, WHITE, WHITE);
arm_button.Draw();

FEHIcon::Icon claw_button;
claw_button.SetProperties("CLAW", 10, 140, 80, 50, WHITE, WHITE);
claw_button.Draw();

FEHIcon::Icon platform_button;
platform_button.SetProperties("PLATFORM", 10, 200, 80, 50, WHITE, WHITE);
platform_button.Draw();

FEHIcon::Icon drive_button;
drive_button.SetProperties("DRIVE", 100, 20, 80, 50, WHITE, WHITE);
drive_button.Draw();

while(true){
    float x, y;
    LCD.Touch(&x,&y);
    if(turn_button.Pressed(x,y,0)){
        LCD.Clear(BLACK);
        float eh[1];
        LCD.WriteLine(RPS.Heading());
        LCD.WriteLine("");
        turn(CCW,30,90,eh);
        Sleep(.5);
        LCD.WriteLine("Turn");
        pulseTurn(eh);
        LCD.WriteLine("Pulsed");

        LCD.WriteLine(RPS.Heading());
        LCD.WriteLine("");
        turn(CCW,30,90,eh);
        Sleep(.5);
```

```
            LCD.WriteLine("Turn");
            pulseTurn(eh);
            LCD.WriteLine("Pulsed");

            LCD.WriteLine(RPS.Heading());
            LCD.WriteLine("");
            turn(CCW,30,90,eh);
            Sleep(.5);

            LCD.WriteLine("Turn");
            pulseTurn(eh);
            LCD.WriteLine("Pulsed");

            LCD.WriteLine(RPS.Heading());
            LCD.WriteLine("");
            turn(CCW,30,90,eh);

            LCD.WriteLine("Turn");
            pulseTurn(eh);
            LCD.WriteLine("Pulsed");

        }else if (arm_button.Pressed(x,y,0)) {

            testingServo(arm_servo);

        }else if (claw_button.Pressed(x,y,0)) {

            testingServo(claw_servo);

        }else if (platform_button.Pressed(x,y,0)) {

            testingServo(platform_servo);

        }else if (drive_button.Pressed(x,y,0)){
            LCD.Clear(BLACK);
            float initCoord[2];
            driveStraight(30,7.5,initCoord);
            LCD.WriteLine("Drove forward");
            pulseDrive(initCoord,7.5,true);
            LCD.WriteLine("Pulsed");
            Sleep(1.0);
            driveStraight(-30,7.5,initCoord);
            LCD.WriteLine("Drove backward");
            pulseDrive(initCoord,7.5,false);
            LCD.WriteLine("Pulsed");
        }

    }

}

/*
* Main function
*/

int main(void){
```

```
    //Initialize robot

    initialRobotState();

    //Initialize RPS and open SD log

    RPS.InitializeTouchMenu();
    SD.OpenLog();

    //Perform final action

    finalAction();

    float currTime = TimeNow();
    while(getLightColor() != 0 || TimeNow() - currTime < 30);

    //Starting message
    LCD.Clear(BLACK);
    LCD.WriteAt("Go",120, 95);

    //Drive to and complete token
    driveTo(TOKEN);
    complete(TOKEN);

    //Drive to and complete DDR
    driveTo(DDR);
    complete(DDR);

    //Drive to and complete foosball
    driveTo(FOOSBALL);
    complete(FOOSBALL);

    //Drive to and complete lever
    driveTo(LEVER);
    complete(LEVER);

    //Drive to and complete final button
    driveTo(BUTTON);
    complete(BUTTON);

    //Close SD log
    SD.CloseLog();
}
```