R10: Report Second Draft

Engineering 1282H

Spring, 2015

Harper 3:00

G9: The Copperheads

Mark Luke

Edward Hughes

Nader Elkurd

George Hawkins

Executive Summary

Robots are useful for completing tasks in conditions a human cannot safely operate. A group of engineers and researchers in the Arctic area are attempting to update the oil infrastructure but are unable to do so because of extreme weather. OSURED has been commissioned to select a prototype robot to do the tasks the engineers need in arctic weather. This robot needed to be built with a 160$ budget, operated autonomously, and fit in a 9” by 9” by 12” box. The robot also needed to pick up a salt bag and move it into a garage, turn a crank, press a set of buttons, and toggle a switch. The G9 Copperheads designed and tested a robot to fulfill the conditions required by OSURED.

To complete these objectives, the roles and responsibilities of Documenter, Builder, and Coder were split among 4 team members. The robot was designed and built over the course of three months. The final design included a tread based drivetrain, a dish and prong for a crank turner and switch toggle, a dragger for moving the salt bag, and a branched tower for button pressing. The deadlines included performance tests that checked the ability for the robot to deal with specific obstacles and evaluations that included two competitions that required the robot to complete an entire course of obstacles. To complete these objectives, the roles and responsibilities of Documenter, Builder, and Coder were split among 4 team members.

The G9 Copperhead robot performed adequately at each deadline and at the first competition. The final competition results were far worse than the previous tests had indicated. However this was not due to design flaws but rather execution, both of construction and of coding. In order to improve in future production, using lasercut or machined parts instead of manually created pieces should greatly increase reliability.

Table of Contents

List of Figures

List of Tables

1. Introduction

In environments of extreme weather, sending a robot is often the safest and most reliable option. A group of engineers and researchers in the Arctic area are attempting to update the oil infrastructure to deal with increased demands. Due to the weather conditions, the main power system has been disabled which is unacceptable for working conditions within the base of operations. OSURED has been commissioned to select a prototype robot to deal with the infrastructure problems faced by the engineers in the arctic tundra. The main objective is to build a self-controlled, self-contained, and self-propelled robot. This report outlines the development and functionality of the robot built by the G9 Copperheads, an OSURED design team. Documentation was done by George Hawkins, Nader Elkurd, and Edward Hughes. Building and Electrical were done by Mark Luke and George Hawkins. Programming was done by Mark Luke and Edward Hughes. The project was started February 3rd, 2015

The Preliminary Concepts section presents initial ideas brainstormed by the group. The Analysis, Testing and Refinements section presents the changes and adjustments made during testing of the robot, both of coding and physical design. The Individual Competition section presents the performance at April 3rd. The Final Design section presents the final form of the prototype which was used at the final competition. The Final Competition section presents the performance of the robot on April 11th. The Summary and Conclusion section presents a summary of the development, functions, and performances of the Copperhead robot.

2. Preliminary Concepts

The robot will be required to do several tasks in a timely and safe manner within certain physical boundaries. The course diagram is Figure 1 below.

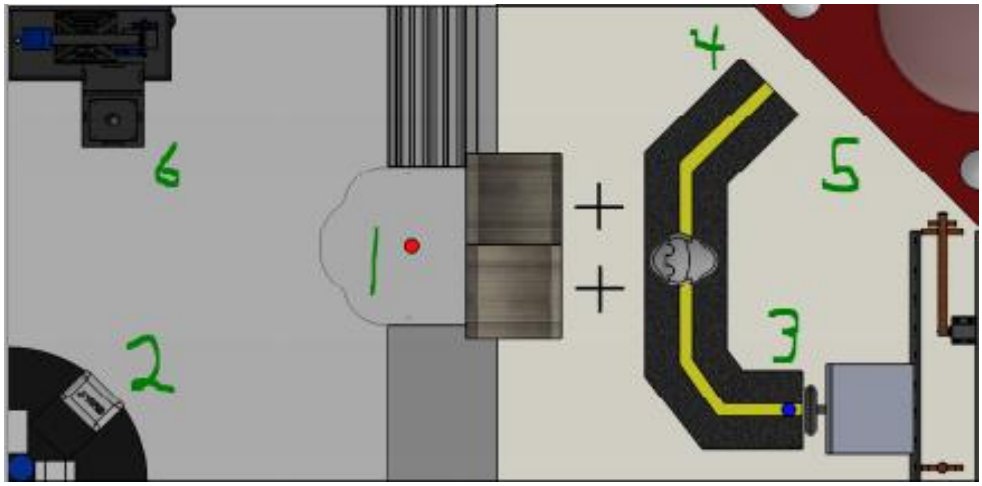


Figure 1: An overhead view of the course with the locations of each task

It must first start by detecting the start signal which is a light on the ground which can be found at point 1 on the course. Then the robot must, in no particular order, pick up the salt bag at point 2, turn the crank 360 degrees at point 3, drop off the salt bag at point 4, press three buttons at point 5. Finally, the robot must end the run by toggling a switch at point 6. In addition to those tasks, the robot must avoid the trees next to the road and be able to deal with the snow on the road.

To help with these tasks and the navigation required to move the robot, the project site has a Robot Positioning System (RPS). To interact with the RPS system, the robot will have a QR code attached on top of the vehicle. The QR code, in addition to the RPS system, will provide the

robot with a coordinate location and heading. The course also has specific floor coloring that can be used as landmarks as well as walls with which the robot can orient itself.

There are also many constraints that the robot had to follow in order to be qualified for competition. The budget of the vehicle was $160. Any over-budget spending was paid for out of pocket and resulted in a lower score. All parts of the robot had to be accounted for within the budget. The controller for the robot was a programmable Proteus. Any modification or self-repair to the Proteus was prohibited. The dimensions of the robot must fit within a 9 by 9 by 12 inch box. The robot was not allowed to be controlled from outside the course.

Before the group first started working together, each individual member brainstormed on their own to come up with solutions for the obstacles that were faced by the robot. Each of these ideas were judged on the criteria of cost, ease of construction, feasibility, and effectiveness. Each of the criteria were given a score from 0 to 3 and the sum was used to choose the top three. The decision matrices can be found in the Appendices.

2.1. Chassis

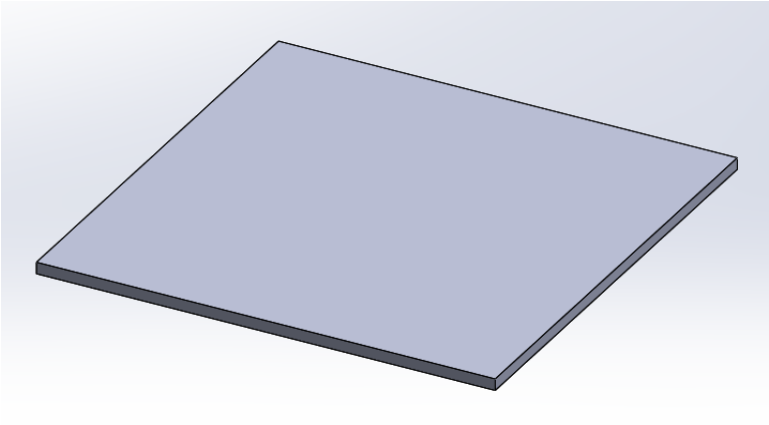
The ideas presented for the chassis were as follows.

- Rectangular platform with space on the side for wheels
- Square platform with the drivetrain on the bottom
- Rolling cage with space on the side for wheels
- Rectangular skeleton frame with space on the side for wheels

- Octagonal skeleton frame with the drivetrain on the bottom
- Box with space on the side for wheels

The box, rolling cage, and both skeleton frames were ruled out due to the higher difficulty of construction. In addition the rolling cage and skeleton frames had high costs and made adjustments very difficult. The square and rectangular platforms tied for the score of 11. The decision between a square or rectangular platform came down to wheel placement since both ideas were very similar in all other ways. The rectangular was chosen because it was thought that the wheels/tread would need space on the sides to fit. The material was chosen to be wood because of how easy it is to adjust.

Figure 2: Possible chassis



2.2. Drivetrain

The ideas presented for the drivetrain were as follows.

- 4 wheeled
- 2 wheeled back drive
- 2 wheeled front drive

- Treads
- 2 sets of 4 wheeled in different directions

The 4 wheeled and 2 sets of 4 wheeled were immediately ruled out due to their extremely high costs. The treads had a higher cost but the wheels were more difficult to execute. The treads had the highest score of 9 and was chosen due to the ease of execution. Another advantage with treads is that turning with treads requires no adjustment of positioning because of the centered placement of the treads.

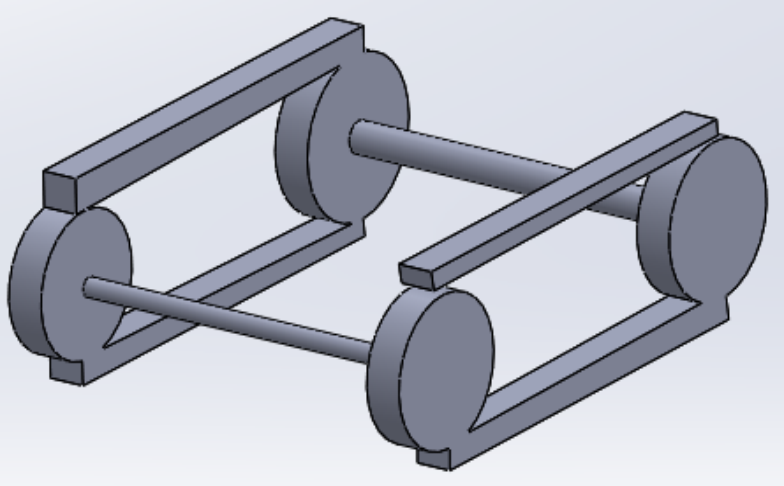


Figure 3: Possible drivetrain

2.3. Button

The ideas presented for the button pressing mechanism were as follows.

- Angling rod
- 3 pronged puncher
- Rotating dish with arm

- Claw
- A block that presses all three at once

The block that presses all three buttons at once was ruled out due to the fact that it could only press the buttons in one order. The claw was ruled out because of the difficulty and cost of the idea. The rotating dish with a prong was dismissed due to a lack of space. The 3 pronged puncher was chosen over the angling rod because of how easy it would be to program. Although the cost was high, the 3 pronged puncher still resulted in the highest score of 9

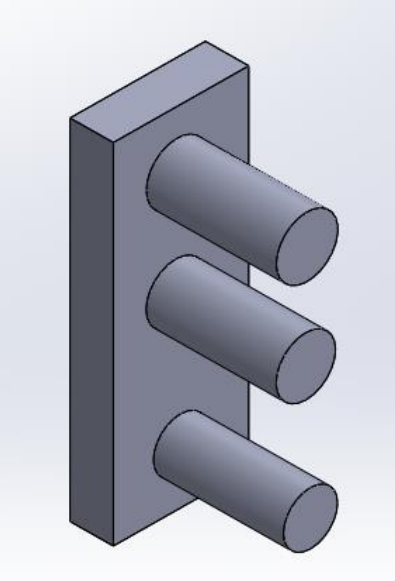


Figure 4: Possible button pushing mechanism

2.4. Crank

The ideas presented for the crank turning mechanism were as follows.

- Rotating dish with arm

- Rotating plunger
- 2 prong jaw that grabs the center
- Claw
- Pentagon that would fit the crank
- Rotating forked arm.

The 2 prong jaw was eliminated because of the difficulty of execution and high costs. The claw was also eliminated because of high costs. The pentagon was eliminated due to the small margin for error that method required. The rotating plunger was scrapped due to the high chance for failure or inconsistency. Both the rotating dish and rotating forked arm scored a 7 but the rotating dish's chance of being used for another task resulted in a much higher score of 10. The plan was to attach it to a hacked servo which would rotate it. The chosen material for this was 3d printing.

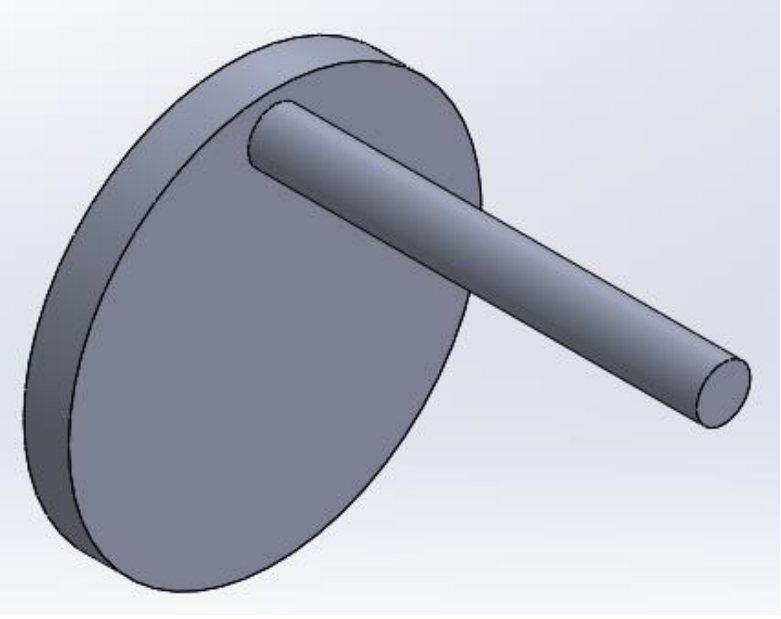


Figure 5: Possible crank turning mechanism

2.5. Saltbag

The ideas presented for the saltbag mechanism were as follows.

- Claw/clamp
- Catapult
- Dumptruck style grabber
- Plow
- Angleable Scoop

The high levels of complexity for the first three ideas immediately ruled them out. Both the plow and angleable scoop were easy to execute and were cheap. However, the angleable scoop had an advantage with dealing with ramps and fitting vertically in the robot. The scoop scored an 11. The material for the scoop was chosen to be wood. The scoop would be lifted with some kind of winch mechanism that utilized a hacked servo.

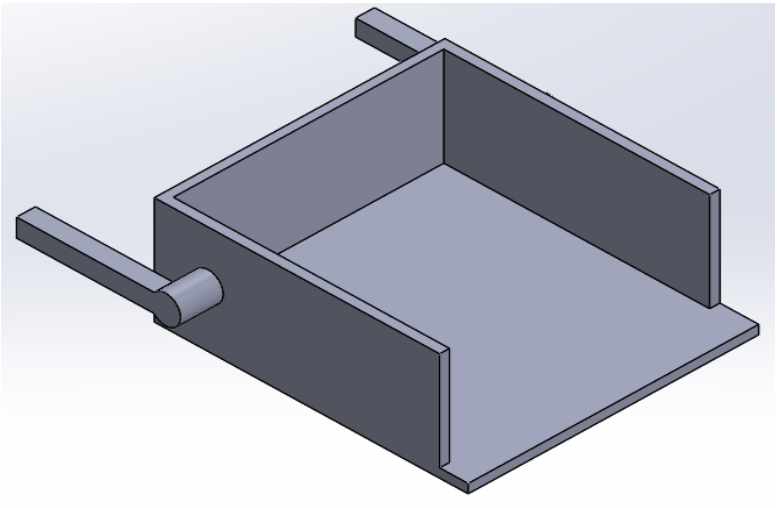


Figure 6: Possible salt bag transportation mechanism

2.6. Switch

The ideas presented for the switch mechanism were as follows.

- Rotating arm
- Stationary arm
- Sliding arm
- Swiveling two pronged arm
- Rotating dish with arm
- Lasso
- Wrecking ball
- Claw

The claw, wrecking ball, and lasso were eliminated due to their high degree of difficulty. The swiveling two pronged arm was eliminated because it required a high degree of precision. The sliding arm was eliminated because no clear idea on how to execute it was found. The stationary arm was eliminated because it was not space efficient and required extra movement by the robot. The rotating arm and rotating dish tied with a score of 11 but the rotating dish was chosen because of the chance it could be used for another task. A picture of a brainstormed switch toggler can be found in Figure 5.

2.7. Routes

The routes were judged based on different criteria. The criteria were speed, turning, and difficulty. The ideas presented for the routes were as follows.

1. Pickup > Smooth > Crank > Buttons > Drop > Rough > Pump
2. Pickup > Smooth > Crank > Buttons > Drop > Smooth > Pump
3. Pickup > Smooth > Drop > Buttons > Crank > Smooth > Pump
4. Pickup > Rough > Drop > Buttons > Crank > Smooth > Pump
5. Rough > Buttons > Crank > Smooth > Pickup > Rough > Drop > Smooth > Pump
6. Pickup > Rough > Drop > Crank > Button > Smooth > Pump
7. Pickup > Drop > Smooth > Crank > Button > Smooth > Pump

The 7th route was dropped because of the high degree to turning required. The 6th route was dropped because it required too much precise navigation in the same area. The 5th route was eliminated because of how long the route would take, and even though it cut down on turning the cost was not worth the benefits. The 4th route caused a large amount of backtracking and inefficiency. The 3rd was also not quite as efficient as possible. The 2nd was chosen over the first because it avoided the rough ramp, was still very efficient and short, and still cut down on turning. The chosen route can be found in Figure 7 below.

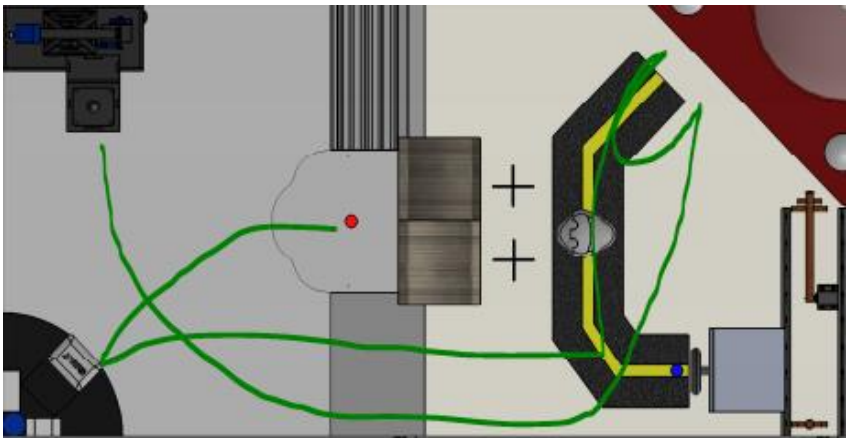


Figure 7: Planned route on the course

2.8. First Prototype

A sketch was created to represent the basic placement of the mechanisms planned for the prototype. Figure 8 below is an image of that sketch

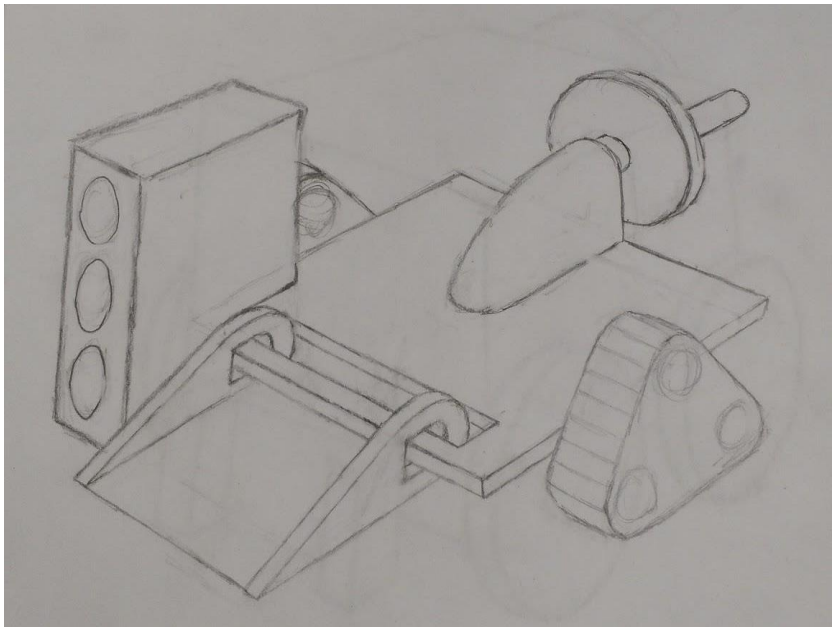


Figure 8: Planned design for robot

A cardboard mock-up was then created represent the initial prototype. Figure 9 on the next page is an image of that mockup.

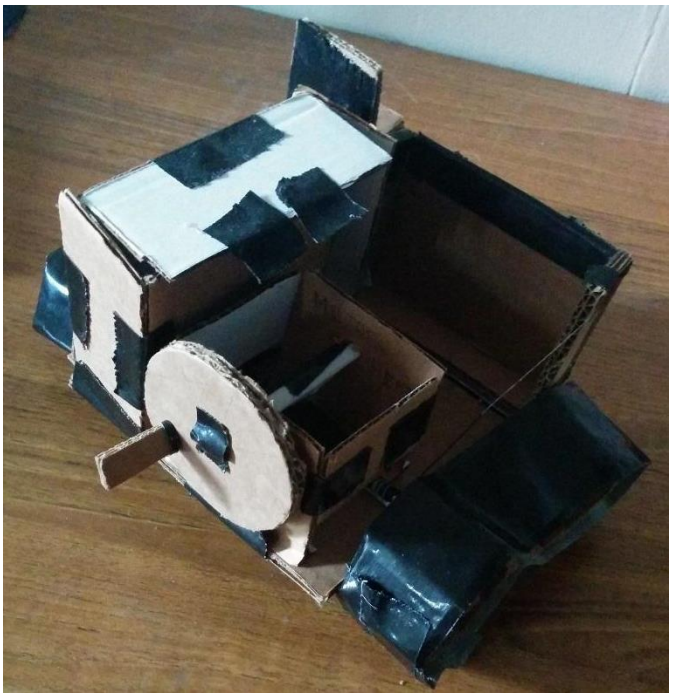


Figure 9: Cardboard mock-up of the robot

A pseudocode was written as a baseline of how the robot would complete the tasks. The pseudocode can be found in the Appendix. A flowchart was also created to represent how the robot would complete the tasks. That can also be found in the Appendix.

3. Analysis, Testing and Refinements

Once the mock-up was finished, the building of the robot commenced. The drivetrain and chassis were the first parts to be built. This included the shaft encoding and a frame for the QR code. Then a CDS sensor was installed to help the vehicle start when the light on the floor turned on. Other sensors including shaft encoders and microswitches were also installed. Then the button presser was installed. Then the saltbag scoop was installed. Then the crank turner and

switch toggler was installed. With each new part added to the robot, the code was updated to deal with the new mechanisms.

3.1. Analysis

Some details needed to be considered before the building of the robot could start. Using a Solidworks mockup calculations of the weight, torque, and placements of mechanisms were made.

3.1.1. Drivetrain Analysis

An estimate of the distance the robot would need to travel was determined by plotting the route on an aerial view of the course in Solidworks. The speed required to travel this route while completing all tasks within the time limit of 2 minutes was calculated. The mass of the finished robot was estimated by creating a solidworks model with the correct materials, using solidworks to calculate that mass, and adding the mass of the proteus. This mass was used to calculate the torque the motor would need to get up the ramp on the course. The calculations, finished February 16th, are shown below.

Distance = 243.3 inches

Radius of wheels = 1 inch

Allotted time to complete course = 75 seconds

Speed = 243.3/75 = 3.24 inches per second

Rotational Speed = Speed/Circumference = 3.24/2π = .557 rps = 33.4 rpm

Weight of Robot = 9.17 lbs

Internal Friction = .5 lbs

Height = 3 inches

Length = 8 inches

$\Sigma F = w*sin(\theta) + F_{int}$

$Tan(\theta) = h/l$

$Sin(\theta) = h/\sqrt{(h^2+l^2)}$

$\Sigma F = w*h/\sqrt{(h^2+l^2)} + F_{int}$ **$= 9.17*3/\sqrt{(73)} +.5 = 3.721bs$**

$\tau = F*r = 3.72 *1 = 3.72$ *lb-in = 59.5 oz-in*

$\tau/2 = 29.75$*oz-in*

Calculations for 10% error

*33.4 rpm * 1.1 = 36.7 rpm*

$\tau * 1.1 = 32.8$*oz-in*

The speed and torque were plotted on a graph of the torques and speeds of the possible motor choices and a motor was chosen from those with the most usable power above the required torque and speed. The plot can be found in Figure 10 below with the black lines representing the required torque and speed.

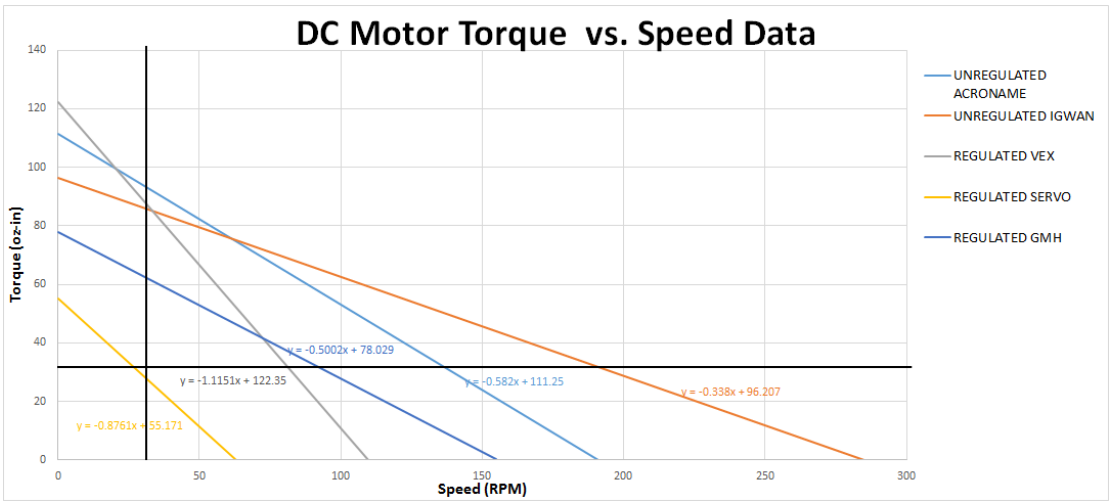
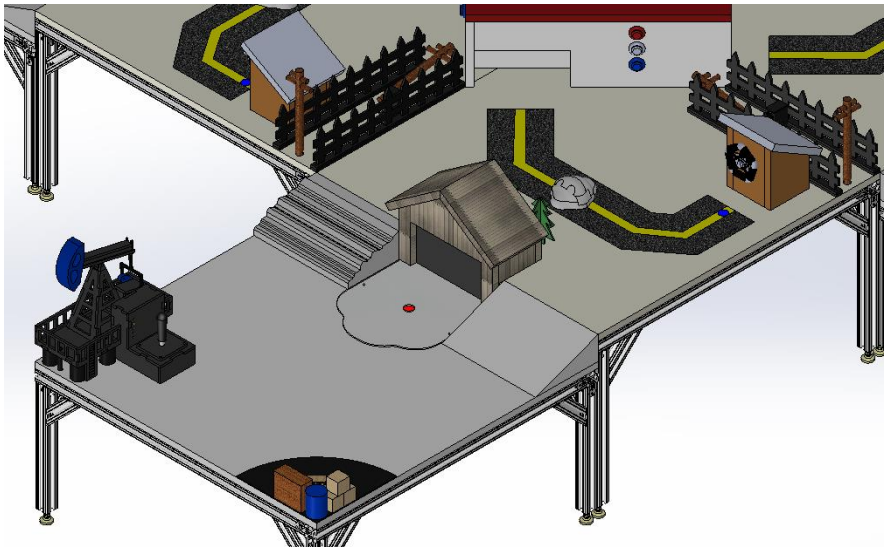


Figure 10 : Torques and speeds of available motors

These calculations were used to determine which motors would be used on the robot. From the graph, it was determined that a hacked servo would be the only motor incapable of driving the robot. The Igwan and Acroname motors had the most usable power available. Since the Acroname motors were significantly cheaper than the Igwans, it was decided that these would be the motors used on the robot.

3.1.2. Placement of Mechanisms

On February 18th, the obstacles on the course were measured to discover the exact dimensions the mechanisms and robot needed to be. Figure on the next page is a picture of the course with all the obstacles visible.



The center of the crank was 4.5 inches off the ground. In order for a prong to fit into the gaps between the spokes, the prong would have to be placed from 0.7 to 1.15 inches from the center of the crank. The crank also juts out 1.3 inches from the wall behind it and is .625 inches thick. The RPS coordinates of the light in front of the crank are (30.5,57.5).

The buttons are beveled with an inner diameter of .75 inches and an outer diameter of 1.25 inches. Their centers are spaced 1.75 inches apart and the bottom one is 2.5 inches from the ground.

The oil pump switch is 3.5 inches tall. At its largest point, which is approximately 3 inches high, it has a 0.5 inch radius. The switch is placed on a box that is 4.5 inches wide, 2 inches tall, and 4 inches long. The switch itself is placed 2 inches back from the middle.

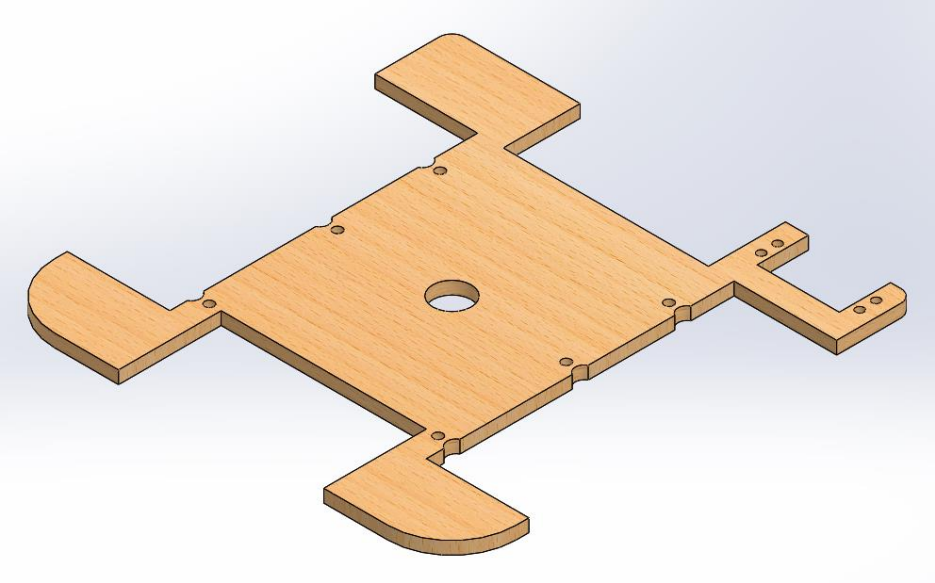
The saltbag is 3.5 inches wide, 1 inches tall, 2.75 inches long. The garage it is supposed to go into is 8.75 inches wide and 4.125 inches tall.

3.2. Building

The building of the robot was divided into three major sections: shopwork, construction, and electrical. The majority of the shopwork and electrical work was done by Mark Luke. The majority of the construction was done by George Hawkins.

3.2.1. Chassis and Drivetrain

The laminated wood bought was 12 by 12 and was a quarter inch thick. This was then cut into a 9 by 9 piece to maximize the parameters. The rectangular chassis established in the brainstorming was scrapped due to the fact there were no reasons to cut the wood unnecessarily. The wood for the chassis was cut by Mark Luke to the shape in Figure 1 below for the following reasons.



The treads that were bought could be spread into an equilateral triangle with a perimeter of 18 inches. The treads were also approximately 1.25 inches wide. To make sure none of the treads would get caught on the chassis, a 6” by 2” cut was made on the center of each side of the platform. The crank was 4 inches wide at its largest point. The oil switch was 2 inches away from the front of the box it lies on. In order to use the same rotating dish mechanism for both tasks, a 4 by 2 cut was made to the back to the platform. Finally, in order to have a scoop that can pick up the saltbag to fit in the 9 by 9 parameters, a 4 by 1.5 cut was made into the front of the chassis platform. Once the platform was finished holes were drilled to place erector set pieces in certain positions. These positions, as shown in Figure 1, allowed the Acroname motors to be placed to create an equilateral triangle. This was completed February 23rd. They also allowed the placements of two additional axles that would hold the two other set of wheels that would complete the triangle. Once the drivetrain, chassis, and QRCode holder were assembled on February 26th, it was found that there was slack in the treads that prevented the robot from moving. This was due to the fact that the Solidworks planning used an incorrectly dimensioned erector set piece. This was fixed by placing a piece of wood that pushed one axle down to

increase tension. However due to the increased tension, the Erector set pieces being used to hold the motors in place were being pulled horizontally. To fix this, a bottom platform was attached to the undersides to the motors to hold them a certain distance apart. This was completed February 26th. At this point, the vehicle could move but had no shaft encoding for control. For shaft encoding, a large gear was attached to the axle and wheel and another smaller gear was attached to a rotary mechanical shaft encoder that was attached to the top of the motor. A small wooden housing was also created to keep the gears in contact. The shaft encoding was finally finished March 8th. As time passed, the treads started to stretch so the top axle was moved up to keep tension. This resulted in the QRCode holder becoming non adjustable as well as slightly shorter. This was completed March 25th. As testing continued, the treads consistently rode on top of the snow causing problems. In order to deal with that, snow guards were installed on April 2nd. Figure 11 is a Solidworks image of the drivetrain and chassis without any mechanisms.

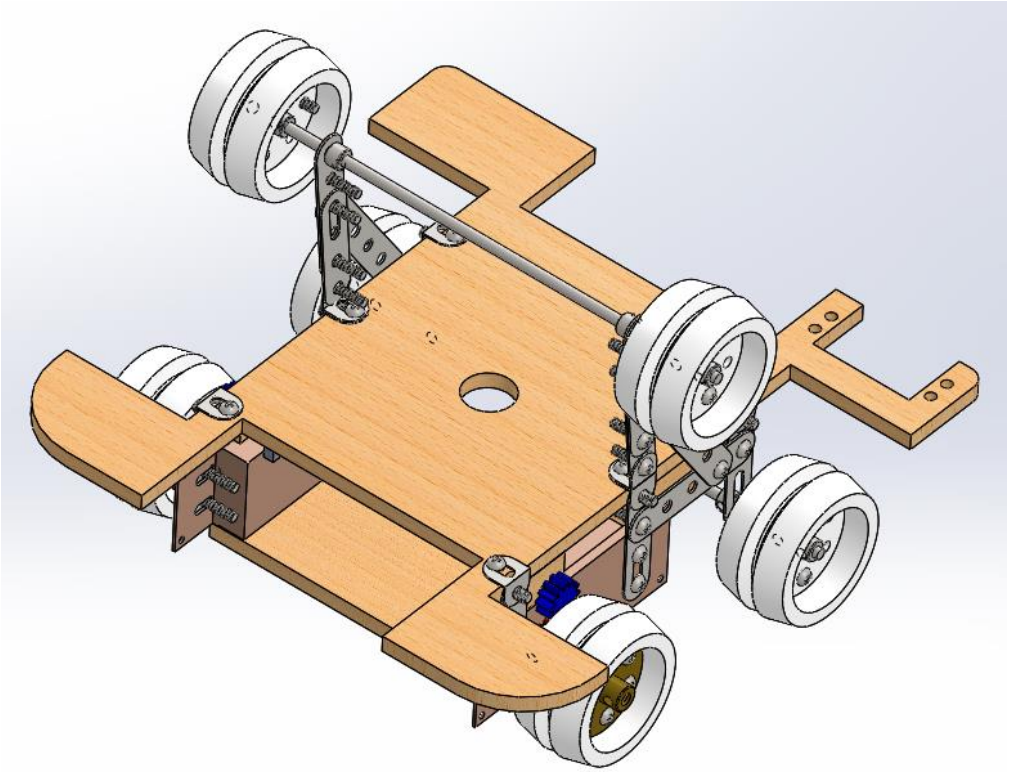


Figure 11 : The drivetrain and chassis without mechanisms

The mechanisms were then placed on the drivetrain and chassis. Figure 12 below shows the placement of all the servo motors for each mechanism. The wiring diagram for the robot can be found in Appendix.

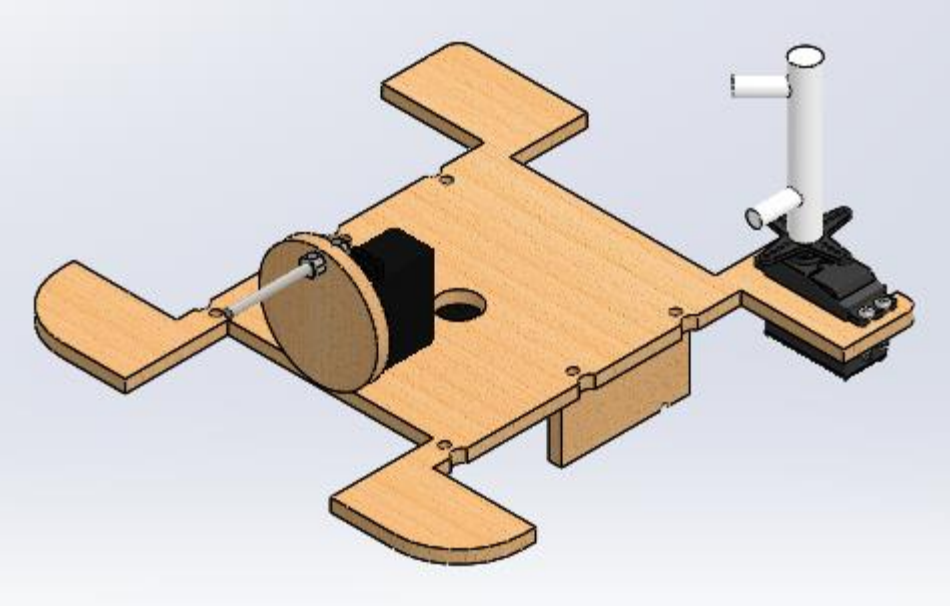
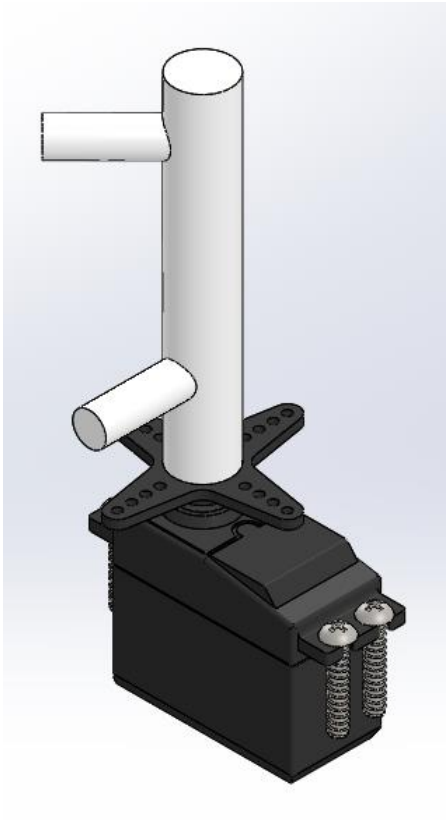


Figure 12: Assembly containing the servo placements. Winch servo is underneath the platform

3.2.2. Button Mechanism

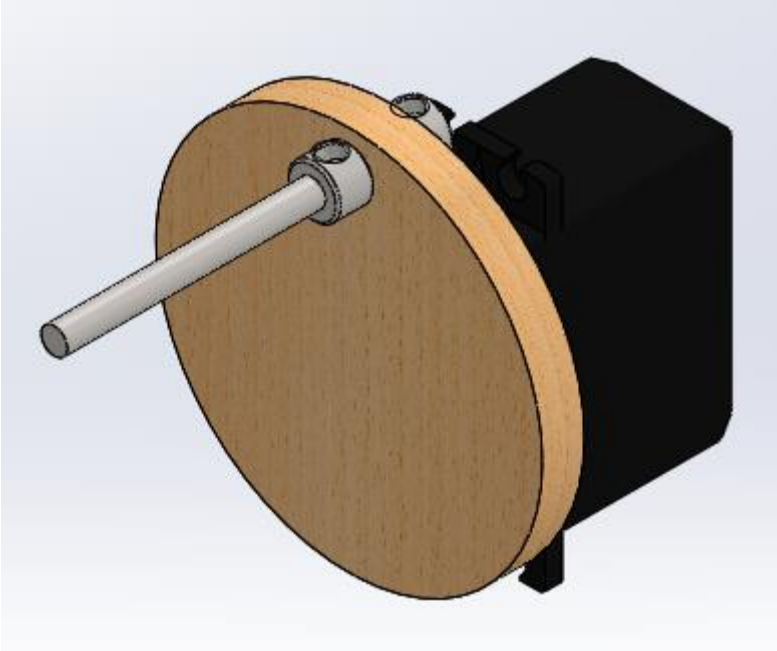
The original 3 punch device was further discussed and scrapped once the cost was assessed and alternate ideas were then brainstormed. An idea was chosen on February 16th which was a 3d printed cylinder with arms at different heights around the outside so that it could be rotated to a position that allows the arm at the desired height to push a button. This was attached to a servo on the front right corner of the robot so that the position would be 180 for the top button, 135 for the middle button, and less than 90 for the bottom button. While the servo was being attached, the front right corner of the platform was broken off. Once it was reattached using Erector set pieces, it was discovered on February 26th that the platform was too high to have the cylinder hit the bottom button. To compensate, the platform would instead ram the bottom button and the cylinder would deal with the top two buttons. Fortunately, the 3D printed piece had not been made at the time and was adjusted accordingly before fabrication. The 3D piece was printed and attached on March 24th. For emergency purposes a makeshift wooden piece was attached to help

with hitting the buttons but that was removed from the final design. Figure XX below is a picture of the button pressing mechanism



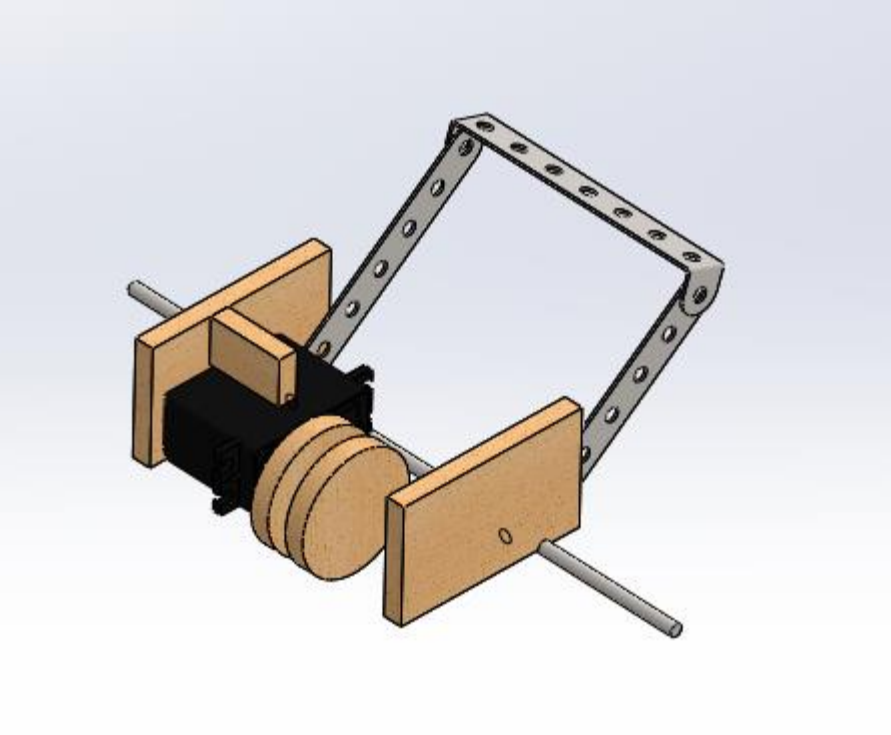
3.2.3. Crank/Switch Mechanism

On March 25th, a hacked servo was attached to the robot, centered on the middle edge of the indentation in the back. It was also placed so that the furthest point was 2.25 inches from the back edge. This hacked servo was going to use a 3d printed dish but scrap wood was chosen because it was easier, cheaper, and stronger. An axle was then placed in the dish to serve as a prong. The axle was cut to a length so that the front end would jut out by 2 inches. The dish was attached using epoxy. On April 2nd, an optosensor was placed on top of the servo to detect when the dish rotated 360 degrees.



3.2.4. Saltbag Mechanism

On March 11th, a hacked servo motor with a spool was placed on the underside of the main platform to act as a winch. Erector set pieces were attached to the front axle to act as arms. At first a wooden scoop was created to pick up the saltbag. It was connected to the winch with string. However testing on March 13th conclusively proved that the saltbag failed to stay on the scoop. To fix the problem, on March 23rd the wooden scoop was removed and was instead replaced with another Erector set piece. The idea was to create a dragging device. Side guards and back guards were placed on the erector set pieces to deal with the turning and backing up. Weights were also placed on top to make sure the dragger kept the saltbag in the vehicle. This new device had a decent rate of success but had a small margin for error. Figure XX below shows the dragger.



3.3. Coding

The coding was done using QTCreator. It was then uploaded onto the Proteus which interacted with the various mechanisms plugged into the machine. The coding was completed after the building of each part was completed. Most of the coding and testing was completed by Edward Hughes and Mark Luke.

3.3.1. Navigation

Due to issues with shaft encoding early on in the project, the first two performance tests were completed using timing to control movement and turning. After some experimentation on February 27th, it was determined that at 50 percent motor power the robot moved 7 in/s and turned 90 degrees in 1.325 seconds. After shaft encoding was fixed, the robot was programmed

to move depending on the number of transitions registered by the encoders. There were several functions to deal with all directions of movement. For forwards or backwards movement each motor was set to an equal percent. For rotation, one motor was set to the negative percent of the other. Due to the gearing of the encoder and axle system, it was determined that there were 60 transitions per rotation of the wheel. As the robot's new navigation system was put into action on March 10th, it was discovered that the encoders were not digital inputs. This resulted in the navigation requiring ten times the number of transitions to function. To fix this the robot's shaft encoders needed to be coded as analog inputs, so a new function was created that counted number of encoder transitions based on voltage ranges. This was inserted into the existing movement code to create functional navigation. The shaft encoders were used in conjunction with RPS to move the robot to specific locations. The RPS code moved the robot in very small increments of encoder counts until the RPS coordinates were in a acceptable range. These RPS values were initially determined by placing the robot on a course in areas that needed the robot to be in a precise location. There were several functions created, each to deal with the correct direction of adjustment. The code for movement using shaft encoders and RPS can be found in Appendix 2.

3.3.2. Button

As the robot approached the buttons, the code started the button pressing function. The function first retrieved the button order from the RPS to find the correct order to press the buttons. Then it rotated the button mechanism to the correct position to push the first button and then moved forward and back again. It waited to register that the correct button was pressed before repeating the process for each button. The function appeared to work correctly but the

correct starting position was difficult to consistently reach during testing. To compensate, if the robot failed to register the buttons being hit, it pushed a solid plank in an attempt to hit all the buttons. That part of the code was removed for final competition. This code can be found in Appendix 2.

3.3.3. Crank

As the robot moved to the crank location, it read the CdS cell value and turned the crank based on the CdS input. The CdS cell was purposely placed on the body of the robot so that if the robot was in the correct position to turn the crank a CdS value could be read. The robot also used an optosensor and a rudimentary pinwheel as shaft encoding to determine when the crank turned 360 degrees. The code for this can be found in Appendix 2.

3.3.4. Saltbag

As the robot approached the saltbag, the robot turned a winch (powered by an hacked servo) to drop the scoop, moved forward, lifted the scoop, and moved backwards. The code for picking up the salt bag failed frequently due to positioning and the unreliable nature of the scoop. Once the scoop was repurposed as the dragger, a new function to pick up the saltbag was created. This function lifted the scoop, moved forward, moved backwards. The success of the function was highly dependent on the positioning of the robot but had a much greater success rate than the scoop.

The second portion of the code that dealt with putting the saltbag into the garage worked as follows. After the robot pushed the snow pile out of the way and approached the garage, The robot lifted the dragger. Then the robot backed up, leaving the saltbag in front of itself. As soon

as it backed far enough, the robot lowered the dragger once more and used it to push the saltbag into the garage. The success of this function depended on whether the snow fell to the left or the right. The robot approached the snow to try to knock the pile to the left but sometimes the snow fell to the right, interfering with the saltbag function.

The functions concerning the salt bag can be found in appendix 2.

3.3.5. Switch

The code for the oil switch involved setting the crank turning device at a position either left or right of the switch, and then turning the device clockwise or counterclockwise. The way the device turned the switch was determined by the RPS. This task relied heavily on the robot being in the correct position because the robot had a small margin for error when approaching the switch. Due to lack of testing and imperfect navigation, the robot was only able to get to and perform the switch task successfully once during testing.

3.4. Testing

During the construction of the robot the vehicle faced several deadlines. In order to meet these deadlines a large amount of testing was done. This section outlines the performance of the robot at each deadline and the observations noted during testing.

3.4.1. Performance Tests

Performance Test 1 occurred February 16th. The requirement was to have a full mock-up or full model in Solidworks. The stretch goal was to have both. At the time, a full cardboard mock-up was built, but the Solidworks was not.

Performance Test 2 occurred February 27th. The requirement was to have navigate to the top of the ramp. The stretch goal was to touch the crank. At the time, the shaft encoding was not yet completed which required the robot to navigate exclusively using time. The code had the robot run along the ramp wall until the robot hit the crank. The robot successfully completed both goal and stretch goal. The Solidworks of the robot at the time can be found in Appendix.

Performance Test 3 occurred March 6th. The requirement was to have navigate to the buttons and push each of the buttons. The stretch goal was to push the buttons in order. The shaft encoding was still not completed at the time, so the robot once again attempted to navigate with time. Using a makeshift ramming device, the robot managed to hit 2 of the three buttons.

Performance Test 4 occurred March 13th. The requirement was to pick up the salt bag and deposit it completely into the garage. The stretch goal was to touch the pump switch. At the time, the robot had completed shaft encoding but also had the saltbag scoop that failed to work. For the performance test, the robot failed to do anything with the salt bag or hit the oil switch.

Performance Test 5 occurred March 27th. The requirement was to have navigate to the crank and turn it the right direction. The stretch goal was to flip the pump switch. The robot had working shaft encoding and a working crank turner at the time. The robot reached the crank turner and correctly turned the crank. It failed to reach the oil pump. The Solidworks of the robot at the time can be found in the Appendix.

3.4.2. Testing Log and Notes

A testing log was created to record the observations noted during the testing of the robot. The log can be found in the Appendix.

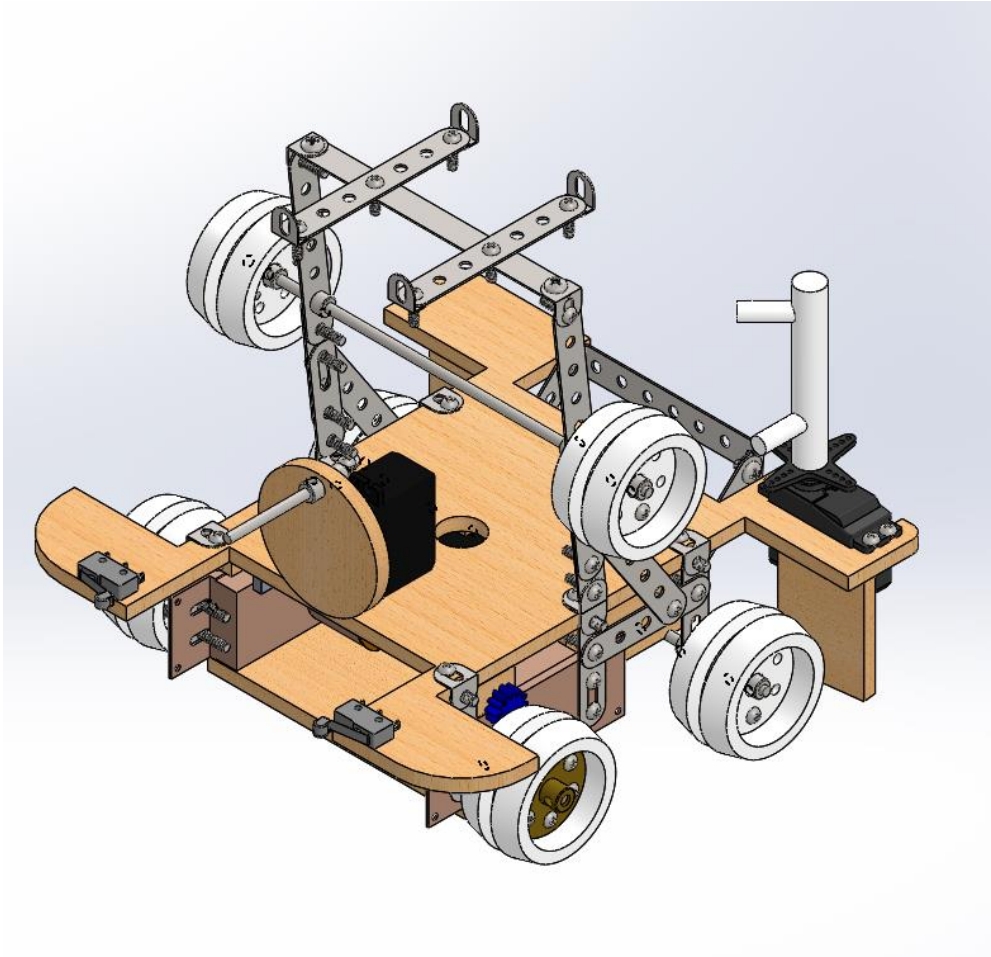
A meeting notebook was created to record the tasks accomplished during the course of the project. A log of meeting notes can be found in the Appendix.

4. Individual Competition

Individual Competition took place April 3, 2015. At the time, all mechanisms besides the saltbag were working. The saltbag was having trouble due fact that it was just changed from the scoop to the dragger. The strategy for the robot was to go the drag the saltbag up to the top side of the ramp, turn the crank, push the snow out of the way, and then attempt to reach the buttons. The oil switch was too difficult to accomplish at the time so it was ignored. The first run resulted in only 15 points due to the fact that the newly installed snowguards interfered with going up the ramp and the new saltbag mechanism. By the second run the snow guards were fixed. In the second run, the robot managed to move the saltbag up the ramp and turn the crank but failed to touch the buttons, resulting in 31 points. For the third run a makeshift button ramming device was attached to try to hit all 3 buttons. The code was also adjusted to better position the robot. These adjustments worked in hitting all three buttons but the salt grabber failed to bring the saltbag up the ramp, resulting in 31 points.

5. Final Design

After the Individual competition, several adjustments were made to the robot. The final design of the robot can be found in Figure 13 on the next page. The final version of the code can be found in Appendix



More extensive diagrams of the final design, including the electrical wiring diagram and the working drawing sets, can be found in Appendix.

5.1. Building

Before final competition, some minor adjustments were made to the robot to improve performance. The front two bumpers rounded heavily to make turning close to walls less of an

ordeal. Often the corners would catch on the walls causing translational movement when only turning was intended. The crank turner was raised on some wooden pieces to better fit into the crank. The original set-up caused some difficulty if the approach of the robot was not head on. The adjustment resulted in less incidents with missing the crank. The tread wheels were recoated in electrical tape coating to increase the friction between the treads and the wheels. The wheels were also realigned to be as parallel as possible. The bottom platform was reattached to keep the motors parallel as well. New treads were also purchased. These were all done to reduce the chances of the treads falling off which happened occasionally but was catastrophic for any run. The main platform was slightly shortened on the front and back to better fit into the 9” by 9” limit.

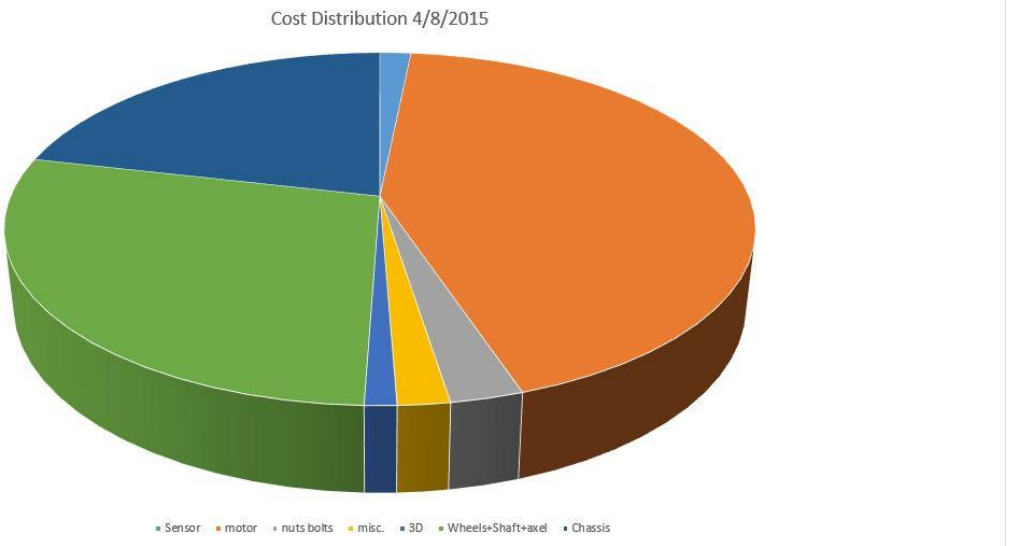
5.2. Coding

After the individual competition, it was noted that the robot performed slightly differently on each course even though it had the same code. To make up for the slight differences of RPS values between each course, the RPS check values were changed to be based off the robots starting location. This changed enabled the robot to perform each task more consistently on the different courses.

Another change happened with the navigation just after the crank task. The robot was previously doing a lot of unnecessary checks to try and navigate itself to the correct position, burning a lot time. Several of the checks were taken out which allowed the robot to get to the correct location quicker than before.

5.3. Budget

The budget available to spend on parts to build the robot was $160. Throughout the purchasing and building process the amount remaining was monitored and spending was adjusted accordingly. By the final competition, $150.78 had been spent on the robot leaving the final balance at $9.22. Them majority of the money was spent on the robot's chassis, drivetrain, and motors as can be seen in the distribution below. Budget money was unnecessarily used by purchasing additional treads, replacing melted gears, and replacing lost set screws. Budget money was conserved by using scraps of wood that remained from the 12"x12" block of wood, from which the 9"x9" chassis was cut out, in other parts of the robot. Scraps were used in creating the rotating winch, the crank turning mechanism, and bumpers on the side of the robot. Dual functionality of the crank turning mechanism in activating the oil derrick was also key in reducing total cost.



5.4. Schedule

The initial schedule was created with the date that the assignment was received as the starting date and the bonus point date requirement as the date for completion. The tasks were scheduled to be completed by the group members following a simple rotation to equally divide work and allow everyone to be involved in all aspects of the project. As the project progressed, it was discovered that some group members were better at certain tasks than others and tasks were assigned accordingly. Table XX in Appendix XX contains the schedule created.

6. Final Competition

Final Competition took place April 11, 2015. The robot used the final design and code as described above. It was intended that the robot would complete all the tasks in approximately 100 seconds.

The first run of the Round Robin started at the light. However, the motor percents used for turning were too low and the robot failed to complete any of the tasks. The motor percents were working in the latest set of test runs and the cause of this change was never determined.

The second run of the Round Robin used an altered code that boosted all motor percents by 10 which resulted in the robot overshooting all navigation. The robot missed all the obstacles by a large margin.

The third run of the Round Robin used a third version of code that scaled back encoder counts to deal with the overshooting issue. The robot started at the light and picked up the

saltbag but after the robot moved up the ramp one of the treads fell off, resulting in a failure to complete the rest of the course. The tread issue is one that has persisted throughout testing and was never solved.

The first run of the head to head competition was not counted due to the RPS system failing for the course. However, during that run the robot spent 60 seconds running at high motor percent into a wall. This resulted in the battery of the proteus being heavily drained.

The second run of the head to head was a rerun of the first. The drained battery caused a great deal of problems with the proper function and navigation of the robot. The robot was very slow but managed to reach the crank where a voltage regulator broke resulting in the robot failing to progress any further.

7. Summary and Conclusions

In summary, the competitions failed to show the strengths of the robot. Much of the time spent working on the robot was used on a navigation system that was never quite as reliable as hoped. The mechanisms and their functions were simple and reliable but the robot often failed to reach the correct position to execute them. The robot was also affected by multiple unsolved and unexplained mechanical issues that often affected the performance of the vehicle. Despite those issues, perfect runs were achieved during testing just before final competition. The reason why the code failed during the final competition was never explained.

However, the design of the robot was not the problem, in fact the design of the robot was its strength. The robot built by the G9 Copperheads is strongly suited for a winter scenario if the robot were to be scaled The robot had enough weight and traction to deal with snow and possessed mechanisms simple enough to work and maintain in very cold conditions. In addition,

the vehicle turned about its center which was exactly where the QR code was housed. The biggest weakness is the speed of the vehicle, but in true arctic conditions slow and consistent would be better than fast but prone to failure.

For future production the main platform would be laser cut and all erector set pieces would be reinforced with Locktite. In addition, a wider cut would be made to accommodate a larger dragging mechanism. Any support pieces made from scrap wood would also be changed into laser cut pieces

References

[1] 2015 Robot Scenario. 2015, February 3. www.carmen.osu.edu

Appendix 1: Pictures of Robot

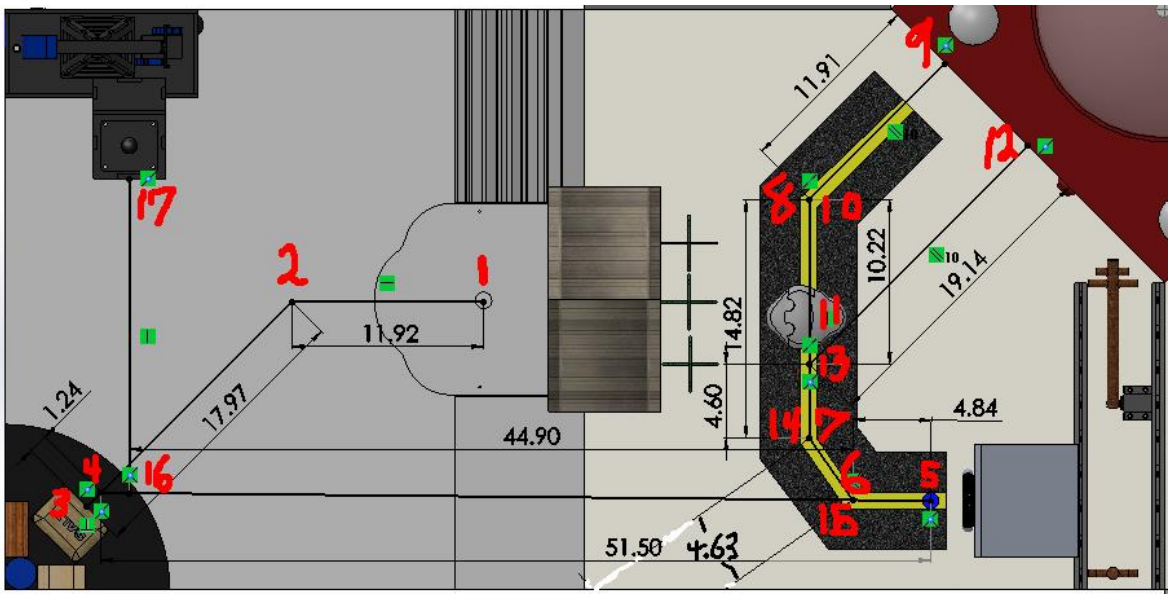
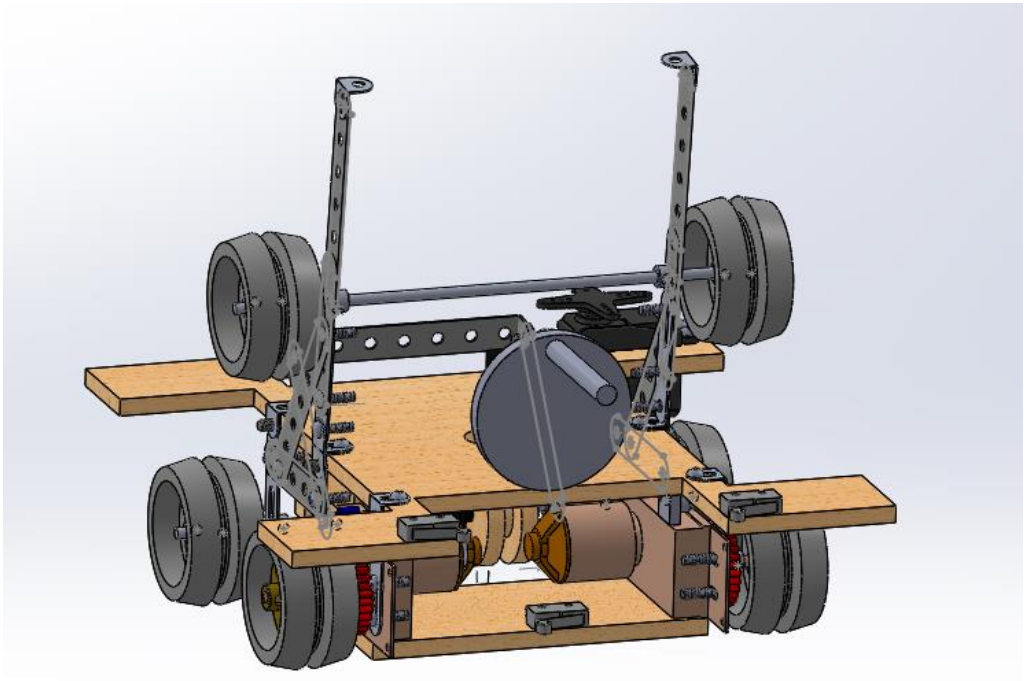
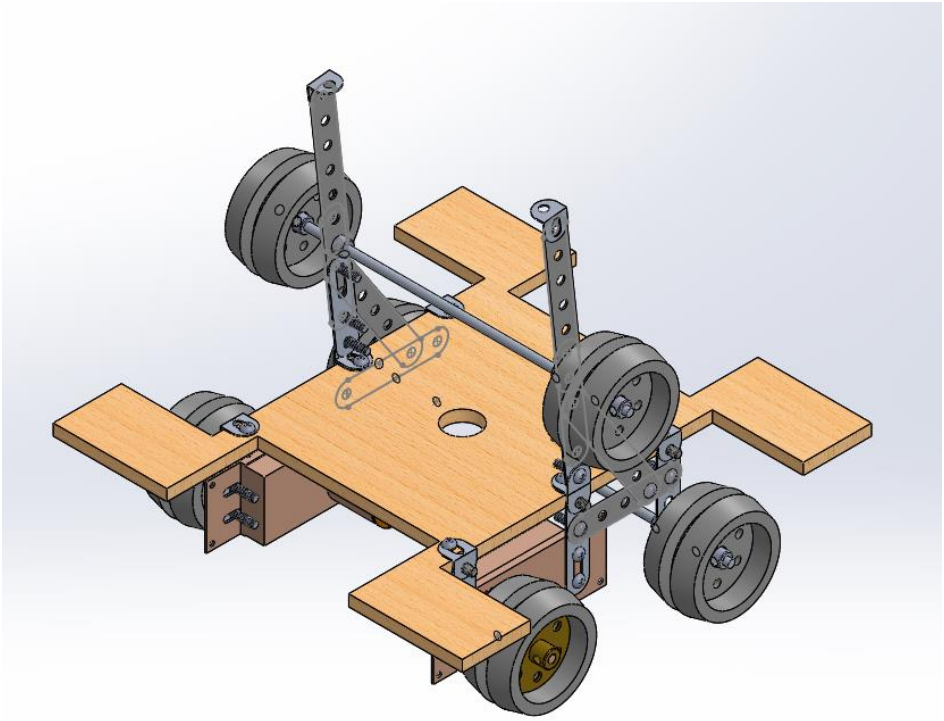


Figure 3 : Picture of the measured distances between obstacles.





Appendix 2: Decision Matrix

Table 1 : Decision matrix giving scores for the preliminary ideas

	Feasible	Effective	Cost	Ease	Total
Route 1	**3**	**3**	**N/A**	**2**	**8**
Route 2	**3**	**2**	**N/A**	**3**	**8**
Route 3	**3**	**2**	**N/A**	**2**	**7**
Route 4	**3**	**3**	**N/A**	**1**	**7**
Route 5	**3**	**1**	**N/A**	**2**	**6**
Route 6	**3**	**2**	**N/A**	**1**	**6**
Route 7	**1**	**3**	**N/A**	**1**	**5**
Chassis 1	**2**	**3**	**2**	**2**	**9**
Chassis 2	**2**	**3**	**3**	**3**	**11**
Chassis 3	**2**	**2**	**1**	**2**	**7**
Chassis 4	**2**	**2**	**1**	**2**	**7**
Chassis 5	**2**	**2**	**1**	**1**	**6**
Chassis 6	**2**	**2**	**3**	**3**	**10**
Chassis 7	**1**	**1**	**1**	**1**	**4**
DT 1	**2**	**3**	**1**	**2**	**8**
DT 2	**2**	**2**	**2**	**2**	**8**
DT 3	**2**	**2**	**2**	**2**	**8**
DT 4	**2**	**3**	**2**	**2**	**9**
DT 5	**1**	**3**	**1**	**1**	**6**
DT 6	**1**	**1**	**3**	**3**	**8**
Button 1	**2**	**3**	**2**	**2**	**9**
Button 2	**2**	**3**	**1**	**3**	**9**
Button 3	**2**	**2**	**3**	**2**	**9**
Button 4	**1**	**2**	**1**	**1**	**5**
Button 5	**1**	**1**	**3**	**3**	**8**
Crank 1	**2**	**2**	**3**	**3**	**10**
Crank 2	**1**	**2**	**3**	**2**	**8**
Crank 3	**2**	**2**	**1**	**1**	**6**
Crank 4	**2**	**2**	**2**	**2**	**8**
Crank 5	**2**	**2**	**3**	**1**	**8**
Crank 6	**2**	**1**	**2**	**2**	**7**
Joystick 1	**2**	**2**	**2**	**3**	**9**

Joystick 2	**2**	**1**	**3**	**1**	**7**
Joystick 3	**2**	**2**	**2**	**1**	**7**
Joystick 4	**2**	**3**	**2**	**3**	**10**
Joystick 5	**3**	**3**	**3**	**2**	**11**
Joystick 6	**1**	**1**	**2**	**1**	**5**
Joystick 7	**2**	**1**	**1**	**1**	**5**
Joystick 8	**2**	**2**	**1**	**1**	**6**
SB 1	**2**	**2**	**1**	**1**	**6**
SB 2	**1**	**1**	**1**	**1**	**4**
SB 3	**2**	**2**	**1**	**1**	**6**
SB 4	**2**	**1**	**3**	**3**	**8**
SB 5	**2**	**3**	**2**	**3**	**10**

Appendix 3: Code

Navigation Functions

```
void counting(int counter)
{
  int r = 0, l = 0, n = 0, m = 0, starttime = TimeNow();
  //While loop to count all of the shaft encoder changes
  while(n < counter)
  {
    if(right_encoder.Value()>1.75&&left_encoder.Value()>1.75&&m==0)
    {
      r++;
      m=1;
    }
    if(right_encoder.Value()>1.75&&left_encoder.Value()<1.75&&m==0)
    {
      l++;
      m=2;
    }
    if(right_encoder.Value()<1.75&&left_encoder.Value()>1.75&&m==0)
    {
      r++;
      m=3;
    }
    if(right_encoder.Value()<1.75&&left_encoder.Value()<1.75&&m==0)
    {
      l++;
      m=4;
    }
    if(right_encoder.Value()>1.75&&left_encoder.Value()<1.75&&m==1)
    {
      l++;
      m=2;
    }
    if(right_encoder.Value()<1.75&&left_encoder.Value()>1.75&&m==1)
    {
      r++;
      m=3;
    }
    if(right_encoder.Value()<1.75&&left_encoder.Value()<1.75&&m==2)
    {
      r++;
      m=4;
    }
```

```
      if(right_encoder.Value()<1.75&&left_encoder.Value()<1.75&&m==3)
      {
         l++;
         m=4;
      }
      if(right_encoder.Value()<1.75&&left_encoder.Value()>1.75&&m==4)
      {
         l++;
         m=3;
      }
      if(right_encoder.Value()>1.75&&left_encoder.Value()<1.75&&m==4)
      {
         r++;
         m=2;
      }
      if(right_encoder.Value()>1.75&&left_encoder.Value()>1.75&&m==3)
      {
         r++;
         m=1;
      }
      if(right_encoder.Value()>1.75&&left_encoder.Value()>1.75&&m==2)
      {
         l++;
         m=1;
      }
      n = (r + l) / 2;
      if (TimeNow() - starttime > 7)
      {
         LCD.WriteLine("TIMED OUT Counter");
         n=1000;
      }
   } //End while
}

void move_forward(int percent, int counts) //using encoders
{
   //Set both motors to desired percent
   right_motor.SetPercent(percent+16);
   left_motor.SetPercent(percent+15);
   //While the average of the left and right encoder are less than counts,
   //keep running motors
   counting(counts);
   //Turn off motors
   right_motor.Stop();
   left_motor.Stop();
}
```

```
void move_backward(int percent, int counts) //using encoders
{
  //Set both motors to desired percent
  right_motor.SetPercent(-percent -16);
  left_motor.SetPercent(-percent -15);
  //While the average of the left and right encoder are less than counts,
  //keep running motors
  counting(counts);
  //Turn off motors
  right_motor.Stop();
  left_motor.Stop();
}

void turn_right(int percent, int counts) //using encoders
{
  //Set both motors to desired percent
  //hint: set right motor backwards, left motor forwards
  right_motor.SetPercent(-1*percent);
  left_motor.SetPercent(percent);
  //While the average of the left and right encoder are less than counts,
  //keep running motors
  counting(counts);
  //Turn off motors
  right_motor.Stop();
  left_motor.Stop();
}

void turn_left(int percent, int counts) //using encoders
{
  //Set both motors to desired percent
  right_motor.SetPercent(percent);
  left_motor.SetPercent(-1*percent);
  //While the average of the left and right encoder are less than counts,
  //keep running motors
   counting(counts);
  //Turn off motors
  right_motor.Stop();
  left_motor.Stop();
}

void check_x_plus(float x_coordinate) //using RPS while robot is in the +x direction
{
  //check whether the robot is within an acceptable range
  while(RPS.X() < x_coordinate - 1 || RPS.X() > x_coordinate + 1)
  {
```

```
    if(RPS.X() > x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(-55, 4);
    }
    else if(RPS.X() < x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(55, 2);
    }
  }
  while(RPS.X() < x_coordinate - .2 || RPS.X() > x_coordinate + .2)
  {
    if(RPS.X() > x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(-55, 1);
    }
    else if(RPS.X() < x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(55, 1);
    }
  }
}

void check_x_minus(float x_coordinate) //using RPS while robot is in the +x direction
{
  //check whether the robot is within an acceptable range
  while(RPS.X() < x_coordinate - 1 || RPS.X() > x_coordinate + 1)
  {
    if(RPS.X() > x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(55, 2);
    }
    else if(RPS.X() < x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(-55, 2);
```

```
    }
  }
  while(RPS.X() < x_coordinate - .2 || RPS.X() > x_coordinate + .2)
  {
    if(RPS.X() > x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(55, 1);
    }
    else if(RPS.X() < x_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.X());
      move_forward(-55, 1);
    }
  }
}

void check_y_plus(float y_coordinate) //using RPS while robot is in the +x direction
{
  //check whether the robot is within an acceptable range
  while(RPS.Y() < y_coordinate - 1 || RPS.Y() > y_coordinate + 1)
  {
    if(RPS.Y() > y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(-55, 2);
    }
    else if(RPS.Y() < y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(55, 2);
    }
  }

  while(RPS.Y() < y_coordinate - .2 || RPS.Y() > y_coordinate + .2)
  {
    if(RPS.Y() > y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(-55, 1);
    }
```

```
    else if(RPS.Y() < y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(55, 1);
    }
  }
}


void check_y_minus(float y_coordinate) //using RPS while robot is in the +x direction
{
  //check whether the robot is within an acceptable range
  while(RPS.Y() < y_coordinate - 1 || RPS.Y() > y_coordinate + 1)
  {
    if(RPS.Y() < y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(-55, 2);
      Sleep(50);
    }
    else if(RPS.Y() > y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(55, 2);
      Sleep(50);
    }
  }
  while(RPS.Y() < y_coordinate - .2 || RPS.Y() > y_coordinate + .2)
  {
    if(RPS.Y() < y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());

      move_forward(-55, 1);
      Sleep(50);
    }
    else if(RPS.Y() > y_coordinate)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Y());
      move_forward(55, 1);
      Sleep(50);
```

```
    }
  }
}

void check_heading(float heading) //using RPS
{
  float heading1 = RPS.Heading();
  //Checks to see if desired heading is above 360 or below 0 and makes it between 360 and 0
  if (heading > 360)
  {
    heading = heading - 360;
  }
  else if(heading < 0)
  {
    heading = heading + 360;
  }
  while(heading1 < heading - 1 || heading1 > heading + 1)
  {
    if(heading1 > heading)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Heading());
      turn_right(70, 1);
      Sleep(0.1);
    }
    else if(heading1 < heading)
    {
      //pulse the motors for a short duration in the correct direction
      LCD.Write(RPS.Heading());
      turn_left(70, 1);
      Sleep(0.1);
    }
    heading1 = RPS.Heading();
  }
}

void check_heading_HARD(float heading) //using RPS
{
  float heading1 = RPS.Heading();
  //Checks to see if desired heading is above 360 or below 0 and makes it between 360 and 0
  if (heading > 360)
  {
    heading = heading - 360;
  }
  else if(heading < 0)
  {
```

```
        heading = heading + 360;
    }
    while(heading1 < heading - 1 || heading1 > heading + 1)
    {
        if(heading1 > heading)
        {
            //pulse the motors for a short duration in the correct direction
            LCD.Write(RPS.Heading());
            turn_right(75, 1);
            Sleep(0.1);
        }
        else if(heading1 < heading)
        {
            //pulse the motors for a short duration in the correct direction
            LCD.Write(RPS.Heading());
            turn_left(75, 1);
            Sleep(0.1);
        }
        heading1 = RPS.Heading();
    }
}

void check_ramp_plus(float start_y, float start_heading)
{
    float y = RPS.Y();
    //Checks to see if the robot is still on the ramp and guns it if it is.
    while (RPS.Y() < 41)
    {
        move_forward(70,3);
//        if (y = RPS.Y())
//        {
//            move_backward(50,40);
//            check_y_plus(start_y - 8);
//            check_heading(start_heading + 180);
//            move_forward(70,100);
//            turn_right(70,10);
//        }
//        y = RPS.Y();
    }
}

void check_ramp_minus(float start_y, float start_heading)
{
    float y = RPS.Y();
    //Checks to see if the robot is still on the ramp and guns it if it is.
    while (RPS.Y() < 41)
```

```
  {
    move_backward(70,3);
//    if (y = RPS.Y())
//    {
//      move_forward(50,40);
//      check_y_minus(start_y - 6);
//      check_heading(start_heading);
//      move_backward(70,100);
//      turn_right(70,10);
//    }
//    y = RPS.Y();
  }
}
```

Button Pushing Function

```
void red_button_press(float start_heading)
{
  int n = 0, red;
  LCD.WriteLine("Trying to press red button");
  //Sets button pusher to desired degree
  check_heading(start_heading + 45);
  servo.SetDegree(48);
  Sleep(500);
  move_forward(40,14);
  Sleep(500);
  move_backward(40,13);
  //Check to see if the button is pressed
  Sleep(500);
  red = RPS.RedButtonPressed();
  //If the button was not pressed, then try 3 more times
  while (red == 0)
  {
    LCD.WriteLine("Red button trying to be pressed again");
    //Check coordinates and adjust if needed
    check_heading(start_heading + 43 + 2 * n);
    move_forward(40,14);
    Sleep(500);
    move_backward(40,13);
    //Check again to see if the button has been pressed
    Sleep(500);
    red = RPS.RedButtonPressed();
    n = n + 1;
    if (n = 3)
```

```
        {
            red = 1;
        } //End if
    } //End while
    LCD.WriteLine("Red button pressed");

} //End red button press
void blue_button_press(float start_heading)
{
    int n = 0, blue;
    LCD.WriteLine("Trying to press blue button");
    check_heading(start_heading + 45);
    servo.SetDegree(138);
    Sleep(500);
    move_forward(40,16);
    Sleep(500);
    move_backward(40,15);
    Sleep(500);
    blue = RPS.BlueButtonPressed();
    while (blue == 0)
    {
        LCD.WriteLine("blue button trying to be pressed again");
        check_heading(start_heading + 43 + 2 * n);
        move_forward(40,16);
        Sleep(500);
        move_backward(40,15);
        Sleep(500);
        blue = RPS.BlueButtonPressed();
        n = n + 1;
        if (n = 3)
        {
            blue = 1;
        } //End if

    } //End while
} //End blue button press

void white_button_press(float start_heading)
{
    int n = 0, white;
    LCD.WriteLine("Trying to press white button");
    check_heading(start_heading + 45);
    servo.SetDegree(3);
    Sleep(500);
    move_forward(40,15);
    Sleep(500);
```

```
  move_backward(40,14);
  Sleep(500);
  white = RPS.WhiteButtonPressed();
  while (white == 0)
  {
     check_heading(start_heading + 43 + 2 * n);
     move_forward(40,15);
     Sleep(500);
     move_backward(40,14);
     Sleep(500);
     white = RPS.WhiteButtonPressed();
     n = n + 1;
     if (n = 3)
     {
        white = 1;
     } //End if
  } //End while
} //End white button press

void button_task(float start_heading)
{
  //Calibrates the servo
  servo.SetMin(500);
  servo.SetMax(2471);
  LCD.WriteLine("In Button Task function");
  //If statements to determine the button pressing order
  if (RPS.RedButtonOrder() == 1)
  {
     LCD.WriteLine("Red 1");
     red_button_press(start_heading);
  } //End if

  else if (RPS.BlueButtonOrder() == 1)
  {
     LCD.WriteLine("Blue 1");
     blue_button_press(start_heading);
  } //End else if
  else if (RPS.WhiteButtonOrder() == 1)
  {
     LCD.WriteLine("White 1");
     white_button_press(start_heading);
  } //End else if
  if (RPS.RedButtonOrder() == 2)
  {
     LCD.WriteLine("Red 2");
     red_button_press(start_heading);
```

```
  } //End else if
  else if (RPS.BlueButtonOrder() == 2)
  {
     LCD.WriteLine("Blue 2");
     blue_button_press(start_heading);
  } //End else if
  else if (RPS.WhiteButtonOrder() == 2)
  {
     LCD.WriteLine("White 2");
     white_button_press(start_heading);
  } //End else if
  if (RPS.RedButtonOrder() == 3)
  {
     LCD.WriteLine("Red 3");
     red_button_press(start_heading);
  } //End else if
  else if (RPS.BlueButtonOrder() == 3)
  {
     LCD.WriteLine("Blue 3");
     blue_button_press(start_heading);
  } //End else if
  else if (RPS.WhiteButtonOrder() == 3)
  {
     LCD.WriteLine("White 3");
     white_button_press(start_heading);
  } //End else if
} //End function button task
```

Crank Turning Function

```
void crank_turn()
{
  int x = 0, y = 0, z=0;
  float start_time=0;
  while(z==0)
  {
     start_time=TimeNow();
     while(TimeNow()-start_time<3 && leftbumpswitch.Value()==1 &&
rightbumpswitch.Value()==1)
     {
          move_backward(30,1);
     }
     if(leftbumpswitch.Value()==0 || rightbumpswitch.Value()==0)
     {
          z=1;
```

```
    }
    if (z==0)
    {
        move_forward(50,20);
        turn_left(70,2);
        move_backward(50,10);
        turn_right(70,5);
    }
}
//While loop to find the CDS cell value
while(x == 0)
{
    LCD.Write(cds.Value());
    //If statement to check what color the light is
    if (cds.Value()<0.1)
    {
        LCD.WriteLine(cds.Value());
        crank.SetPercent(-75);
        while (y < 5)
        {
            if (opto.Value() > 2.6 && y == 0)
            {
                y = 1;
            }
            if (opto.Value() < 2.6 && y == 1)
            {
                y = 2;
            }
            if (opto.Value() > 2.6 && y == 2)
            {
                y = 3;
                crank.SetPercent(-45);
            }
            if (opto.Value() < 2.6 && y == 3)
            {
                y = 4;
            }
            if (opto.Value() > 2.6 && y == 4)
            {
                y = 5;
            }
        Sleep(0.2);
        crank.SetPercent(0);
        x = 1;
        }
    } //End if
```

```
      else if (cds.Value()>0.1)
      {
         LCD.WriteLine(cds.Value());
         crank.SetPercent(75);
         while (y < 5)
         {
            if (opto.Value() > 2.6 && y == 0)
            {
               y = 1;
            }
            if (opto.Value() < 2.6 && y == 1)
            {
               y = 2;
            }
            if (opto.Value() > 2.6 && y == 2)
            {
               y = 3;
               crank.SetPercent(45);
            }
            if (opto.Value() < 2.6 && y == 3)
            {
               y = 4;
            }
            if (opto.Value() > 2.6 && y == 4)
            {
               y = 5;
            }
         } //End while
         crank.SetPercent(0);
         x = 1;
      } //End if
   } //End while
}
```

Saltbag Pick-up and Put-down Functions

```
void salt_up()
{
   int x = 0, y = 0;
   float start_time = TimeNow();
   //While loop to determine if the robot is on the black or not
   while ( x == 0)
   {
      move_forward(40,2);
      LCD.WriteLine("Not on black");
      //If the cds value reads it is on black, then move forward a certain number of counts and
attempt to grab the bag
```

```
      if (cds.Value() > 2.3)
      {
         x = 1;
         move_forward(40,5);
         scoop.SetPercent(50);
         Sleep(1.0);
         scoop.SetPercent(0);
         turn_right(70,5);
         turn_left(70,10);
         turn_right(70,4);
         move_backward(35,30);
         scoop.SetPercent(-60);
         Sleep(1.2);
         scoop.SetPercent(0);
         move_forward(35,25);
         scoop.SetPercent(50);
         Sleep(1.0);
         scoop.SetPercent(0);
         move_backward(35,55);
      }
      //If the time is more than 5 seconds, then attempt to grab the saltbag anyway
      if (TimeNow() - start_time > 5 && x == 0)
      {
         LCD.WriteLine("NOT ON BLACK");
         x = 1;
         scoop.SetPercent(50);
         Sleep(1.0);
         scoop.SetPercent(0);
         turn_right(70,5);
         turn_left(70,10);
         turn_right(70,4);
         move_backward(35,30);
         scoop.SetPercent(-60);
         Sleep(1.2);
         scoop.SetPercent(0);
         move_forward(35,25);
         scoop.SetPercent(50);
         Sleep(1.0);
         scoop.SetPercent(0);
         move_backward(35,55);
      } //End if
   } //End while
} //X at salt bag = 28.1, y = 8.4, heading = +45 from start heading

void salt_down(float start_heading)
{
```

```
  //Sets the saltbag down and attempts to push it into the garage
  scoop.SetPercent(-80);
  Sleep(1.2);
  scoop.SetPercent(0);
  move_backward(40,50);
  scoop.SetPercent(50);
  Sleep(0.8);
  scoop.SetPercent(0);
  check_heading(start_heading + 45);
  move_forward(40,64);
  check_heading(start_heading + 45);
  move_backward(40,40);
  scoop.SetPercent(-80);
  Sleep(2.0);
  scoop.SetPercent(-20);
}
```

Switch Function

```
void flip_switch()
{
  float starttime;
  if(RPS.OilDirec()==0)
  {
     crank.SetPercent(-50);
     Sleep(300);
     crank.SetPercent(0);
     starttime = TimeNow();
     while(centerbumpswitch.Value()==1 && TimeNow()-starttime<1)
     {
        move_backward(50,1);
     }
     while(RPS.OilPress()==0)
     {
        crank.SetPercent(-50);
     }
  }
  if(RPS.OilDirec()==1)
  {
     crank.SetPercent(50);
     Sleep(300);
     crank.SetPercent(0);
     starttime=TimeNow();
     while(centerbumpswitch.Value()==1 && TimeNow()-starttime<1)
     {
        move_backward(50,1);
```

```
        }
        while(RPS.OilPress()==0)
        {
            crank.SetPercent(50);
        }
    }
    crank.SetPercent(0);
}
```

Main Function

```
int main(void)
{
    //MAKE SURE IF FACING COURSE, LEFT SIDE IS SLIGHTLY OFF THE WALL WHILE RIGHT SIDE IS ON
    int x = 0, y = 0;
    float starttime;
    servo.SetDegree(138);
    ButtonBoard buttons( FEHIO::Bank3 );
    //Set the LCD screen color
    LCD.Clear( FEHLCD::Scarlet );
    LCD.SetBackgroundColor(FEHLCD::Scarlet);
    LCD.SetFontColor( FEHLCD::Gray );

    while(!buttons.MiddlePressed())
    {
        LCD.WriteLine(opto.Value());
    }
    Sleep(250);
    LCD.WriteLine("MENU");
    LCD.WriteLine("Left Button: RPS Initialization");
    LCD.WriteLine("Middle Button: Test Light Values");
    LCD.WriteLine("Right Button: Don't Know Yet");
    while(y == 0)
    {
        if (buttons.LeftPressed())
        {
            Sleep(500);
            //Call this function to initialize the RPS to a course
            RPS.InitializeMenu();
            y = 1;
            LCD.Clear();
        }
        else if (buttons.MiddlePressed())
        {
            light_test();
            Sleep(500);
```

```
            LCD.Clear();
            LCD.WriteLine("MENU");
            LCD.WriteLine("Left Button: RPS Initialization");
            LCD.WriteLine("Middle Button: Test Light Values");
            LCD.WriteLine("Right Button: Don't Know Yet");
        }
        else if (buttons.RightPressed())
        {
            LCD.Clear();
            LCD.WriteLine("Hi");
            y = 1;
            LCD.Clear();
        }
    }
    //Press middle button to begin
    LCD.Clear();
    LCD.Write("Press Middle Button to begin");
    while (!buttons.MiddlePressed());
    starttime = TimeNow();
    LCD.WriteLine("JUST BEFORE WHILE LOOP, TIME NOW + START TIME");

    //Wait for middle button to be pressed
    while(x == 0)
    {
        if (cds.Value() < 0.5)
        {
            x = 1;
        }
        if (TimeNow() - starttime > 30)
        {
            LCD.WriteLine("TIMED OUT");
            x = 1;
        }
    }
    //Wait for middle button to be unpressed
    float x_real=0,xadjust=0;//x_real will be established once a qr code is attached to the robot
    float t_real=0,yadjust=0;//y_real will be established once a qr code is attached to the robot
    float motor_percent = 50; //Input power level here
    float counts_per_inch = 14.14; //Input theoretical counts here
    float start_heading=RPS.Heading();
    float start_x = RPS.X(), bottom_x;
    float start_y = RPS.Y(), bottom_y;
//    move_forward(50,90);
//    check_y_minus(start_y - 11.5);
//    turn_left(70,120);
//    check_heading(start_heading + 182);
```

```
  //once started, moves forward until the robot hits the wall in front of it
  while(leftbumpswitch.Value() == 1 || rightbumpswitch.Value() == 1)
  {
     //move_backward(50,2); //DOES NOT DRIVE STRAIGHT, ADD POWER TO LEFT
MOTOR? leftmotor.setpercent...rightmotor.setpercent...
     left_motor.SetPercent(-63);
     right_motor.SetPercent(-60);
  }
  //stops
  left_motor.SetPercent(0);
  right_motor.SetPercent(0);
  Sleep(250);
  //aligns the y-coordinate
  bottom_y = RPS.Y();
  //moves to the point that is 45 degrees from the saltbag
  move_forward(50,85);
  check_y_plus(bottom_y + 11.0);
//   turn_right(70,100);
//   check_heading(start_heading - 90);
//   while(leftbumpswitch.Value() == 1 || rightbumpswitch.Value() == 1)
//   {
//      //move_backward(50,2); //DOES NOT DRIVE STRAIGHT, ADD POWER TO LEFT
MOTOR? leftmotor.setpercent...rightmotor.setpercent...
//      left_motor.SetPercent(-61);
//      right_motor.SetPercent(-60);
//   }
//   left_motor.SetPercent(0);
//   right_motor.SetPercent(0);
//   Sleep(250);
//   bottom_x = RPS.X();
//   move_forward(50,90);
//   check_x_plus(bottom_x + 10.6);
  //faces the saltbag
  turn_right(70,120);
  //if (RPS.X() < start_x + 5)
  //{
     check_heading(start_heading + 230);
  //}
  //moves toward the saltbag
  move_forward(50,90);
  check_x_plus(start_x + 9);
  check_heading(start_heading + 225);
  //grabs the saltbag
  salt_up();
  //faces the wall
  turn_right(70,45);
```

```
Sleep(500);
check_heading_HARD(start_heading + 162);
//runs into wall
move_backward(70, 300);
//makes sure it runs up the ramp
check_ramp_minus(start_y, start_heading);
//turns toward the wall again in case the seam in the wall made the robot turn left
turn_right(70,30);
move_backward(60,35);
//stuff to get to the crank
check_y_minus(start_y + 21.5);
turn_left(70,30);
check_heading_HARD(start_heading + 190);
move_backward(50,35);
check_y_minus(start_y + 24.7);
check_heading(start_heading + 184);
move_backward(50,5);
check_y_minus(start_y + 25.3);
check_heading(start_heading + 187);
move_backward(50,10);
//check if the robot is touching the crank
check_crank(start_y);
//turn the crank
crank_turn();

//this is to deal with the snow and to place the saltbag into the garage
move_forward(50,10);
turn_right(70,20);
move_forward(50,28);
check_y_minus(start_y + 24.5);
turn_right(70,55);
check_heading(start_heading + 95);
move_forward(50,150); //Pushes snow
check_x_minus(start_x - 8.0);
move_backward(50,50);
check_x_minus(start_x - 6.0);
turn_right(70,40);
check_heading(start_heading + 45);
move_forward(50,38);
check_x_minus(start_x - 8.8);
salt_down(start_heading);

check_heading(start_heading + 45);
move_backward(40,10);
check_y_plus(start_y + 23.5);
turn_left(70,45);
```

```
  check_heading(start_heading + 93);
  move_backward(35,80);

  check_x_minus(start_x + 2.2);
  turn_right(70,40);
  check_heading(start_heading + 45);
  move_forward(45,30);
  check_heading(start_heading + 45);
  move_forward(45,20);
  check_x_minus(start_x - 4.2);
  check_heading(start_heading + 45);
//   check_x_minus(start_x - 6.4);
//   turn_left(70,5);
//   check_heading(start_heading + 105);
//   move_backward(50,20);
//   check_x_minus(start_x - 4.4);
//   check_heading(start_heading + 45);
//   move_backward(50,10);
//   check_x_minus(start_x - 7.5);
//   turn_right(70,60);
//   check_heading(start_heading - 45);
//   move_forward(50,60);
//   check_y_plus(start_y + 26.5);
//   turn_left(80,60);
//   check_heading(start_heading + 45);
//   move_forward(50,6);
//   check_heading(start_heading + 45);
  button_task(start_heading);
  scoop.SetPercent(0);

  move_backward(60,20);
  check_heading(start_heading + 58);
  move_backward(60,140);
  check_x_minus(start_x + 12.0);
  turn_right(70,40);
  check_heading(start_heading + 15);
  move_backward(60,180);
  check_y_plus(start_y - 11.5);
  turn_right(70,30);
  check_heading(start_heading - 30);
  move_backward(60,90);
  check_y_plus(bottom_y + 4.0);
  turn_right(70,20);
  check_heading(start_heading - 86.5);
  check_x_plus(start_x);
  flip_switch();
```

```
   //turn_left(70,30); //Code for avoiding snow
   //check_heading(start_heading + 272); //Code for avoiding snow
   //move_forward(50,30); //Code for avoiding snow
//    turn_right(70,40);
//    check_heading(start_heading + 168);
//    move_backward(60,200);
//    check_ramp_minus();
//    check_y_minus(start_y + 21);
//    turn_left(70,20);
//    check_heading(start_heading + 190);
//    move_backward(50,50);
//    check_y_minus(start_y + 24);
//    check_heading(start_heading + 180);
//    move_backward(50,10);
//    check_y_minus(start_y + 27);
//    crank_turn();


//    turn_left(70,80);
//    Sleep(500);
//    check_heading(start_heading + 180);
//    move_forward(50,212.1);
//    Sleep(500);
//    check_ramp();
//    check_y_plus(49.0);
//    turn_left(70,80);
//    Sleep(500);
//    check_heading(start_heading + 270);
//    move_forward(50,180);
//    Sleep(500);
//    move_backward(50,150);
//    check_x_minus(29.3);
//    turn_right(70,40);
//    Sleep(500);
//    check_heading(start_heading + 250);
//    move_forward(50,142);
//    Sleep(500);
//    check_x_minus(22.5);
//    turn_right(70,30);
//    Sleep(500);
//    check_heading(start_heading + 225);
//    move_forward(50,84);
//    Sleep(500);
//    check_x_minus(14.0);
//    button_task();
```

```
  servo.SetDegree(138);
}
```

Appendix 4: Meeting Notes and Testing Log

Meeting Notes

April 10(3:00-5:05)

- Performed competition style runs
- Sped up robot motion

April 9(2:00-9:30)

- Improved navigation to better handle salt bag and buttons

April 8(3:00-7:30)

- New gear attached
- Budget updated
- Worked on navigation

April 7 (6:00-8:00)

- Stabalized parts of robot using epoxy
    - Plate underneath robot to keep wheels in place
- Improved robot traction by removing paint on tape and repainting
- DUE:

- STARTED:
- CLASS:

April 6 (3:00-5:05)

- Improved robot navigation
- Trimmed snow guards
- Presentation outline in progress
- DUE:
- STARTED:
- CLASS:

April 3(3:00-5:05)

- Individual Competition
- 15, 31, 31
- DUE:
- STARTED:
- CLASS:

April 2(2:00-10:00)

- Organized the wires
- Added snow guards
    - First sideways, then vertical, then sideways again
    - Having guards that are too long causes problems with getting up the ramp
- Tested and edited code to the the following
    - Reliably get the saltbag
    - Get up the ramp along the wall
    - turn the crank the correct amount
    - push the snow into a corner
    - put the saltbag into the garage
        - later decided to leave it with the snow

    - Attempt to hit the buttons

- Add a temporary ramming device for the buttons to try to hit all the buttons when desperate
- Bought new treads
- Reglued the bottom platform
- DUE:
- STARTED:
- CLASS:

April 1(3:00-7:30)

- Marked out the places needed to cut to replace the microswitches
- Got the salt dragger to work
- Re aligned the wheels
- DUE: R11
- STARTED:
- CLASS:

March 31(6:00-8:30)

-
- DUE:
- STARTED:
- CLASS:

March 30(3:00-5:05)

- Made the robot fit the box again
    - Removed the microswitches and reshaped the saltbag dragger
- Added 4 pennies and spare nuts, bolts, and washers as weight to deal hold down the dragger
- Added side and back guards to keep the saltbag from moving
- DUE:
- STARTED:
- CLASS:

March 27 (3:00-5:05)

- Completed performance test 5 successfully
- Filed space for the salt bag scoop
- Adjusted budget
- DUE:
- STARTED:
- CLASS: PERFORMANCE TEST 5, NOTEBOOK CHECK

March 26 (1:00-7:30)

- Filed down two screws in the way. Headed by George
    - didn't previously realize that these were in the way
- Tested preliminary crank turning code. Headed by Mark and Edward
    - Hacked servo motor failed randomly
    - Discovered faulty wiring with the voltage regulator
- Readjusted the height and power of the crank turner
    - needed more power to deal with the torque and wasn't quite centered so it was raised onto a couple of wood scraps
- DUE:
- STARTED:
- CLASS:

March 25 (3:00-5:05)

- Built crank turner and attached it. Headed by Mark and George
    - made sure to be parallel with the ground but not the chassis
- Filed down two screws in the way. Headed by George
    - lets the robot approach the oil pump with far more margin for error
- Fixed budget issues. Headed by Nader
- Worked on preliminary crank turning code. Headed by Edward
- DUE: R10
- STARTED: R14 R12

- CLASS:

March 23 (3:00-5:05)

- Controller and motor/sensor placement diagram completed. Headed by Nader
- Final Report continued. Headed by George
- Soldering and attaching of microswitches. Headed by Mark and George
    - Useful for RPS adjustment and crank turning
- Updating budget and notes. Headed by George
- DUE:
- STARTED:
- CLASS:

March 13 (3:00-5:05)

- Create another hacked servo. Headed by George
- Attach prongs on the front of the scoop to help pick up the bag. Headed by Mark
- Performance test failure, the scoop is too thick to pick up the bag
- Positioning of robot is occasionally too low in the y value
- Controller and motor/sensor placement diagram continued. Headed by Nader
- DUE:
- STARTED:
- CLASS: PERFORMANCE TEST 4, NOTEBOOK CHECK

March 12 (2:00-6:00)

- Added another set of shaft lock screws, helped with axle sliding. Headed by George
- Scoop falls off occasionally.
    - erector set pieces don't take hot glue very well
- Reattached at an angle to best get under the saltbag. Headed by Mark
- Adjusted code to turn the servo motor the correct distance to properly lift the scoop
- Testing. Headed by Mark and Edward
- DUE:
- STARTED:
- CLASS:

March 11 (3:00-5:05)

- Scoop does not pick up bag
    - Fails to get underneath, only seems to push it
- shortened string on pulley for scoop
    - was getting outside the spool and getting stuck
- Find out why wheels are sliding on treads
    - probably due to stretching
    - temporarily fixed with electrical tape stuff
- Filed down the areas interfering with the scoop knots. Headed by George
- Controller and motor/sensor placement diagram started. Headed by Nader
- DUE:
- STARTED:
- CLASS: LAB QUIZ

March 10 (2:00-6:00)

- Bought items for scoop mechanism
    - erector set pieces and a servo motor (will be hacked)
- encoding count seems to be far larger than it needs to be. However, it is consistent so it is easy to work with
- Filed down the erector set pieces in the way of the scoop. Headed by George
- Discovered that the rotary shaft encoders are not digital and require the manual coding an analog input requires.
- DUE:
- STARTED:
- CLASS:

March 9 (3:00-5:05)

- Reorganized website. Headed by Edward
    - Revamped sections and homepage.
    - Removed empty pages

- Sorted the purchases for further budgeting. Headed by Nader
    - Created an excel sheet for graphing purposes
- Worked on building the scoop. Headed by George
    - using the front axle as a an axle for the scoop
- Worked on repairing the right bumper permanently. Headed by Mark
    - used erector sets to keep the right bumper in place
- DUE: R09-2
- STARTED: R10
- CLASS: Technical Inspection 1

March 8 (1:00-4:00)

- Fixed shaft encoding on both sides. Headed by George
    - put in more screws through the bush wheel give more points of contact for the gear
    - hotglued the gear onto the bolt ends.
    - moved the encoders a little to better fit the axle gear
- Broke the right bumper again
    - Considering complete removal
- DUE:
- STARTED:
- CLASS:

March 6 (3:00-5:05)

- Performance test 3
    - Hit two of the three buttons
    - Used a makeshift ramming device
- Shaft encoding failed again.
- Fixed the front right bumper again.
    - Used wood glue this time

- Created temporary code that works with time because of the failed shaft encoding
- Began electrical diagram
- DUE:
- STARTED:
- CLASS:

March 5 (2:00-7:30)

- Reglued the right bumper
- Put a new gear on the left axle using epoxy
- Applied electric tape covering on tread wheels
- Second Solidworks of robot created
- DUE:
- STARTED:
- CLASS:

March 4 (3:00-5:05)

- Put on shaft encoding.
- Tested shaft encoding
- Shaft encoding failed badly (it fell off)
    - There was nothing really keeping the gears forced together
    - The gears also slipped on the axle
- Notebook Check
- Updated Budget heavily
    - virtually copied from the store website
- DUE:
- STARTED: R11
- CLASS:

March 2 (3:00-5:05)

- Turned in optosensor line following lab abstract
- Cracked front right panel
    - i.e. it completely fell off

- Built Proteus cage
    - erector set pieces and hot glue
    - extends from the third axle
- DUE:
- STARTED:
- CLASS: Lab 03

February 27 (3:00-5:05)

- Work on line following lab code.
- Fixed problem with wheels slipping
    - Dr Harper suggested filing the axle to hold on better
- Performed performance test 2
- DUE: R09-1
- STARTED:
- CLASS: Preformance Test 2

February 26 (1:00-7:00)

- Finish chassis
    - Deviated a little from the plan for the third axle which resulted in less erector pieces.
    - This also was then extended to hold the QR code
- Continue working on code for programming test 2
- Discovered problems with wheels slipping on driven axle
- DUE:
- STARTED:
- CLASS:

February 25 (3:00-6:30)

- Continue working on chassis
    - Hot glue helps a great deal with securing the erector set pieces

    - Solidworks pieces do not match the size and caused a great deal of problems
        - luckily it ended up fitting with zero margin for adjustment
- Begin programming for performance test 2
- DUE: R08 LAB2
- STARTED:
- CLASS: Guest Speaker 2

February 23 (3:00-8:00)

- Continue to work on chassis
    - Bought erector pieces to place motors on platform
    - Difficult to decide where exactly to place and to keep secure
- Finish Shaft Encoding and RPS Lab
- Work on Line Following Lab
- Started Final Report Outline. Headed by Edward and George
- DUE: R07 Peer Evaluation
- STARTED:
- CLASS: LAB3

February 20 (3:00-5:05)

- Follow the Line Lab coding began. Headed by Nader
    - Difficulty dealing with curves
- Create and submit first parts order. Headed by Edward and Mark
    - Decided on acroname motors with parts for treads
- Begin constructing chassis. Headed by George.
- Cut the laminated wood for the chassis. Headed by Mark
    - Cuts made in the sides and fronts
- Coding for the Encoding lab continued. Headed by George and Nader
    - Having trouble with the vehicle going the correct distance
    - ***Later discovered that the robot was faulty*** Thanks Owen

- Solidworks for chassis begun. Headed by George
- DUE: LAB1 R06
- STARTED:
- CLASS: Notebook Check Speaker 1

February 18 (3:00-5:05)

- Began pseudocode. Finished later. Headed by George
    - Figuring a way to align the vehicle on the road was difficult
- Sensor Lab finished. Headed by Edward and Mark.
- Shaft Encoding lab coding begun. Headed by Nader and George
    - Trouble progressing, not sure why
- Created team notebook. Headed by Edward
- Took measurements of the course. Headed by Mark and George
    - Placement of buttons, crank, toggle
    - Dimensions of necessary constraints
- DUE: R05
- STARTED: R07 R08 R09
- CLASS: LAB2

February 16 (3:00-5:05)

- Reconsider the 3 prong button pusher for a tower that rotates instead.
    - Far lower cost and even more simple.
    - 3d printed
    - Suggestion from Robby Breetz
- Sensor Lab coding done. Headed by Edward and Mark
- Drivetrain calculations begun. Headed by Edward
- DUE:
- STARTED: R06
- CLASS: Performance Test 1

February 14 (12:30-3:00)

- Finished the remainder of the mockup. Headed by Mark
    - Mockup decided to be made with cardboard.
    - Based on sketches in the decision matrix.
- Created a basic Solidwork mockup. Headed by Nader
    - ***Later scrapped due to the fact it was not an assembly.***

February 13 (3:00-5:00)

- Mock up begun. Headed by Mark
- Coding for Sensor Lab begun. Headed by the group.
- DUE: R03 R04
- STARTED: R05
- CLASS: LAB1

February 11 (3:00-5:00)

- Mark is informally decided as the shop guy.
- Created the schedule. Headed by Edward.
    - Jobs should be rotated with no particular bias
    - Jobs to be finished one day before the due date.
- Decision matrix started. Headed by George.
    - Formally decided on treads, dish with wheel, 3 prong pusher, scoop
- DUE:
- STARTED:
- CLASS: Shop Tours

February 6 (3:00-5:00)

- Everyone tried soldering, no clear decision as to who the electrical guys is.
    - Forgot to put on shrink wrap
- DUE: R02
- STARTED: R03 R04
- CLASS:

February 4 (3:00 - 5:00)

- Favored on treads, dish with wheel, 3 prong pusher, scoop.
    - Treads were chosen because it simplifies navigation
    - Other ideas are also chosen for simplicity
- Finished TWA. Headed by entire group.
- DUE: R01
- STARTED: R02
- CLASS:

Testing Log

February 25 (3:00-5:05)

- Noted the shaft lock collars were not properly gripping the axle
    - Problem worsened with more testing
- The robot overturned a planned 90 degree turn
- Due to a lack of shaft encoding, navigation was based on time
- When robot hits an object, it tends to climb the object
- Otherwise functioning as intended

February 26 (2:00-4:05)

- Fixed shaft lock collar problem
- The turning was still slightly variable and overturned slightly.
- Changed program to rely on following wall
- Successfully hit crank in test runs.

February 27 (3:00-5:05)

- Performance test 2!
- Robot successfully climbed the ramp and touched crank
- Reduced turn values by a few milliseconds to guarantee hitting the wall

March 4 (3:00-5:05)

- Attempted to navigate the vehicle using shaft encoding
- Robot failed to do so due to failure in shaft encoders
    - Shaft encoders not securely fastened to robot
- Continued testing resulted in total failure of shaft encoding
- Wheels also started slipping on the treads

March 6 (3:00-5:05)

- Performance test 3!
- Code based on timing due to failure of shaft encoding
- One of the runs hit two of the buttons
- Electrical Tape covering successfully removed slipping of the treads

March 9 (5:00-6:45)

- Tested the navigation of the robot with newly secured shaft encoders and RPS
- Found the shaft encoders were not counting properly
- Treads were slipping on tires when the robot attempted to turn as well

March 10 (3:00-6:00)

- Attempted to fix the navigation with new counting values for the shaft encoders
- Treads still slipping on tires when turning
- Check x, y, and heading functions are not working
- Fixed the counting error with the shaft encoders by increasing the values greatly
- Fixed the check x and y statements by increasing the values slightly and adjusting the loops inside the function
- Fixed check heading statement by adjusting the count values for the turns and adding a sleep function inside the loop
- Treads having trouble driving over the snow
- Added a guard to the left tread to block the snow
- Snow still sometimes gets in the way of the robot if the snow is pushed by the robot into a wall
- Able to navigate the robot with shaft encoding and rps pretty consistently

- Learned that our shaft encoding is using analog values, not digital values, explaining why our original values for shaft encoding were incorrect and explaining the small inconsistency of the distances moved by our robot currently
- Button pressing task seems to work correctly; the robot needs to be able to get into the right position

March 12 (11:00-4:00)

- Check y minus function broken and fixed
- Tested scoop mechanism
    - Reached location
    - Could not pick up salt bag
- Wheels started slipping again

March 13 (3:00-5:05)

- Vehicle occasionally failed to position in the y correctly.
    - Possibly due to RPS being slightly off
- New shaft lock screws fixed the sliding problem
- Tested scoop mechanism
    - Reached location
    - Could not pick up salt bag
- Tried making the vehicle continue to push forward for a certain time
    - did not help pick up the bag
- Attached prongs
    - did not help pick up the bag

March 26 (3:00-5:05)

- Tested the crank turning mechanism
- Hacked servo stopped working
- Discovered it was a wiring problem and fixed
- Code and mechanism to turn crank in correct direction worked

- Robot cannot reach crank yet

March 27 (3:00-5:05)

- Robot reached crank and turned it
- Inconsistent position in relation to the crank
    - Adjusting code to fix

March 30(3:00-5:05)

- Robot dropped drag mechanism on salt bag but salt bag was lost at ramp
- Tested button code. Hits blue button when it hits red button.

March 31(6:00-8:30)

- Worked on navigation
- Consistently reached salt bag storage area and crank

April 1(3:00-5:30)

- Worked on salt bag dragging mechanism
- Added weight to salt bag dragging mechanism
- Raised top axle to increase tread tension

April 2(2:00-10:00)

- Improved salt bag mechanism
- Made robot capable of taking salt bag up and turning crank
- Couldn't be made to hit buttons

April 6(3:00-6:45)

- Tested robot's performance completing whole course up to button pressing
- Discovered treads were slipping off

April 8(3:00-7:30)

- Tested navigation to make salt bag more consistent

April 9(1:30-9:30)

- Improved movement to reach buttons more accurately
- Added function to flip switch but robot could not reach switch in time

- Treads slipped
- Switched treads around and slipping stopped

April 10(3:00-5:05)

- Attempted to speed up robot
- Robot overshot on checks
- Power on checks and some movements lowered to reduce overshooting but robot stopped moving on checks
- Increased power could not test before time ran out

Appendix 5 : Schedule and Budget

Budget

ORDER 1				
013A	**Axle Rod, 8**	**$1.48**	**2**	**$2.96**
CHAW01	**1/4" Laminated Plywood, 12"x12" sheet**	**$4.30**	**1**	**$4.30**
ELEC10	**Motor wire, 2-conductor 18AWG (per foot)**	**$0.20**	**2**	**$0.40**
MOT04	**Acroname Gear Motor (6v - 12v)**	**$12.00**	**2**	**$24.00**
MOTTERM	**Terminal block for continuous rotation motors**	**$0.05**	**2**	**$0.10**
SCREW01	**10 Pack #8 Screws(.25"), Nuts, Washers**	**$0.20**	**1**	**$0.20**
SCREW05	**4-Pack #8 Screws (1") Nuts, Washers**	**$0.20**	**1**	**$0.20**
SCREW08	**10 Pack, #6 screws (0.5"), nuts, washers**	**$0.20**	**1**	**$0.20**

SCREW09	**6 Pack #6 Screws(1.5"), Nuts, Washers**	**$0.20**	**1**	**$0.20**
SCREW10	**6 Pack, #6 screws (0.75"), nuts, washers**	**$0.20**	**1**	**$0.20**
WHLROV1	**Rover Single Tread**	**$4.90**	**1**	**$4.90**
WHLROV2	**Large 6-Hole Rover 3D printed tread wheel (2 halves)**	**$2.00**	**4**	**$8.00**
Sum : $45.66				
ORDER 2				
12	**Angle Bracket, .5x.5" 1x1 hole**	**$0.31**	**8**	**$2.48**
024B	**Bushwheel, 6 hole (with 1 69S)**	**$2.89**	**2**	**$5.78**
055A	**Slotted Strip 2**	**$0.79**	**6**	**$4.74**
WHLROV2	**Large 6-Hole Rover 3D printed tread wheel (2 halves)**	**$2.00**	**2**	**$4.00**
Sum : $17.00				

ORDER 3				
WHL00	**Shaft Lock Collar w/set screw (fits erector axle) (with 1 SCREW15)**	**$0.35**	**8**	**$2.80**
Sum : $2.80				
ORDER 4				
SCREW08	**10 Pack, #6 screws (0.5"), nuts, washers**	**$0.20**	**1**	**$0.20**
Sum : $0.20				
ORDER 5				
055A	**Slotted Strip 2**	**$0.79**	**2**	**$1.58**
Sum : $1.58				
ORDER 6				
SCREW08	**10 Pack, #6 screws (0.5"), nuts, washers**	**$0.20**	**1**	**$0.20**

Sum : $0.20				
ORDER 7				
069S	**Set Screw, slotted head (for erector parts)**	**$0.25**	**2**	**$0.50**
Sum : $0.20				
ORDER 8				
SENROT	**Rotary mechanical shaft encoder w/10 kOhm resistor (24 clicks per revolution)**	**$1.00**	**2**	**$2.00**
WHL08	**x-small nylon gear (12 tooth)**	**$1.00**	**1**	**$1.00**
WHL10	**medium nylon gear (30 tooth)**	**$1.50**	**1**	**$1.50**
Sum : $4.50				
ORDER 9				
5	**Strip, 2.5" 5 hole**	**$0.60**	**2**	**$1.20**

Sum : $1.20				
ORDER 10				
5	**Strip, 2.5" 5 hole**	**$0.60**	**2**	**$1.20**
Sum : $1.20				
ORDER 11				
SCREW08	**10 Pack, #6 screws (0.5"), nuts, washers**	**$0.20**	**1**	**$0.20**
Sum : $0.20				
ORDER 12				
5	**Strip, 2.5" 5 hole**	**$0.60**	**1**	**$0.60**
Sum : $0.60				
ORDER 13				
3	**Strip, 3.5" 7 hole**	**$0.74**	**2**	**$1.48**
Sum : $1.48				

ORDER 14				
WHL00	**Shaft Lock Collar w/set screw (fits erector axle) (with 1 SCREW15)**	**$0.35**	**4**	**$1.40**
Sum : $1.40				
ORDER 15				
069S	**Set Screw, slotted head (for erector parts)**	**$0.25**	**2**	**$0.50**
Sum : $0.50				
ORDER 16				
WHL10	**medium nylon gear (30 tooth)**	**$1.50**	**1**	**$1.50**
Sum : $1.50				
ORDER 17				
WHL08	**x-small nylon**	**$1.00**	**1**	**$1.00**

	gear (12 tooth)			
Sum : $1.00				
ORDER 18				
MOT06	**Futaba Servo Motor (with Hardware)**	**$10.00**	**1**	**$10.00**
Sum : $10.00				
ORDER 19				
2	**Strip, 5.5" 11 hole**	**$0.89**	**1**	**$0.89**
3	**Strip, 3.5" 7 hole**	**$0.74**	**2**	**$1.48**
12	**Angle Bracket, .5x.5" 1x1 hole**	**$0.31**	**2**	**$0.62**
Sum : $2.99				
ORDER 20				
4	**Strip, 3" 6 hole**	**$0.60**	**2**	**$1.20**
12	**Angle Bracket,**	**$0.31**	**4**	**$1.24**

	.5x.5" 1x1 hole			
Sum : $2.44				
ORDER 21				
SCREW08	**10 Pack, #6 screws (0.5"), nuts, washers**	**$0.20**	**1**	**$0.20**
Sum : $0.20				
ORDER 22				
3D	**3D printed part, sold per gram**	**$0.25**	**7**	**$1.75**
Sum : $1.75				
ORDER 23				
WHL00	**Shaft Lock Collar w/set screw (fits erector axle) (with 1 SCREW15)**	**$0.35**	**2**	**$0.70**
Sum : $0.70				

ORDER 24				
WHL10	**medium nylon gear (30 tooth)**	**$1.50**	**1**	**$1.50**
Sum : $1.50				
ORDER 25				
WHL08	**x-small nylon gear (12 tooth)**	**$1.00**	**1**	**$1.00**
Sum : $1.00				
ORDER 26				
WHL08	**x-small nylon gear (12 tooth)**	**$1.00**	**1**	**$1.00**
Sum : $1.00				
ORDER 27				
EPOXY	**Epoxy, Individual Packet w/stick & cup**	**$0.20**	**1**	**$0.20**
Sum : $0.20				

ORDER 28				
069S	**Set Screw, slotted head (for erector parts)**	**$0.25**	**1**	**$0.25**
Sum : $0.25				
ORDER 29				
006A	**Strip, 1.5" 3 hole**	**$0.40**	**2**	**$0.80**
Sum : $0.80				
ORDER 30				
SCREW10	**6 Pack, #6 screws (0.75"), nuts, washers**	**$0.20**	**1**	**$0.20**
Sum : $0.20				
ORDER 31				
4	**Strip, 3" 6 hole**	**$0.60**	**2**	**$1.20**
MOT14	**High-Torque FITEC Servo Motor**	**$12.00**	**1**	**$12.00**

WHL00	**Shaft Lock Collar w/set screw (fits erector axle) (with 1 SCREW15)**	**$0.35**	**2**	**$0.70**
Sum : $13.90				
ORDER 32				
ELEC10	**Motor wire, 2-conductor 18AWG (per foot)**	**$0.20**	**1**	**$0.20**
Sum : $0.20				
ORDER 33				
MOTTERM	**Terminal block for continuous rotation motors**	**$0.05**	**1**	**$0.05**
Sum $0.05				
ORDER 34				
STRING	**String (per foot)**	**$0.01**	**5**	**$0.05**

Sum $0.05				
ORDER 35				
VOLTREG	**Voltage Regulator (per motor)**	**$3.25**	**1**	**$3.25**
Sum $3.25				
ORDER 36				
ELEC10	**Motor wire, 2-conductor 18AWG (per foot)**	**$0.20**	**1**	**$0.20**
MOT14	**High-Torque FITEC Servo Motor**	**$12.00**	**1**	**$12.00**
MOTTERM	**Terminal block for continuous rotation motors**	**$0.05**	**1**	**$0.05**
VOLTREG5V	**Nominal 5.3 Volt Output (2x390 Ohm) (per motor)**	**$3.25**	**1**	**$3.25**
Sum $15.50				
ORDER 37				

069S	**Set Screw, slotted head (for erector parts)**	**$0.25**	**1**	**$0.25**
Sum $0.25				
ORDER 38				
069S	**Set Screw, slotted head (for erector parts)**	**$0.25**	**1**	**$0.25**
Sum $0.25				
ORDER 39				
SENSWRM	**Microswitch, Roller Blade (medium)**	**$1.20**	**2**	**$2.40**
Sum $2.40				

ORDER 40				
ELEC13	**3/32" dia. shrink tubing (per foot)**	**$0.20**	**1**	**$0.20**
Sum $0.20				
)))))				
ORDER 41				
016A	**Axle Rod, 2.5**	**$0.67**	**1**	**$0.67**
WHL00	**Shaft Lock Collar w/set screw (fits erector axle) (with 1 SCREW15)**	**$0.35**	**2**	**$0.70**
ORDER 42				
ELEC13	**3/32" dia. shrink tubing (per foot)**	**$0.20**	**1**	**$0.20**
ORDER 43				

EPOXY	**Epoxy, Individual Packet w/stick & cup**	**$0.20**	**1**	**$0.20**
ORDER 44				
048C	**Double Angle Strip 4.5"x.5" 1x9x1 hole**	**$1.41**	**1**	**$1.41**
ORDER 45				
VELCRO	**Velcro strips sold by inch length**	**$0.15**	**1**	**$0.15**
ORDER 46				
X045	**PENNIES**	**$0.01**	**4**	**$0.04**
ORDER 47				
WHLROV1	**Rover Single Tread**	**$4.90**	**1**	**$4.90**
ORDER 48				

SCREW16	**Long Set Screw, #6-32x1/4"**	**$0.25**	**1**	**$0.25**
ORDER 49				
EPOXY	**Epoxy, Individual Packet w/stick & cup**	**$0.20**	**1**	**$0.20**
ORDER 50				
ELEC08	**Length of Solder (2 feet)**	**$1.00**	**1**	**$1.00**
ORDER 51				
X056	**CARDBOARD (PER 10 SQ IN)**	**$0.01**	**1**	**$0.01**
ORDER 52				
EPOXY	**Epoxy, Individual Packet w/stick & cup**	**$0.20**	**1**	**$0.20**

Schedule

Appendix 6: Pseudocode and Flowchart

Preliminary Pseudocode

1. Saltbag Pickup
   1.1. If the light turns on, move forward 12 inches. If the light doesn't turn on, wait 10 seconds then proceed forward 12 inches.
   1.2. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   1.3. Turn 45 degrees to the left.
   1.4. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   1.5. Move forward until the front CdS sensor detects that the floor is black or the robot moves 15 inches.
   1.6. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   1.7. Lower the scoop by activating the motor that deals with the winch.
   1.8. Move forward until the two front microswitches are activated or 3 seconds of movement.
   1.9. Check RPS for heading and positioning and adjust accordingly. If RPS is unresponsive proceed after 1 second.
2. Crank Turning
   2.1. Move backward 1.25 inches.
   2.2. Check RPS for positioning and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.3. Turn right 45 degrees.
   2.4. Check RPS for heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.5. Move backward until the floor is detected to be black. Since this is 35 inches, periodically check RPS for heading and positioning and adjust accordingly.
   2.6. Turn left 37.12 degrees.
   2.7. Check RPS for heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.8. Move backward until the floor is detected to be yellow or 2.5 inches.
   2.9. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.10. Move back .5 inches.
   2.11. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.12. Turn 87.5 degrees to the right.
   2.13. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.14. Move backwards until the floor is detected to be black or 2.75 inches.
   2.15. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   2.16. Move forwards .5 inches.

2.17. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
2.18. Turn left 55.84 degrees.
2.19. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
2.20. Move back until the floor is detected to be neither black nor yellow or 4.75 inches.
2.21. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
2.22. Check the floor using CdS sensor to detect red or blue light.
2.23. Move back until the back microswitch is triggered or for 3 seconds.
2.24. Turn the dish with a prong 360 degrees in the correct direction.

3. Saltbag Dropoff
   3.1. Move forward until the floor is detected to be black or 5.5 inches
   3.2. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.3. Move backward .5 inches
   3.4. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.5. Turn right 55.84 degrees.
   3.6. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.7. Move forward until the floor is detected to be black or 5 inches.
   3.8. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.9. Move back .5 inches
   3.10. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.11. Turn right 34.16 degrees.
   3.12. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.13. Move forward until the floor is detected to be black or 15 inches.
   3.14. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.15. Move back .5 inches
   3.16. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.17. Turn right 45 degrees.
   3.18. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.19. Move forward until from microswitches trigger or 12 inches.
   3.20. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
   3.21. Release the Saltbag via compressed spring.
4. Button Pressing
   4.1. Move back until the floor is detected to be black or 12 inches.

4.2. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.3. Turn left 45 degrees.
4.4. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.5. Move back 10 inches.
4.6. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.7. Turn right 45 degrees.
4.8. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.9. Move forward until the front microswitch triggers or 19 inches.
4.10. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.11. Move back .5 inches.
4.12. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.13. Turn the button pressing tower to the correct button height.
4.14. Move forward until the microswitch triggers or .5 inches.
4.15. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
4.16. Repeat 4.11 – 4.15 until all three buttons are pressed.

5. Oil Switch
5.1. Move back until the floor is yellow or 19 inches.
5.2. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.3. Move back .5 inches.
5.4. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.5. Turn left 45 degrees.
5.6. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.7. Move back until the back microswitches are triggered or 14 inches.
5.8. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.9. Move forward 3 inches.
5.10. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.11. Turn right 90 degrees.
5.12. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.13. Move back until the floor is black or 42.18 inches. Since this is a long distance, check RPS periodically
5.14. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.15. Move back .75 inches.

5.16. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.17. Turn right 90 degrees.
5.18. Position the dish with the prong on the correct side of the toggle.
5.19. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.20. Move back until the back microswitches trigger or 19.5 inches
5.21. Check RPS for positioning and heading and adjust accordingly. If RPS is unresponsive proceed after 1 second.
5.22. Turn the dish with a prong to toggle the switch.

Basic Modular Functions

1. MoveForward(distance)
    a. Takes the input and divides it by the number of transitions per inch on the shaft encoding. That is used to tell when the motors should stop. Has a positive motor percent. Returns nothing
2. MoveBackward(distance)
    a. Takes the input and divides it by the number of transitions per inch on the shaft encoding. That is used to tell when the motors should stop. Has a negative motor percent. Returns nothing
3. TurnRight(degrees)
    a. The distance between the two wheels times pi is the circumference of rotation. The degrees divided by 360 times that circumference will give the distance. Take that distance divided by the number of transitions per inch on the shaft encoding. That value is used to tell when the motors should stop. The right motor will have a negative percent and the left motor should have a positive percent. Returns nothing.
4. TurnLeft(degrees)
    a. The distance between the two wheels times pi is the circumference of rotation. The degrees divided by 360 times that circumference will give the distance. Take that distance divided by the number of transitions per inch on the shaft encoding. That value is used to tell when the motors should stop. The right motor will have a positive percent and the left motor should have a negative percent. Returns nothing
5. LowerScoop()
    a. A function that will turn the winch a certain length. Returns nothing
6. RaiseScoop()
    a. A function that will turn the winch in the opposite direction a certain length. Returns nothing.
7. TurnCrank(color)
    a. Depending on the color, the dish will turn a set number of transitions in a chosen direction. Returns Nothing
8. SetDish(color)
    a. Depending on the color, the dish will be turned a set number of transitions to prepare for the oil switch. Returns Nothing.
9. ReleaseSalt()

    a. Activates the spring loaded mechanism that pushes out the salt bag. Returns nothing
10. ToggleSwitch(color)
    a. Depending on the color, the dish will turn a set number of transitions to toggle the crank. Returns nothing.
11. Button(color)
    a. Depending on the color scanned, a certain order of buttons will be established.
    b. The function will include the rotation of a motor controlling the button pressing tower as well as the MoveForward() and MoveBackward() functions. The rotation will set the prong at a certain height and the move functions will press the buttons. Returns nothing.
12. RPS(stepnumber)
    a. A multipart function that will first call the RPS to find the coordinates of the vehicle. Returns nothing.
    b. That coordinate will be compared to a set of coordinates that are related to the value of the input. If within a certain tolerance, the vehicle will continue
    c. If not within tolerance, the vehicle will have to compensate. The compensations will be done in the following order and be looped.
        i. If too far in the Y, the vehicle will back up and start over from the beginning of the RPS (function)
        ii. If too behind in the Y, the vehicle will move up and start over from the beginning of the RPS (function)
        iii. If too far in the X, the vehicle will back up, rotate to an angle to intersect the coordinate point, and move forward the correct distance.
        iv. If too behind in the X, the vehicle will back up, rotate to an angle to intersect the coordinate point, and move forward the correct distance.
        v. If the vehicle has the incorrect heading, the vehicle will rotate on the spot to the correct angle.
13. Scan()
    a. A function that is used to scan the ground for a range of values using the CdS sensor. It will return a color.
14. End()
    a. A function that turns off the all motors, servos, and the proteus.

Preliminary Flowchart

Appendix 7: Robot Drawings

Electrical Diagram