Surfer

Group 1

MAE 162E Project Report

Josue Araujo Hernandez

Elisabeth Wilgenburg

Rodolfo Barranco

Kristen Biermayer

Susan Su

Udit Khandelwal

Vibhav Gaur

June 18, 2017

Abstract **6**

A. List of Figures **8**

B. List of Tables **8**

1 Project Scope **9**
1.1 Problem Statement 9
1.2 Client Identification and Recognition of Need 9
1.2.1 High Level Design Requirements 10
Functionalities 10
Weight and Size 10
Aesthetics/Control 10
1.2.2 Low Level Design Requirements 10
Material 10
Propulsion 11
Motors 11
Buoyancy 12
Battery 12
1.3 Project Goals and Objectives 12

2 Project Planning and Task Definition **13**
2.1 Task Identification 13
2.1.1 Modeling 13
The Chassis 13
Steering System 14
The Propeller System 18
The Completed CAD 20
Molds for Chassis and Wheels 20
2.1.2 Manufacturing 21
Additive Manufacturing 21
Urethane Pour Foam 22
2.1.3 Arduino Programming 23
Propellers 24
Steering 25
Sensors 25
2.1.4 Communication 26
2.1.5 Waterproofing 26
2.1.6 Testing 26
Components 27
Completed Robot: Wall Ascent 27
2.2 Timeline 28
2.2.1 Quarter 1 28
2.2.2 Spring Break 29
2.2.3 Quarter 2 29
2.3 Gantt Chart 30

3 Literature Review **31**
3.1 Propellers 31

3.2 Propeller Motors 31
3.3 Servo Motors 33
3.4 Manufacturing (Foam, Materials, etc.) 33
3.5 Batteries 36
3.6 XBee 38
3.7 Distance Sensors (IR and Ultrasonic) 39
3.8 IMU Sensor 41
3.9 Arduino and Raspberry Pi 42

4 Preliminary Design 43
4.1 Concept Generation and Evaluation 43
4.1.1 First Design 43
Spherical Wheels 43
Chassis 44
Propeller and propeller casing 44
Spherical Wheel Frame 45
Design Overview 45
4.1.2 Second Design 45
Tapered-cylindrical wheels 45
Steering mechanism 46
Chassis 47
4.2 Selected Design 48
4.3 Preliminary Analysis 48
4.3.1 Free body analysis of vehicle on wall 48
4.3.2 Abaqus Analysis 49
4.4 CAD Models 51

5 Detailed System Design 52
5.1 Overview of Detailed System 52
5.2 Detailed Drawings 52
Propeller ring 52
Propeller pillars 52
Propeller housing 53
5.3 Engineering Analysis 57
5.3.1 Steering Mechanism Link Design 57
5.3.2 Buoyant Force Equations 57
5.4 Electronics System Design 60
5.5 Control System Design 61
5.6 Materials 66
5.7 Design Reiterations 68
Wheel Mounting Link 68
Brushless motor housing 68
Chassis 69
Wheels 69

6 Summary and Discussion 70

7 Conclusions and Recommendations 71

References 74

Abstract

This is a report on Group 1's senior design capstone project for MAE 162 D/E. The project was completed over two quarters by a group of seven students. It was inspired by a semi-autonomous robot built by Disney, called VertiGo. Similar to the VertiGo, the robot, Surfer, was built with the goal to be an air-propelled, remote controlled car, that could travel on flat surfaces, climb walls, and float on water thanks to its dual setup of two-axis propellers. A budget of $700 was allotted by the instructors.

With the aforementioned goals in mind, the design process began. Different materials were researched, and polyurethane foam was selected as the primary one for its extreme lightness, strength, and ability to be shaped. After this, different brushless motor and propeller combinations were considered, their power and thrust specs chosen in conjunction with the Surfer's theoretical weight. With these major components in place, work on the electronics, communication, and programming components was started. In parallel, a completely in-house remote controller was developed with the ability to control steering, and propeller pitch & RPM.

As different parts started arriving or getting manufactured and a better sense of their usability was formed, other aspects of the robot went through design iterations. Finally, assembly of the Surfer began in earnest in the final quarter of the time period given. As this stage neared completion, certain issues began to arise. One such issue was that the actual weight of the Surfer was more than the calculations and the added safety factor, rendering water floatation unachievable. Secondly, certain circuitry interferences affected the reliability of the servo motors. Unfortunately, right at the very end, with the safety of the Surfer in mind, the wall mounting aspect was put on indefinite hold. The final product performed as expected on ground, and the code to make it drive on the wall was also finished.

The report goes through the need statement, initial goals, and design requirements. It then explores, in detail, the iterative processes taken to have arrived at the final designs for various parts and sub-assemblies. A section on manufacturing, with a focus on materials, is followed by the work done in programming and controlling the robot for the desired situations it was expected to encounter. Finally, the last two sections talk about what was learnt from this incredible experience that was simultaneously a learning process and a way to test the engineering skills the

group members honed for four years. Furthermore, recommendations are described regarding what could have been done differently to reduce the unexpected challenges and better prepare for unforeseen circumstances.

A. List of Figures

1a. The “X” Shaped Chassis
1b. The “H” Shaped Chassis
2. Trapezoid steering mechanism
3. Initial Direct Steering Design
4. Current Direct Steering Design
5a. A CAD Model of the propeller pillar for the pitch motor
5b. A CAD Model of the propeller pillar for propeller ring
6. A CAD model of a mold for a hollow spherical wheel
7. Two different CAD models for molds for spherical wheels
8. A first attempt at pouring the foam shows there is a learning curve
9. Propeller roll and pitch movement
10. Feedback Loop for Steering System
11. Water Floatation test
12. Wall Ascent testing
13. Gantt Chart for the two quarters
14. Omnidirectional spherical wheel design
15. Tapered cylindrical wheels (interim design)
16. Four-bar linkage steering mechanism
17. First 3-D printed chassis mold
18. Free body analysis of still vehicle on wall
19. Abaqus analysis on chassis
20. Completed CAD - Isometric View
21. Completed CAD Model - Top View
22. Optimization of rotation of the bar by varying y-component of Link 1
23. Theoretical radius of a wheel for a given chassis weight with varying wheel thickness
24. Electronics circuit schematic
25. Final Product - robot and remote
26. Electronics circuit schematic of the remote controller
27. Chassis mold
28. Wheel mold
29a. Old Wheel mounting link
29b. Revised wheel mounting link
30. FEA model of propeller mounting shaft

B. List of Tables

1. Optimization Analysis for vector $R1_y$

1 Project Scope

1.1 Problem Statement

Structurally unsound buildings are dangerous and hazardous for search and rescue personnel to enter. In the case of a dangerous event, perhaps an earthquake, it can be dangerous to send in humans to check for survivors. In this case, it would be far easier and safer to send in a robot that can climb walls and navigate areas that are hard to reach because of it's small size. The robot would be less disruptive to the surroundings than a human because it would be lightweight. The robot would also be able to help locate people in unstable buildings so that search and rescue would only have to enter buildings after knowing the exact whereabouts of the survivors.

Furthermore, many people are adventurous and want to document their escapades. The extreme adventurers, such as rock climbers, have a difficult time showing off their skills from different perspectives. A wall climbing robot would be able to show off their skills from a new angle and present new opportunities of film and photography. Additionally, surfers would also be able to use this new robot as it would use similar techniques to stay afloat on water. This would expand one's creativity and provide new ways to tell stories.

1.2 Client Identification and Recognition of Need

The main client base would be first responders in countries where unsound buildings are commonplace, and outdoor enthusiasts everywhere. However, potential clients could also include photographers and filmmakers with some modifications to the robot, such as it's weight carrying requirement to accommodate cameras. There are many applications for a robot that can travel on multiple terrains, such as walls and water, so there is also a rather broad client base for the product.

1.2.1 High Level Design Requirements

Functionalities

The vehicle will be able to handle several different terrains. It will have to move across horizontal land, up vertical walls, and be capable of floatation. Each of the different scenarios has its own constraints. On land, the robot will have a full set of steering and speed capabilities. On the water, it will maintain a lower speed range so that the robot can navigate safely. During the transition onto the wall, the robot will have full control of its capabilities to reduce human error. Then on the wall, the robot will return the steering and speed capabilities mostly to the human, so long as it does not fall off the wall. It must also be able to communicate with a handheld remote at large distances. It needs to be able to accurately relay information back to the user. Some of these functionalities were not possible to complete at the time of this report which will be detailed more to follow.

Weight and Size

The robot will have to be lightweight as it needs to be able to roll up walls and float on water. It it needs to be portable since it's clientele base includes people who will have to hike or walk with a lot of equipment. The robot will have to be a fairly small size so that it is easily stored in a vehicle and carried around by a person.

Aesthetics/Control

The robot is a cross between a remote controlled car and a quadcopter drone. Thus, the controls are mainly done using xbee controls.

1.2.2 Low Level Design Requirements

Material

The constraints of portability and wall traveling capability require the primary material of the vehicle to be extremely lightweight. Since it also has to traverse across different terrains, and experience moderate impact forces while moving from ground to wall and then back, the material should also be strong. The material should also be buoyant and waterproof to allow floatation on water.

With these conditions in mind, we considered two materials: carbon fiber composite and AeroMarine urethane pour foam. Carbon fiber composite has excellent tensile strength for its weight, and can be made strong by changing its grain direction depending on the requirements. However, the setup to make carbon fiber parts is expensive and time consuming. The alternative, urethane pour foam, is even lighter than carbon fiber but with moderate tensile strength, and is 95% water resistant. It is available as a combination of two liquids, which when combined, react to grow into the closed cell foam. We considered samples of three different densities - 2 lb, 4 lb, and 8 lb per cubic foot - and chose the 4 lb per cubic foot as it struck the best balance between strength and weight.

Propulsion

The vehicle is powered by air propulsion. By controlling the direction of thrust, the vehicle can be propelled forwards, or counter gravitational forces when climbing on a wall. In the latter case, the task is more achievable if there are two propellers, one for forward momentum and the other to generate thrust towards the wall. This necessitates two degrees of freedom for each propeller, specifically, roll and pitch rotations. Hence, each will have two motors - for rotation, roll, and pitch.

The size of the propeller is constrained by the need to make the vehicle portable. The propeller diameter, in turn, limits the width of the chassis, and this acts as the main variable determining the motor sizes, the battery size, and in turn the vehicle size and weight.

The direction of movement of the vehicle is controlled by steering the front wheels.

Motors

Each propeller uses three motors: a brushless DC motor for high RPM propeller rotation, and one servo motor for each roll and pitch. Servomotors were chosen for implementing the roll and pitch of the propellers because it is important to know the orientation of the propellers for accurate thrust vectoring. A servomechanism with inbuilt potentiometer is an ideal choice for that.

Similarly for steering, we chose servo motors to rotate the input shaft in order to turn the wheel. Our design uses a custom mechanism to directly steer the wheels: by converting the rotational servo output to wheel rotation.

Buoyancy

The vehicle has to float on water, so the overall density should be less than that of water. This is another reason why we chose urethane pour foam. Calculations were done to make sure the robot floats above the spoke line on the four wheels for the robot's expected weight (which turns out was lower than the actual weight). The four wheel's diameter was also chosen such that there is enough clearance for the propellers to avoid ground contact at any pitch angle.

Battery

The propeller motors are the main sources of power consumption, but the battery, on the other hand, is the biggest weight factor. Clearly, there is a design loop between these two components: the battery should be big enough to power the propellers for a reasonable duration, while the propellers should be wide enough to generate enough thrust to counter the battery weight. The other power consumers are the microcontroller (Arduino Uno), the steering, roll, and pitch motors, the sensors, and the regulators. To power all the devices mentioned above, we chose a Lithium Polymer (LiPo) battery since it has a high energy density and is relatively compact. LiPo batteries offer 3.7 volts, as the average voltage of a charge, per cell. This is important because the motors for the propellers require a voltage just under a multiple of 3.7. The main problems that will have to be considered regarding the battery, are over charging and any damages, such as punctures, which can result in the battery catching on fire.

1.3 Project Goals and Objectives

The goals of this project are to make a working prototype model capable of the following:

1. Use air propulsion to drive itself.

2. House a control system so vehicle has the ability to climb a wall and travel on it.
3. Be light and buoyant enough to float and move on water like a boat.

The working prototype is meant to be a proof of concept of the possible applications listed in the previous sub-sections.

2 Project Planning and Task Definition

2.1 Task Identification

2.1.1 Modeling

We used both Solidworks and GrabCad for our project. Modeling was the most time consuming part of our project and it is one of the most important aspects of the project. It took week four of winter quarter to week 5 of spring quarter for us to finish our design. Modeling provides clear direction and goals for the group. It also allows us to perform preliminary calculations. The final CAD gave us a list of items that needed to be purchased to build the robot. Due to the nature of the robot, this took several iterations because weight was such an important factor. As each of the iterations were performed the CAD model changed slightly. It also changed as the robot was being built due to any problems we had or any changes we needed to make. GrabCad made this easier because everyone had access to the most current model of the project. Modeling was vital to the success of the robot and helped determine different groups to perform different tasks. The modeling process focused on steering system, propeller system, completed CAD, and molds.

The Chassis

The chassis was the first thing that we began to CAD. Each member designed a few CADs so that we could choose a design amongst those designs. The two main design types were enclosing the propellers within the chassis and having the propellers extend beyond the chassis.

We pursued many of the enclosed designs. A lot of these were designed similar to boats, since one of the design goals was to float on water. However, enclosing the

propellers adds a significant amount of weight. Thus, all of the enclosed designs were removed as weight could be reduced by cutting the designs open around the propellers.

There were two main versions of the open propeller chassis, an “X” shape and a “H” shape as seen in Figures 1a and 1b below. Initially, the “X” shape seemed the better version. It reduced the weight on the sides of the chassis, but the shape was not very stable for large sizes of propellers. This was due to the length of the thin arms. When the propellers increased in size from six inches to nine inches. The “X” shape was no longer able to work for the project.

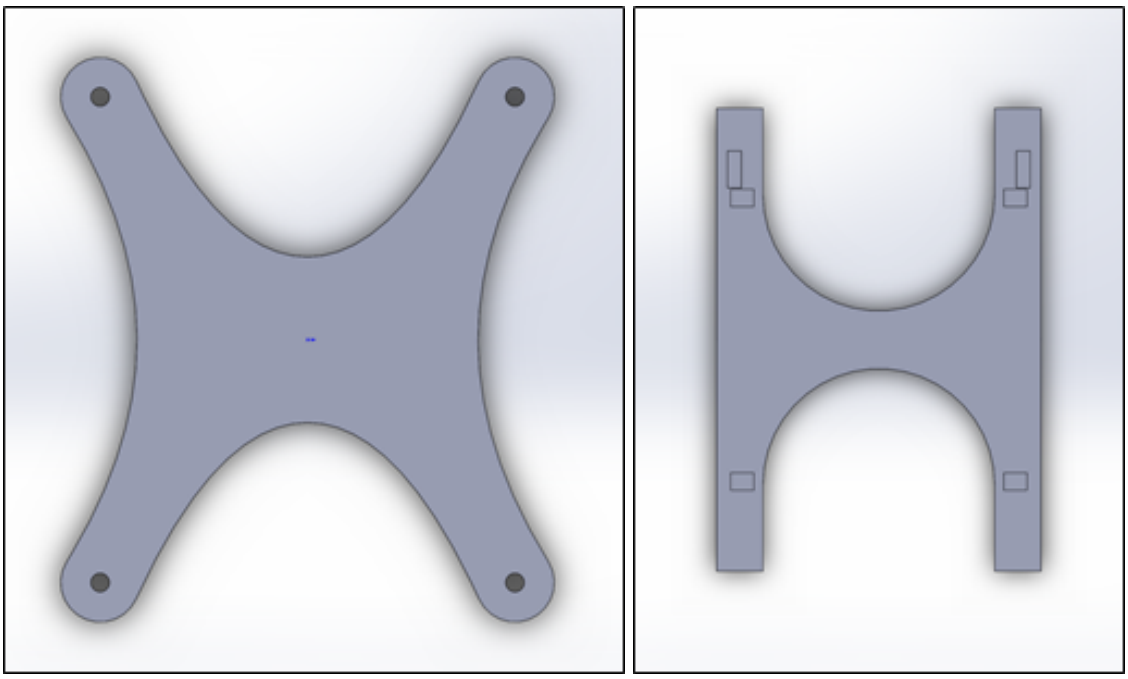


Figure 1a: The “X” Shaped Chassis [left] **Figure 1b: The “H” Shaped Chassis [right]**
Figures are not pictured to scale

Thus the solution was the “H” shape in Figure 1b. This allowed the arms to be slightly thicker in the length. This was also an easier shape to CAD and make molds for. In order to keep the weight as low as possible, the distance between the propeller rings was minimized. It has enough room to hold the battery but that was the only item that would really fit there. The Arduino and the other electronics would be placed to either side of the battery.

Steering System

We considered several options for the steering system. In order to properly evaluate the different steering options, each option was CADed. Each steering system had different requirements such as the number of motors used and the number of links in the system. All of the systems needed to be able to utilize Ackermann steering as that was the ideal steering model for our robot. Ackermann steering is commonly used in motorized vehicles as it prevents the tires from skidding. This is done by having the wheels all turn about a single point rather than using parallel steering.

Several options that we considered utilized only one motor these were the trapezoid steering mechanism, rack and pinion steering, and bell crank mechanisms. Trapezoid steering mechanism was the first steering option we considered. As seen in Figure 2, the model has two long bars and two short bars. The four bars look like a trapezoid. This mechanism requires that one of the long bars is fixed and the other rotates to determine the angle of the wheels. This system has Ackermann steering built into it.

Figure 2: Trapezoid steering mechanism

The trapezoid steering system required that the chassis have adequate space for one of the long bars to rotate. It would need to be placed so that the wheels extended beyond the propellers and such that neither bar would interfere with the propeller system. This would require a lot of extra space in the front of the chassis, which would inevitably add a significant amount of weight to the robot.

Since the goal of the project was to be able to roll up walls, weight reduction was the main consideration for each decision. Therefore the trapezoid steering mechanism was revealed to be the least ideal steering option. The only benefit of this mechanism

was the convenience of Ackermann steering being a built in function of the design. Therefore this system was rejected.

The next option considered was the rack and pinion steering mechanism. This is used in many different vehicles including cars. This steering mechanism uses five main bars including the rack. This moves linearly from side to side which causes the two bars attached to the bar to shift accordingly. Those two bars then move two bars that have a fixed rotational point. These are the steering bars. As they are moved the wheels turn according to Ackerman steering.

This rack and pinion system can be designed around the propeller housing system. This means that it would work better than the trapezoid steering mechanism. However, the rack and pinion need to have contact at all times. This is apparently a problem in small remote control cars where there is often skipping. Skipping is caused when the pinion skips teeth on the rack. In the small scale, dust can also cause problems as it can eventually lead to chipping of the teeth on the rack. Due to the various problems of rack and pinion on the small scale, we looked to another similar steering mechanism, the bell crank.

The bell crank steering mechanism works in a similar way to the rack and pinion. Rather than using linear motion, it uses rotational motion. The bell crank is attached to the motor and as the bell crank is rotated it moves the other bars. The same way the rack and pinion steering does. On the small scale, the bell crank is slightly more reliable than the rack and pinion system; however, since these steering links would extend across the entire width of the chassis. This would add a significant amount of weight to the robot; therefore, this steering mechanism was also discarded.

The final idea was to have individual motors to turn the front to wheels. Initially this design was going to be attached to a y-frame so that the motors would attach to the top of the steering mechanism. Then the motors would only have to turn a single shaft. This would reduce a significant amount of weight from the reduced steering linkages as seen in Figure 3. The one drawback of this option is that we have to program two motors for steering.

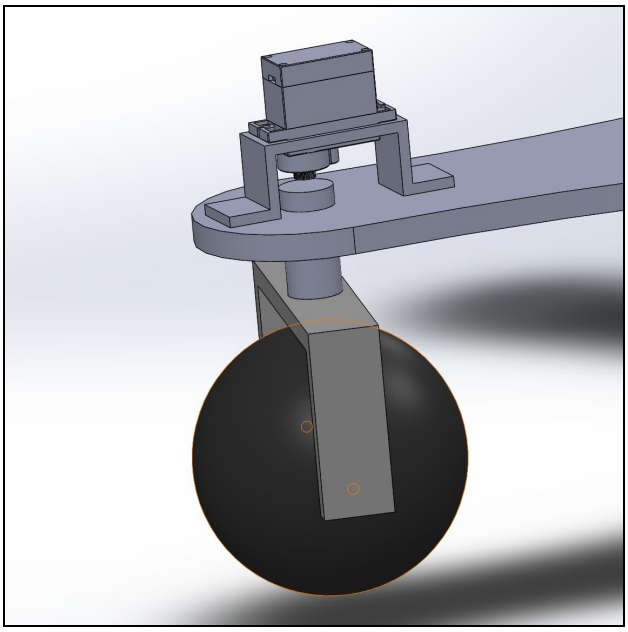


Figure 3: Initial direct steering design

As the full robot design changed over time, the steering changed as well. Near the end of the first quarter, the wheels were moved to the side of the chassis to bring the center of mass of the robot lower to the ground. Being near the end of the quarter, the direct steering was kept in the design. A new steering mechanism had to be designed for the change.

This was done with a four bar mechanism. As is shown in Figure 5, the bar in the front is held fixed and the two other bars are able to rotate. The fourth bar is the servo horn which is the driving bar. As this bar is turned the wheel turns. In order to accommodate the size of the wheel, the fixed bar has to be fairly long so that the wheel is able to have a lot of rotational freedom.

In Figure 4, we could see the necessary components that needed to be manufactured for the steering mechanism and the calculations that needed to be accomplished in order to properly design the four bar. Subsequently, the four bar was modeled in MATLAB to find the ideal lengths of all the steering links. The Matlab code can be found in Appendix 1. The CAD also gave us the ability to know the length and placement of the steering mechanism on the chassis itself.

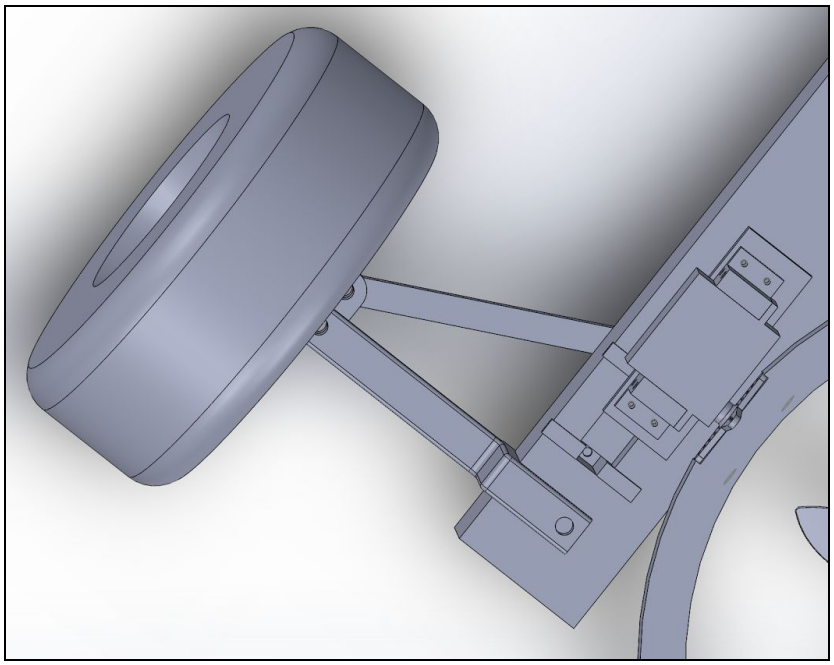


Figure 4: Current Steering Mechanism

The Propeller System

The propeller system is clearly one of the most important parts of the design. The robot uses air propulsion, which is provided by the propellers. Some of the parts of this design were easily designed, such as the propeller ring and propeller pillars. However, the propeller housing required several iterations to achieve the final design.

The propeller ring was clearly necessary from the beginning. The robot propellers were going to move with two degrees of freedom. This meant that we needed a way to turn the roll of the propeller. Thus, it became clear that we needed to have a ring so that we could center the propeller, control the pitch of the propeller, and have control of the roll. The ring was designed to be one inch in diameter greater than the propeller length. This ensured that the ring was protected from the propeller. This also meant that the chassis needed to have space for the propeller ring.

In the first few designs of the robot, we planned on having the propellers flush with the surface of the chassis. This was going to be made possible by the spherical wheels underneath the chassis as shown in Figure 3. However, in the calculations that forced us to switch steering designs, we realized that it benefited the robot if propellers were raised. The closer the center of mass was to the wall the better, and the further out the propellers were the better. In the calculations, it is revealed that when the propellers are raised the thrust counteracts any moments caused by the center of

mass. Due to those important results, it became necessary to design pillars that could handle the weight of the propellers and any of the acting forces from the motors, especially the brushless motor.

The pitch motors and the ring needed to be mounted on the pillars. This meant that there was a platform that was placed on the top of the pillars and that there needed to be two different designs. One pillar was designed around the size of the servo motor that we purchased. The other was designed with the ring in mind. The two final designs of the pillars are shown in Figure 5a and Figure 5b.

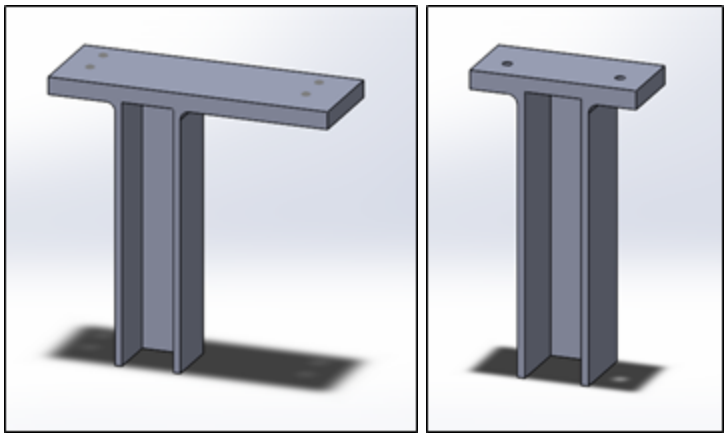


Figure 5a: A CAD Model of the propeller pillar for the pitch motor
Figure 5b: A CAD Model of the propeller pillar for propeller ring

The propeller pillars were also designed with as little mass as possible. This was mainly due to the weight concern of the robot. Therefore, the pillars were made as a shell and designed to be epoxied into place. This meant that there were less screws needed to secure parts which also reduces weight. Despite being hollow, the pillars are strong and can support the weight and forces from the propellers.

As is clear in Figures 5a and 5b, the pillars need to have extra supports for securing the ring and the motors. This led to the design of L-brackets to secure the motor onto one pillar and a propeller ring mount. These parts were screwed and epoxied to the pillars respectively.

The part that went through the most iterations was the propeller housing. Our first design placed the brushless motor and the roll servo in the center of the steering ring. This design involved having a bar that was fixed to the ring, and having the roll servo motor attached to that fixed motor. The propeller housing would be fixed with respect to servo motor. When the servo motor turns, the propeller housing would turn

as well. Unfortunately, the first iteration was quite large. The second iteration cut out most of the excess weight from the first design. This was done by first choosing a slightly smaller brushless motor and small roll motor. Then, reducing the thickness of the propeller housing to the minimum that was safe for the motion of the propellers. However, in trying to keep the housing balanced, it was still quite heavy.

In the third and final design of the housing, the roll motor was moved from next to the brushless motor to the ring of the propeller system. This allowed us to streamline the design of the housing. The propeller housing was designed to carry just the brushless motor and have two bars extending from that housing. One side of the bar is the connected to the roll motor. The other side of the bar is connected to a propeller ring mount.

The Completed CAD

Once the parts were designed, they were assembled into the completed CAD file. This model was been a guideline for the project throughout the project. GrabCad allowed us to upload new files without destroying the old ones, so we could see a clear progression of our project. The Cad modeled version also is a direct representation of our current model. This means that if we were to rebuild the robot all the parts are in one folder.

The completed CAD also let us do stress testing on Solidworks and Abaqus. The completed CAD has the proper dimensions and material properties. GrabCad was also useful in setting the electronics layout as we could see where everything would fit. Ultimately, the electronics were placed to allow the least amount of added weight in the wiring. But the CAD was useful in planning.

Molds for Chassis and Wheels

Modeling was also a vital part of the mold creating process. We chose to use 3D printing to construct our molds, which requires that we have a model to send to the printer. This option was the most economical because we had access to free 3d printers and our own personal printer. 3D printing also allowed us to test several molds because they are easily formed in a short amount of time. This allowed us to pick the best mold for our project.

This robot required many unique parts, mainly the chassis and the wheels. Since both those parts are made of a urethane pour foam, we needed to create negatives of the desired shapes. As there were mainly two designs for the chassis and we chose the "H" frame relatively quickly, there was only one chassis mold design. However, the wheel took several iterations before coming to a conclusion. Therefore, we designed numerous iterations of wheel molds.

In mold design, there are more things to be considered than just the negative shape. For instance, we needed to consider how the molds were going to be held together, which lead us to create interlocks between mold pieces. We considered how the foam expanded and pushed outward against the mold walls. This meant that we left some excess space for the foam to expand and push against without force fitting the mold pieces. We also had to consider how we were going to separate the molds. After several unsuccessful attempts, we concluded that we needed to include handles on the wheel mold. These were several of the concerns that we had when designing the mold. Even with these considerations, it was easy to make mistakes in the design and have to reprint parts.

2.1.2 Manufacturing

Manufacturing is another vital part to the completion of the project. Without the unique parts needed, the robot would not be able to function. There are two main ways to manufacture in this project: Additive Manufacturing and Pour Foam. Although, cutting and machining of plywood and sheet metal was also utilized.

Additive Manufacturing

Additive manufacturing was the main form of manufacturing for this project. It was used for printing parts such as the steering links and the molds to pour the chassis and the wheels. Additive manufacturing also allowed us to iterate on designs and any mistakes could be rectified relatively quickly by printing out a new piece. This allowed for more chances to test parts. The more iterations in a build design, the better the result will be, generally. Additive manufacturing also allowed us to use light materials like ABS. One problem was the limitations on the dimensions we were able print due to the size of the printer. The original chassis mold that was manufactured using additive

manufacturing, had to be printed in four pieces due to the size of the chassis and the limited size of the printing area. Ultimately, the chassis had to be made out of plywood and sheet metal with additive manufacturing used to cover the bottom of the chassis which was a 3D printed plate that had built in placeholders for the pillars and servo motors.

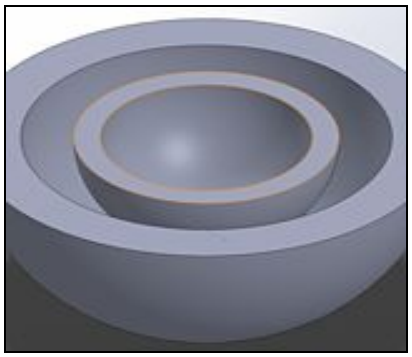


Figure 6: A CAD Model of a mold for a hollow spherical wheel

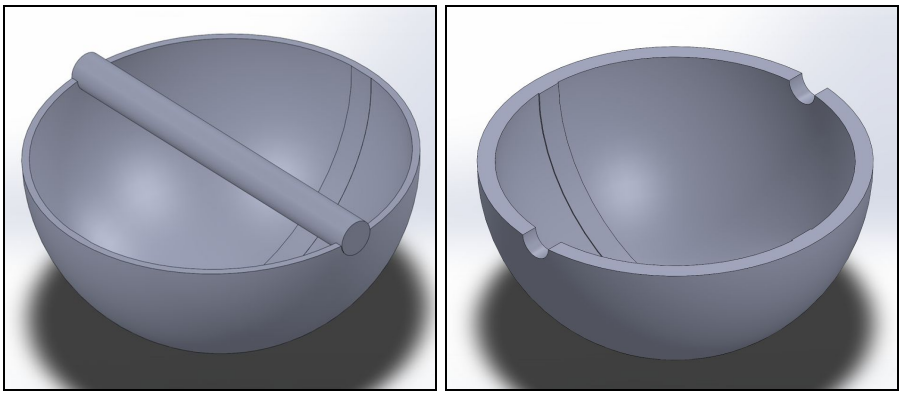


Figure 7: Two different CAD models for molds for spherical wheels

Urethane Pour Foam

The most challenging part of manufacturing was working with the pour foam. This material expands quickly and has a working time of 60 seconds. It hardens within a minute and continues to harden throughout the day. This material is useful because it is extremely light and also is closed cell so that it can float on water without absorbing very much of the water. This pour foam is also very adhesive when in liquid form. To get the ideal shape for foam, the molds have to be precise and allow the foam to expand in way that is easy to remove excess foam.

Figure 8: A first attempt at pouring the foam shows there is a learning curve

The foam can be shaped using sand paper; however, it is ideal to minimize the need for this. Fortunately, the foam pouring process is quick and therefore can be used to produce several renditions within an hour. Pieces of foam manufactured separately could also be easily adhered together using Gorilla Glue as both the foam and glue are urethanes and adhere well together.

2.1.3 Arduino Programming

The robot requires that the components work together without the user specifying what part of the robot needs to be adjusted. This requires sensors and motors working in tandem together. In order to do this, the robot relied on the Arduino Uno. This took in any commands from the controller and then decided what parts needed to change. There were three main groups of programming: the propellers, steering, and the sensors.

The Arduino Uno board was chosen to be the “brain” of the robot. It was tasked with controlling all the motors through user input which was wirelessly communicated by means of a remote controller. Each input on the remote controller was processed by the Arduino and converted to a change in the state of the robot, for instance - rate of propeller rotation, pitch angle of the propellers, steering position of the wheels etc.

Another major task of the Arduino was to read information about the state of the robot from the inventory of sensors such as ultrasonic sensors. Furthermore, the original goal was to also implement an IMU for the robot to make use of while changing its state from horizontal to vertical. The Arduino would have read the IMU’s acceleration and rotation and used this information to control the direction of thrust so

as to facilitate the robot's transition from horizontal to vertical. Unfortunately, due to time constraints, this was not implemented since it was not completely necessary.

Propellers

The propellers had the important task of propulsion and maintaining balance when the robot was on the wall. When the robot was on a relatively flat surface, or anything that is on the ground level, the propellers are solely for propulsion; therefore, the propellers were generating thrust in the forward direction. The propellers were not used for steering. The next scenario considered was when the robot approached a vertical wall, the front propeller needed to generate an upward thrust and the rear propeller needed to generate a forward thrust. This caused the robot to mount the wall, from a programming point of view.

As previously mentioned, when the robot is on the wall, the propellers needed to generate thrust upward and towards the wall. This was to balance the effect of gravity and keep the robot on the wall.

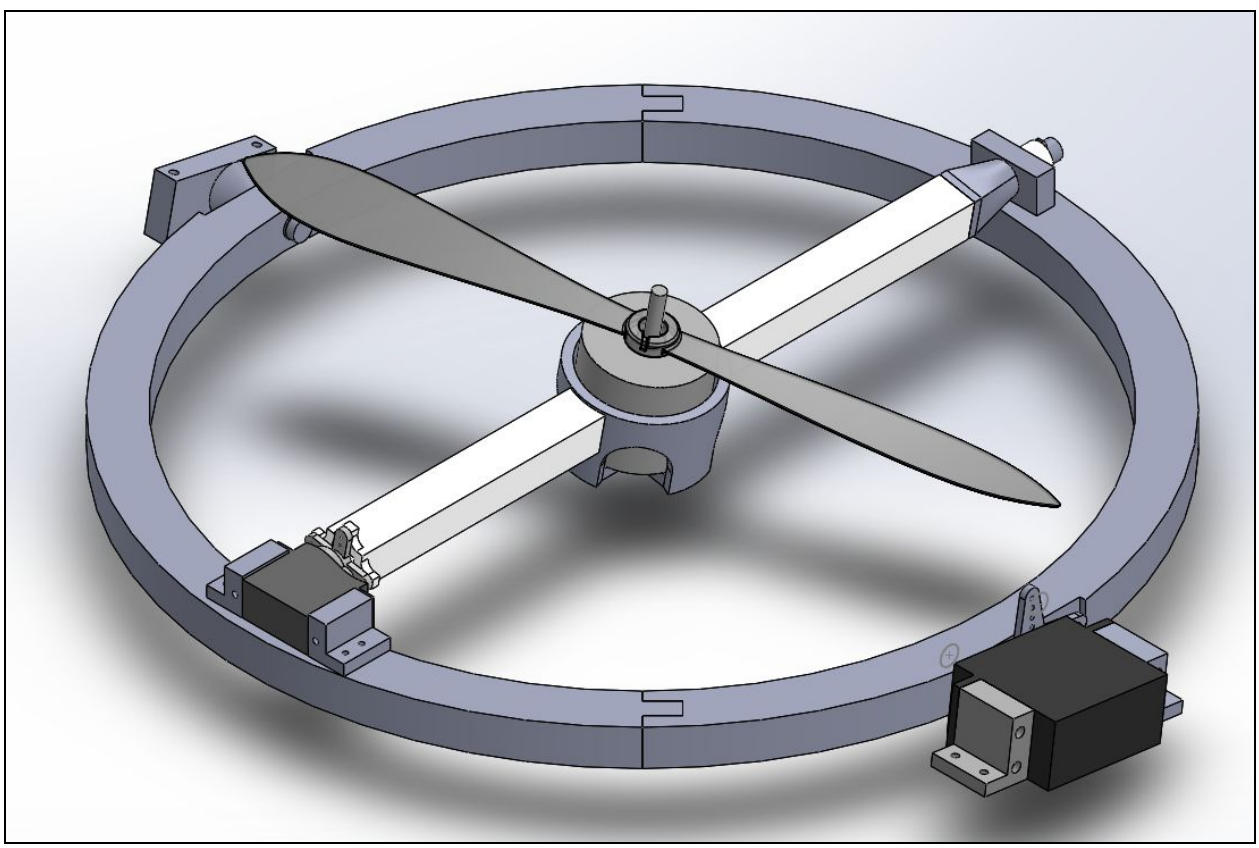


Figure 9: Final propeller ring allowing roll and pitch movement. The ring has a final diameter of 11 in. In order to be printed out, the ring had to be split into two pieces which were then unified using epoxy.

Steering

The steering also worked with the programed Arduino as the two motors had to work together and adjust at the same rate of time. This meant that if the left wheel turned then the right wheel must turn in accordance to parallel steering. The Arduino also had to make sure that when the robot is going straight, the wheels were properly aligned after turning. The following diagram shows a brief overview of how the user input, Arduino, and servo motor worked together to achieve the desired position.

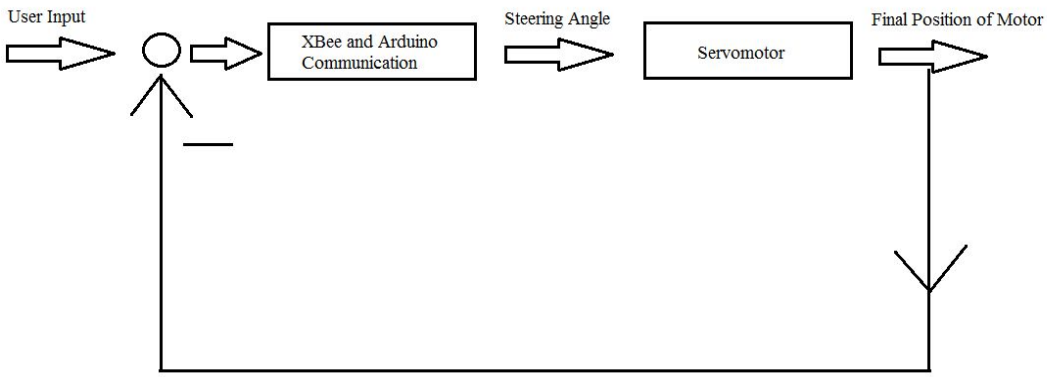


Figure 10 : Feedback loop for steering system

Sensors

There are several sensors that were intended to be used in this robot; however, only the ultrasonic sensors proved to be absolutely necessary. They provide the Arduino with the information necessary to determine when the robot is close enough to the wall that mounting can begin. The ultrasonic sensor on the back of the Arduino would have also been used when the robot was dismounting the wall. The ultrasonic sensor essentially serves one purpose: as a distance measurer. The IMU could have determined which way is downward due to the effect of gravity. This would have provided the Arduino with additional information to determine which direction the propellers needed to be applying thrust to compensate gravity.

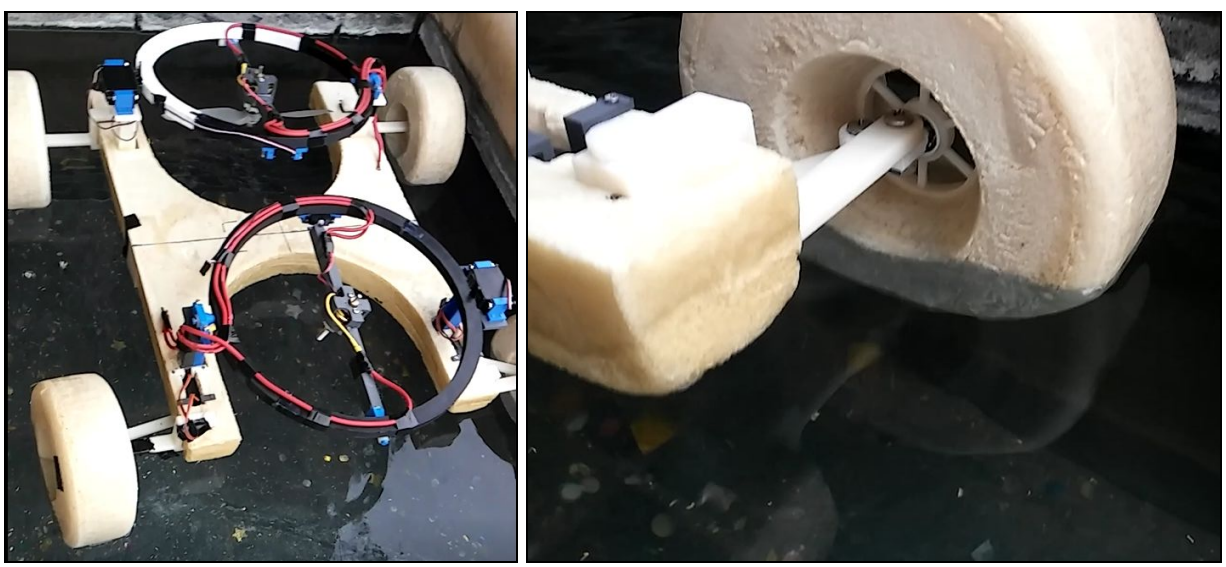
The lipo battery monitor is extremely important to the design of the robot. It will inform the Arduino that the lipo battery is low. Then the Arduino will inform the user that the battery is low. This is important because the user needs time for the robot to dismount from a wall prior to the battery falling. If the battery failed at the top of a building, the robot would not have been able to sustain the damage from the fall.

2.1.4 Communication

The user is able to be use a controller in order to communicate with the robot. This was done using an Xbee controller and an Xbee on the robot. The two Xbee communicated with each other and the Xbee on the robot communicated with the Arduino. This gave the robot a range of about 100 meters. That should be good enough for either someone rockclimbing or for search and rescue. Due to the budget of the project the Xbee controller was built with PVC pipe and joysticks. Getting the two Xbees to work together and with the Arduino took a bit of time.

2.1.5 Waterproofing

The electronics were not waterproofed in order to reduce weight. A preliminary test on water showed that the robot would not be capable of floating in water safely. Figures 11a and 11b below demonstrates this. With a preliminary weight of 1.6kg, the robot reached its maximum water level.



Figures 11a and 11b: Water floatation test (without battery and electronics)

2.1.6 Testing

The final task of the project was testing the completed build and making any adjustments necessary. However, throughout the build testing was done on all the parts before placing them on the robot.

Components

The components that make up the robot must be up to spec or else the robot will not run at full capacity. Thus, the testing each of the components that make up the robot was necessary. Each of the components required a different type of testing specific to the requirements of the component. For instance, the battery needed to be tested for voltage, amplitude, and battery life. This required the use of a multimeter to measure voltage and amplitude. Then the battery was also hooked up to the propellers and motors and left to run until the battery died to test the life of the battery.

The main testing of components was done on the chassis, wheels, and steering mechanism. Each of these parts needed to withstand stress and the weight of the robot. The wheels and steering mechanism needed to undergo a lot of torque. The chassis needed to be tested to make sure that it will not break at any point due to the weight and the thrust. So stress and strain testing was a major portion of testing the components.

Each of the motors needed to be tested for rpm and torque. RPM was tested by attaching a small gear to a large gear and having an encoder read the movement of the large gear. Torque can be measured by measuring how long it takes for the motor to accelerate a bar of a specific moment of inertia. This yields the angular acceleration which is related to the torque.

The sensors were tested for reliability in each of their respective jobs. For instance, the ultrasonic sensor needs to read out distance to the wall. Therefore, placing the sensor at different distances from the wall and checking the readout was used to determine how much error was associated with varying differences.

Completed Robot: Wall Ascent

The completed robot was tested climbing the wall using a rig mechanism similar to the mechanism used in rock climbing. This was done in order to prevent the robot from completely falling to the ground in case the electronics failed and the robot lost thrust. The team had to wear safety glasses in order to prevent injuries in case of propeller failure. During preliminary testing, the nut securing the propeller in place was observed to come off, making testing dangerous. The propeller running at just half speed capability has the ability to make small indentations in wood; for this reason the team handled the propeller with extreme caution.

Figure 12: Wall ascent testing

2.2 Timeline

The timeline for this project is split into three sections: Quarter 1, Spring Break, and Quarter 2.

2.2.1 Quarter 1

The project was selected during the first quarter and the main task was to create a completed CAD. This required doing research into materials, propeller types, batteries, controllers, steering, and electronics. From Week 3 to Week 5, we focused on researching all of the different options necessary and doing initial CAD options for the respective parts. Week 5 was used to select steering and materials. During Week 6, the material was purchased and initial testing commenced. Preliminary calculations were done as well during this week such as thrust and buoyancy calculations. During the last three weeks of the quarter, several molds were made for the wheels, the final design was chosen, and the steering mechanism design was finished.

2.2.2 Spring Break

During spring break not a lot of work was performed on the robot. Parts were ordered and the team waited for their arrival through the first week of the quarter.

2.2.3 Quarter 2

This quarter was focused on building the completed design. The goal was to have a completed prototype by Week 4. However, different parts of the robot arrived at different times preventing a full assembly. The final assembly was completed during week 10, and it still presented some problems as parts broke. The 3D printing presented problems since throughout the quarter there was a high demand for 3D printed parts for other projects in Professor Schaefer's class. Additionally, the wheel molds for the chassis and the wheels went through many iterations until Week 6 when the chassis was finally completed. After the chassis was completed, we used it for initial testing of the completed model. Some parts broke, and needed to be reprinted.

Programing the Arduino and the Xbee Controller took place in Quarter 2 as well. This was the most intense part of Quarter 2 because the success of the robot was dependent upon the success of the coding. During Week 2 and Week 3, the Xbee controller was built and programmed to work with the Arduino. The Xbee Controller was built in Week 3 and programed in Week 4.

2.3 Gantt Chart

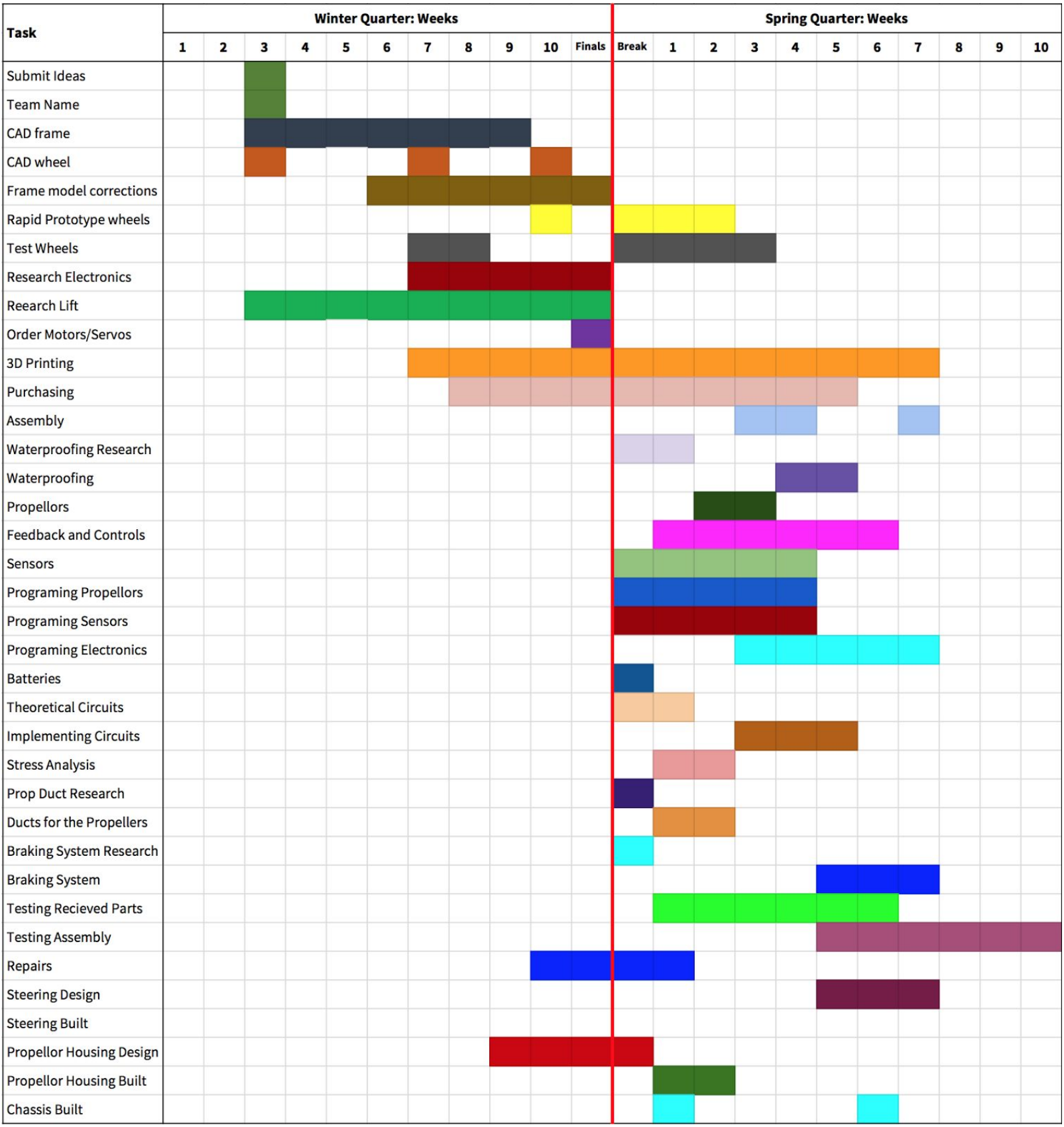


Figure 13: Gantt Chart for the two quarters

3 Literature Review

3.1 Propellers

An important component of our robot is the propellers that will be mounted on the chassis. Controlling the rotation and orientation of the propellers will enable us to control the direction and magnitude of thrust applied by the robot. Through this, the robot will be able to achieve motions such as flipping onto the wall and traveling across a wall. Therefore, the size of the propellers that we choose will play an integral part in the behavior of our robot. Because our application of propellers is relatively unique, we used propeller sizing for RC Planes as a model of a comparison. The following research details the information gathered on propeller sizing.

The performance of an electric motor is dependent on the propeller size. Therefore, a common way of approaching motor and propeller sizing is to refer to the manufacturer's recommendations. Although using these recommendations can be helpful, it is useful to understand the basics of propeller size labeling. Propellers are generally designated with two types of measurements - diameter and pitch. The diameter measures, from tip to tip of the propeller, the arc created when spinning; the pitch measures the distance the propeller will travel through air with respect to a single revolution. The higher the pitch value, the faster the plane can travel. Values given by the pitch should be viewed as a theoretical value as real life effects, such as air density, can significantly affect this value (Carpenter, 2017). The diameter of the propeller influences the amount of thrust it can generate. For example, larger diameter propellers can reduce the motor's RPM as the engine has to do more work to achieve the same effect as a smaller diameter propeller (Carpenter, 2017). Considerations of different combinations of diameter and pitch, along with trial and error, will help narrow down our final dimensions for the propellers.

3.2 Propeller Motors

To understand the advantages of Brushless DC motors, it is important to distinguish the characteristics of Brushless and Brushed DC Motors. A brushed motor is composed of four basic components - brushes, magnets, armature, and the

commutator (Mahoney, 2012). The magnets reside on the outer shell of the motor, commonly known as the motor stator as the magnets remain stationary. Motor stator's are the source of a constant magnetic field in the motor when operating. Unlike the stationary behavior of the magnets, the armature composes the rotating section of the motor and consists of poles, terminals, and the commutator. Commutator rings serve as the connection between the armature and DC power source. The poles are electromagnets; there are often three or more poles on a conventional brushed motor. The terminals are termed for the site where the copper wire of the poll connects. Lastly, the commutators are switches on armatures that contribute to the reversal of current that travels to the poles (Brown, 2016). With the aforementioned components, an understanding of the mechanisms behind a brushed motor can be understood. When the motor is supplied with a power source, a current will flow through the armature. The combination of the current flow and magnetic field will create an electromagnetic force that will cause the armature coil to rotate. The commutator plates are significant as increasing the number of plates helps to smoothen the rotation of the motor. As commutators exhibit continuous rotational behavior, they are prone to damage with increasing use. Another component that is also prone to damage are the brushes due to motion; the brushes connect the commutator plates and armature (Mahoney, 2012).

Disadvantages of brushed motors include the wear and tear of the brushes and commutator. Along with this, the friction from the brushes can lead to the motor slowing down, shorter battery life, and lower power to weight ratio (Brown, 2016). As the RPM of our propellers will depend on the characteristics of the motor we use, it is important that the motor is able to run for long periods of time without significant wear and tear. Therefore, we considered the advantages of brushless motors over brushed motors to better choose the type of motor for our propeller.

Brushless motors operate in a similar fashion to brushed motors with the exception that the poles remain static as the magnets have been relocated onto the rotating section of the motor. Because the poles are stationary, there is no need for brushes. As a result, the degradation of the brushes and commutators is no longer a concern (Brown, 2016). In addition, brushless motors provide better results as they are not prone to the friction and voltage drop that is present in brushed motors. Therefore, a greater efficiency can be achieved when using a brushless motors (Mahoney, 2012).

As the benefits of brushless motors significantly outweigh the benefits of brushed motors, we conceded to choose brushless motors to run our propellers.

3.3 Servo Motors

Servomotors are defined as a brushless motor, with an extra addition of a position sensing device, such as a digital decoder (Northwestern University, 2017). Small in size, servomotors can yield significant torques. Servomotors contain a brushless motor (either AC or DC), a potentiometer, gears, and a controlling circuit. When the motor rotates, the resistance of the potentiometer changes such as that the control circuit can control the amount and direction of rotation (Jameco Electronics, 2017).

The servomechanism incorporates three parts - controlled device, output sensor, and a feedback system. In the closed loop system, a feedback signal is created from the comparison of the input and output signal. When there is a difference between the input and output signal, the servo mechanism will act to minimize this value, in spite of interferences such as noise (AdaFruit, 2017).

The direction of the servo motor is controlled through Pulse Width Modulation, a type of digital signal. Although PWM signals can only be controlled as high or low at a time, the percentage of time that a signal is high compared to when it is low can result in an output that appears analog in behavior (Jordan, 2017). When the PWM signal sets the servo position, the servo will move accordingly and remain in its position until commanded otherwise. When an external force attempts to change the servo position, the servo will be able to resist forces that are within their torque rating (Jameco Electronics, 2017). Therefore, in choosing the servomotor for our application, a consideration of the torque that will be needed must be estimated.

3.4 Manufacturing (Foam, Materials, etc.)

We considered several materials for use in our wheels and the chassis of our robot including urethane pour foam, Styrofoam with fiberglass, PET, ABS, PLA, carbon fiber, and aluminum. Due to the fact that keeping the overall weight of our robot as low as possible is of utmost importance, we decided not to move forward with materials

such as aluminum, PET, ABS, and PLA. After researching what Disney used for the chassis of their wall climbing robot, VertiGo, we discovered that they used carbon fiber. However, we decided not to proceed with carbon fiber or Styrofoam with fiberglass due to difficulty of manufacturing the shapes we needed. We ultimately decided to proceed with pour foam as our choice of material for both our wheels and chassis. Polyurethane foam is roughly able to float 60 pounds of "dead weight" per cubic foot making this a helpful material to keep our robot afloat during its water expeditions. We decided to purchase the urethane pour foam from AeroMarine Products, and we considered three different densities of urethane pour foam; two pounds per cubic foot, four pounds per cubic foot, and eight pounds per cubic foot.

All three densities of pour foam share some common characteristics and properties. They are all pour-in place foams and are all a closed cell, rigid urethane foam. The pour foam is at least 95 percent closed cell meaning it will absorb very little water. All of the urethane pour foams can be sanded into any shape desired after cured. A potentially useful characteristic of the foam is the compatibility with styrofoam. This is potentially useful for making parts of the robot lighter. Styrofoam is an open cell foam so it is not suitable for use in direct contact with water. Though urethane pour foam is comprised of 95 percent or more closed cell and is essentially "waterproof," no pourable foam can be 100 percent closed cell. However, it can be made 100 percent waterproof by painting it with marine grade paint or AeroMarine's 300/21 epoxy. Urethane pour foam is not UV resistant; therefore, if it is left out in the sun for extended periods of time, it is necessary that it be protected. Marine grade paint can also serve as a UV protectant for the foam in addition to completely waterproofing it. Furthermore, though the urethane pour foam on our robot is unlikely to encounter fuel, oil, and solvents, it is resistant to splashes of these as well. The urethane pour foam is heat tolerant and once cured, can withstand exothermic temperatures up to 250 degrees Fahrenheit and surface temperatures up to 200 degrees Fahrenheit. Under no circumstances, will the foam ignite. In the presence of heat exceeding its upward tolerance or flames it will char in a similar manner to wood.

AeroMarine's urethane pour foams is a combination of two liquids that expand when mixed together and are provided in two separate containers. When the two parts are mixed together (ideally for 20 to 30 seconds), they expand fully within a minute. They should not be mixed in or stirred with a porous material as this will contaminate

the foam. The optimal mixing conditions for the pour foam are 70 degrees Fahrenheit. Should the foam be mixed at a lower ambient temperature or the materials used for mixing and pouring are at a lower temperature, the foam will not expand the advertised amount of 20-25 times its liquid volume and thus, be denser than expected. If the two parts of the foam are mixed in an ambient temperature greater than 70 degrees Fahrenheit, the foam will expand more than it is suppose to, making it less dense than expected.

When considering the three density options for the urethane pour foam, we considered the purpose the urethane pour foam needed to serve in our project. Thus, we needed to ensure that the density we chose provided both excellent flotation for our robot while being durable and strong enough to serve as the carrier of our robot's electronic components. The lower density pour foams are best suited for floatation. The higher density pour foam was designed for use in repairing large areas such as boat decks. However, according to AeroMarine Products, the lower density pour foams are strong enough for boat floatation, barges, docks, canoes, kayaks, and pontoon among other things that float.

One urethane pour foam considered was the two pounds per cubic foot pour foam. This is its cured density. Its recommended uses include for use in filling cavities and voids for positive flotation and as an insulator for use around refrigerators and freezers. It has a parallel to rise compressive strength of 40 pounds per square inch, a tensile strength of 30 pounds per square inch, a shear strength of 30 pounds per square inch, and a flexural strength of 50 pounds per square inch. The cost of this particular density is $44 for a half gallon kit.

The second specific urethane pour foam considered has a density of four pounds per cubic foot. It has the same recommended applications as the two pounds per cubic foot density pour foam so it is still a good choice for positive flotation. It has a parallel to rise compressive strength of 60 pounds per square inch, a tensile strength of 115 pounds per square inch, a shear strength of 70 pounds per square inch, and a flexural strength of 115 pounds per square inch. This foam was slightly more expensive than the two pounds per cubic foot pour foam at a price of $47 for a half gallon kit.

The third and final urethane pour foam considered had a density of eight pounds per cubic foot. This is the toughest foam with the thickest cell walls and can thus support the most weight. It has a parallel to rise compressive strength of 95

pounds per square inch, a tensile strength of 220 pounds per square inch, a shear strength of 125 pounds per square inch, and a flexural strength of 335 pounds per square inch. This is the most expensive foam we considered at $49 per half gallon kit.

Ultimately, after experimenting on samples of the three different densities discussed above, we decided that the best choice would be the four pounds per cubic foot density as it strikes a balance between strength and low density.

Because the foam cannot be free formed, a mold is required. To be able to generate the unique shapes that are optimal for our robot's wheels and chassis we chose to use additive manufacturing to create the molds we used. The materials used for the molds are ABS or PLA. Because ABS and PLA are porous and the foam will adhere to both of them, the mold must be lined with either a release wax or a polyethylene bag such as a garbage bag. Since it is rather difficult to create a smooth surface on curved features of a mold using a garbage bag, we choose to use a mold release wax. Partall Paste Wax #2 is a reliable, non-silicone based release wax. A total of four coats must be applied of Partall Paste Wax #2 to prime the molds to be ready for pouring foam.

Another manufacturing consideration was being able to join separately molded foam parts together. After reviewing possible adhesives, the best option available appears to be Gorilla Glue. Both AeroMarine's urethane pour foam and Gorilla Glue are urethanes so therefore, should create a strong chemical bond upon their joining.

3.5 Batteries

There are four main categories of batteries in the market - Nickel Cadmium, Nickel Metal Hydrides, Lithium Ion, and Lead Acid Batteries (PRBA, 2017). Nickel Cadmium batteries (NiCd) are relatively more mature than the other types of batteries. However, even with the high level of knowledge on NiCd, the batteries still possess low energy densities. The NiCd battery is also not environmentally friendly due to the presence of toxic metals within. There are advantages, however, as NiCd are commonly used when long life, economy, and high discharge rate are important factors (Buchmann, 2017). Because NiCd batteries contain low internal resistance and encourage current conduction, these batteries are capable of supplying high current

discharge rates. As NiCd batteries have a long lifespan, they are economical as users will not have to spend more to replace these batteries (PRBA, 2017).

The second category of batteries is the nickel metal hydride batteries, which possess higher energy densities than the NiCd batteries. However, this advantage is counterbalanced by the NiCd's shortened cycle life. Another advantage of NiCd batteries is their lack of toxic metals (PRBA, 2017).

Lithium polymer batteries (or LiPo batteries) have been increasing in popularity in the RC industry as their high power and long run times appeal to users. There are three main advantages that LiPo batteries possess over Nickel Metal Hydride (NiMH) and Nickel Cadmium (NiCd) - lighter weight, higher capacities, and higher discharge rates. However, LiPo batteries exhibit shorter lifetimes than NiMh and NiCd batteries. Along with this, the sensitive makeup of the batteries make it a possibly hazardous materials if they are not stored properly (Schneider, 2016). Although there are disadvantages of using LiPo batteries, we opted with this choice due to the popularity of the battery, high power, and long run times.

Three main ratings need to be understood when using a LiPo battery - discharge (C) rating, capacity, and the voltage. A LiPo cell has a nominal voltage, or resting voltage, of 3.7 V. On the market, LiPo batteries of different voltage can be offered as combinations of the batteries in series and/or parallel lend to varying voltages. The voltage of a LiPo battery affects the RPM of an electric motor, such as brushless motors. Because we will use brushless motors with the propellers on our robot, the LiPo battery will also affect the rotational speed of our propellers. Therefore, a higher voltage battery contributes to a higher motor speed and a subsequent higher propeller speed (Schneider, 2016).

The capacity of a battery is another important measure in understanding the performance of LiPo Battery. Typically measured in milliamp hours (mAh), the capacity is a way of measuring the power a battery can possess. Therefore, higher capacities indicate a longer operation time. However, a longer operation time can lead to the motor heating up (Schneider, 2016).

The last factor to be considered is the discharge rating, or C rating. The C-Rating value indicates the speed at which battery discharge can occur safely. Calculating the C-Rating of the battery requires knowing the capacity of the battery. Once the aforementioned value is determined, the maximum safe continuous amp draw can be

determined. Operating at conditions higher than the calculated value can lead to the degradation of the battery. Therefore, this value is vital to know in order to promote optimal performance of the battery (Schneider, 2016).

LiPo batteries, due to their specialized chemistry, require special care and attention. When charging, LiPo batteries require constant current and constant voltage. NiMh and NiCd batteries, on the other hand, charge with the pulse charging method. Therefore, we would need to purchase a LiPo compatible charge to prevent any damage to the battery. Extra safety precautions, such as being present while the battery is charging in case of incident, must also be taken (Schneider, 2016).

3.6 XBee

Remote communication is an important aspect of the robot as the robot will be user-controlled. Prior to finalizing the decision on choosing XBee as the method of communication, we considered using RF transmitters/receivers from reputable companies such as Spektrum and Futaba. However, a lack of data on the range of the aforementioned transmitter/receiver pairs concerned us about whether we could control the robot at significantly far ranges. The review of the Sparkfun document on XBee specifications encouraged us to proceed with the XBee as the choice of user and robot communication.

The XBee is a small blue chip manufactured from Digi that allows communication in two modes - AT and API. In the AT mode, communication goes through the XBee; in the API mode, interaction with the XBee enable it to send commands and data to another XBee. Similar to the transmitter/receiver pairs from Spektrum and Futaba, most XBee modules operate at 2.4 GHz (Sparkfun, 2016). The XBee Series 2 modules also utilize Zigbee standard; the Zigbee standard provides the IEEE 802.15.4 protocol with enhancements such as authentication with valid nodes, data routing, and mesh networking capabilities (Frenzel, 2013). The XBee Series 1 modules are differentiated from XBee Series 2 as they only utilize IEEE 802.15.4 Protocol, which allows for point to point communication. Supported by such technology, the basic model of XBee Series 1 is capable of reaching 300 ft/100 m (Sparkfun, 2016). More advanced versions of the XBee Series 1 is the XBee Pro 900 XSC which can reach ranges of 28 miles. However, the price of this version would cost

$66.95, a significant cost when compared to the $24.95 cost of the XBee Series 1 (Sparkfun, 2016).

In determining which XBee to use, we considered three main categories - cost, ease of use, and function. The XBee Series 1 is the cheapest model of XBee that is available. Along with this, the XBee Series 1 is the simplest to use of the variety of XBee Models. Lastly, the XBee Series 1 allows for point to point communication, a functionality that we will need in controlling our robot. Therefore, from our observations from researching XBee characteristics, we successfully concluded that the XBee Series 1 would serve as the optimal remote communication method for our robot.

3.7 Distance Sensors (IR and Ultrasonic)

In general, Ultrasonic and Infrared sensors can be used for distance measurements without contact between the sensor and the obstacle. In this project, the distance sensor will be used to detect the presence of a wall-like obstacle, which will prompt the robot to switch into a wall-climbing mode. The question of which type of distance sensor to use was resolved through the review of several literature papers.

Infrared sensors operate by using an LED to send a light with a wavelength in the IR Spectrum. Then, the distance between the sensor and an obstacle is determined by analyzing the difference in intensity of the light before and after hitting the obstacle. In general, IR Sensors may offer higher resolution and faster response time than ultrasonic sensors (Adarsh, 2016 : pg. 2).

Ultrasonic sensors emit a high-frequency sound wave in the range of 40 kHz through a piezo-electric transducer. Then, the returning pulses of the emitted sound waves, or echoes, are tracked by a separate transducer in the ultrasonic sensor. This information is then converted to a proportional voltage variation that is used to indicate distance before an obstacle (Adarsh, 2016 : pg 2).

When choosing the model of sensor to use, we narrowed down two possibilities - the Sharp GP2Y0A21YKOF as an IR Sensor and the HC SR04 for the Ultrasonic Sensor). Several technical qualities separate the two sensors. The range of the Sharp IR Sensor is approximately 10 cm to 80 cm; the range of the HC Ultrasonic Sensor is 2 cm to 10 m. The beam pattern of the Sharp IR Sensor is narrow; the beam pattern of the HC

Ultrasonic Sensor is conical (Adarsh, 2016 : pg. 2). As both sensors meet our range requirements, it was necessary to evaluate whether the IR Sensor or Ultrasonic sensor would work better under our operating conditions (outdoors and indoors, different types of walls). An experiment conducted by the Department of Electronics and Communications Engineering at the Amrita School of Engineering on the two aforementioned sensors would serve as an important tool in determining which model to use.

In the experiment, the IR sensors and Ultrasonic sensors were mounted on a moving vehicle and controlled by an Arduino. As the setup of the robot in this experiment is similar to our robot configuration as well, the results of this experiment do pertain to our robot. The distance readings from the IR and Ultrasonic sensor will be compared to the distance measured by a rotary encoder that has been attached upon one of the wheels in the robot. Different materials selected for analysis included cardboard, paper, rubber and tile (Adarsh, 2016 : pg. 3). Some possible materials that our robot may traverse on that were tested in this experiment include cardboard, wood, plastic, and tile.

Cardboard is a fibrous material that is often at least 0.30 millimeters in thickness. The Ultrasonic sensor was able to provide a steady distance reading after it was 15 cm away from the material. The IR Sensor achieved the same performance capabilities when it was 10 cm to 15 cm away from the obstacle. Although the Ultrasonic sensor provided steadier readings for greater ranges, it provided higher distance values than the actual distance travel due to the cardboard absorption of sound waves. Overall, the Ultrasonic Sensor presents steadier readings when presented with an obstacle made of cardboard (Adarsh, 2016 : pg. 4).

When the IR and Ultrasonic sensors were presented with wood, the Ultrasonic Sensor showed very minimal deviations from the distance values measured by the rotary encoders. On the other hand, the IR sensor measured distances that varied significantly from those measured by the rotary encoders. Because of the wood's rough surface, the reflection of light waves was negatively affected whereas the reflection of the sound waves were benefited. Therefore, Ultrasonic Sensors prove to supply more accurate distance readings when encountering wooden surfaces (Adarsh, 2016 : pg 5).

The smooth surface of plastic enables sound waves to better reflect from it, thus contributing to stable distance readings from ultrasonic sensors. Although the qualities

of plastic are beneficial for ultrasonic sensors, they hinder the readings of IR sensors as the transparency of plastic prevents reflection of light waves. Therefore, Ultrasonic sensors are best suited for measuring distances from a plastic obstacle (Adarsh, 2016 : pg 5).

When traveling towards a material composed of tile, the ultrasonic sensor and infrared sensor readings were relatively similar in accuracy when compared against the rotary encoder's measurements. Therefore, both sensors worked well with measuring the distance to an obstacle composed of tiles (Adarsh, 2016 : pg 6).

The materials described above (wood, tile, plastic, and cardboard) were chosen as they were assumed to be the most similar materials to the ones we would encounter. As ultrasonic sensors tend to produce better results with the aforementioned materials, it seemed reasonable to choose an ultrasonic sensor over an IR sensor (IntoRobotics, 2013). Along with this, ultrasonic sensors are advantageous as they are not as sensitive to changes in sunlight. As our goal is to create a robot that can travel both indoor and outdoor walls, an infrared sensor might be disadvantageous to use due to variations of light. Therefore, we opted to purchase a Ultrasonic HC SR04 Sensor.

3.8 IMU Sensor

Another important sensor that we will utilize in our robot is an Inertial Measurement Unit. A 6 DOF IMU consists of an accelerometer and gyroscope. A 9 DOF IMU will consist of an accelerometer, gyroscope, and magnetometer (Starlino, 2009). Because the Disney Vertigo robot uses a 6 DOF IMU, we concluded that a 6 DOF IMU would be sufficient for our robot as well (Beardsley, 2012). To efficiently use the IMU, we developed our understanding of the unit, detailed in the following research.

An accelerometer is an electromechanical device whose function is to measure acceleration forces, including static (e.g. gravity) and dynamic forces. With an accelerometer, the robot is better equipped to interpret its surroundings (Goodrich, 2013). Working hand in hand with the accelerometer is the gyroscope, another type of electromechanical device that measures rotational motion,specifically angular velocity. A gyroscope can generally measure the rotational behavior about three axes - x, y, and z (Sparkfun, 2017). In summary, linear acceleration is collected from the accelerometer and angular velocity is collected from the gyroscope. The data collected

from the aforementioned sensors are then delivered to a microprocessor on the IMU. In the end, the vehicle coordinates can be successfully estimated (UMD, 2017). The Disney VertiGo uses the information gathered from the IMU, coupled with the data from the distance sensors to determine the robot's orientation in space; these functions can be mimicked in our robot as well.

3.9 Arduino and Raspberry Pi

An important aspect of our robot is the type of hardware that we will choose to perform the necessary calculations of our robot. There were two types of hardware that we considered - the Arduino and the Raspberry Pi. The Arduino is a type of microcontroller board that is derived from the ATmega328P. Fourteen digital input/output pins, 6 analog inputs, USB port, 16 MHz quartz crystal, power jack, ICSP header, and reset button are contained on the Arduino Uno, one of the various models offered by Arduino (Arduino, 2017). The Raspberry Pi is a miniature computer that can plug into the TV and a keyboard (Raspberry Pi, 2017). Narrowing down the distinguishing characteristics between the two, as detailed below, allowed us to make an informed decision regarding which hardware to use.

The Arduino, although not as powerful as the Raspberry Pi, is advantageous in different aspects. Unlike the Raspberry Pi, the Arduino would not require additional software or OS to run. As a result, the Arduino provides a simpler interface between the software and electronic components. Reading a sensor, for example, can be achieved with a few lines of code written in the Arduino application. Trying to achieve the same with the Raspberry Pi would require the installation of additional libraries and softwares. The Arduino is also advantageous as it can be turned on and off without worrying about damage to the hardware. As long as the Arduino is plugged into a power source, the Arduino will run the code. Unlike the Arduino's plug and play characteristic, the Raspberry Pi needs to be properly shutdown to prevent harm to the applications and/or hardware (Circuit Digest, 2017).

One of the main advantages that the Raspberry Pi offers over the Arduino is its ability to run multiple tasks simultaneously. Along with this, the Raspberry Pi is 40 times faster than the Arduino. Therefore, activities such as sending a remote live feed from a camera would be significantly better with the Raspberry Pi. Overall, the

Raspberry Pi is a powerful system that is suitable for projects of a complex level (Circuit Digest, 2017).

In conclusion, we decided to use the Arduino as our board of choice for the robot. Because of our limited software experience, we believed that the Raspberry Pi would be difficult to work with. Along with this, the Arduino provides a simpler means of interacting with electronic components such as sensors and motors. Although the Raspberry Pi would allow us to add a remote live feed to our robot, we believed that Arduino's advantages outweighed this. Therefore, through a detailed research of Arduino and Raspberry Pi, we came to a final decision of using the Arduino.

4 Preliminary Design

4.1 Concept Generation and Evaluation

4.1.1 First Design

Spherical Wheels

The spherical wheels maximize the buoyant force acting on our vehicle while traveling on water. Furthermore, the wheels are made of urethane pour foam in order to minimize the weight. Since many issues would arise with the point contact on the ground with the wheel such as lack of friction, the wheel has a revolved extrusion cut on the surface so that a rubber band could be inserted in this slot. The rubberband increases the contact area and provides more traction to aid in the wall climbing process. The radius of the spheres are selected based on the formula derived for equilibrium. Based on design objectives, the theoretical radius is selected to be approximately 6 inches so that one third of the spheres submerge when the vehicle is in the water.

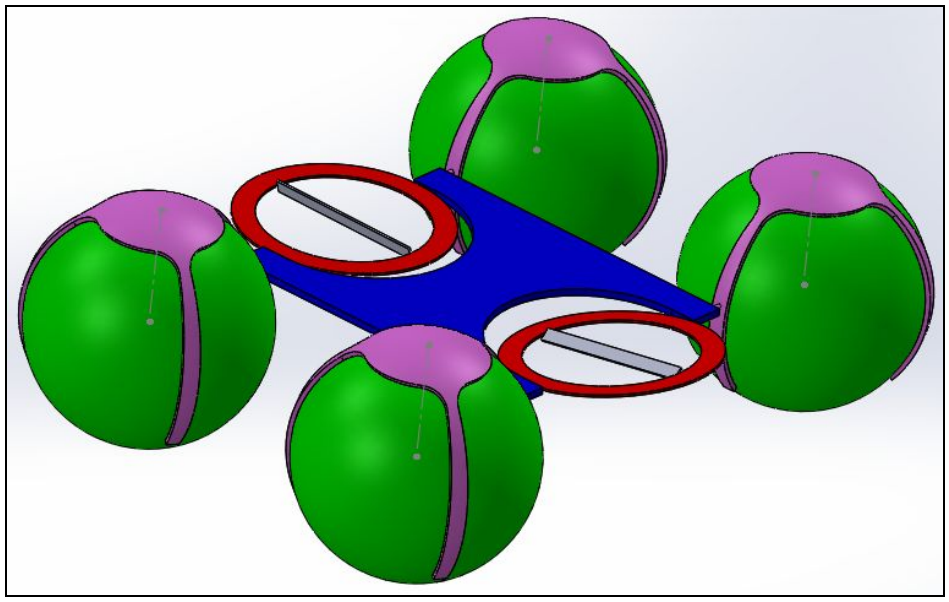


Figure 14: Omnidirectional spherical wheel design

Chassis

The X-shaped chassis was designed such that there was no interference between the propellers and the spherical wheels. Furthermore, in order to minimize the weight of the chassis while maintaining enough mechanical strength to support the deadweight, several abaqus FEM analysis are performed. The FEM analysis are used to determine whether the structure fails under the specified loading conditions. Since weight is the most significant constraint, these analysis are done with varying chassis thickness. In addition, several materials were chosen for the chassis including ABS plastic and foam. Foam was the preferred material due to the fact that it is lightweight and its strength properties met our loading requirements. The largest surface area of the chassis was located towards the center which was used to place the larger components such as the Arduino and the breadboard. Furthermore, each branch of the chassis was used to attach motors and to mount the wheels.

Propeller and propeller casing

The propeller casing was designed in such a manner that it holds two motors. One motor was attached at the middle of the casing and it was used to drive the propeller. The second motor was used to rotate the case. These two motors provide two independent degrees of freedom that are essential to achieve stability during wall climbing. The motor used to rotate the case is designed based on newton's third law. Since the shaft is fixed relative to the propeller casing, it provides an equal and opposite moment to that applied by the motor which creates the rotation. Due to the

forces that are used for this mechanism to operate, a more detailed finite element analysis was needed to determine the structural integrity of this design.

Spherical Wheel Frame

The frame was designed such that the axis of rotation of the spherical wheel was aligned directly with the rotation of the motor. In other words, the design intent was to place the motor directly above the shaft to transmit the power through the shaft directly to the rotation of the sphere. Not only would this reduce the power output since the moment of inertia is minimum at this location, but it also provided a more maneuverable design. A frame that accomplished these tasks was the y-frame design. The y-frame connected to the spherical wheel by a revolute joint along the center of the wheel. Furthermore, it also attached to the chassis by a revolute joint. At the chassis, the head of the shaft of the y frame is directly attached to the motor that is aligned with the rotation of shaft.

Design Overview

Overall, the first iteration of the design met the initial objectives. However, one of the disadvantages of this model was that the center of mass was located too high, which called for a higher required thrust in order to achieve wall climbing. For this reason, a new design was made to improve upon the shortcomings of the first model.

4.1.2 Second Design

The feedback obtained from the first iteration of the model provided good insight into other requirements that needed to be achieved which lead to the re-iteration of the model. One of the potential issues was that the center of mass was too high with respect to its wheels. This characteristic of the first model created a high thrust requirement in order to achieve vertical wall climbing. The new design objective was to reduce the thrust required by the propellers. For this reason, it lead to the re-design of the wheels and the chassis such that the center of mass is lower.

Tapered-cylindrical wheels

The design and geometry of the wheel had to be reconsidered, as the new objective was to lower the center of mass. As before, the fundamental design goal for

the wheel remained the same. The goal is to design a wheel that strives for a balance between surface contact and large volume. A large volume for the wheel is also essential in order to provide the required buoyant force to support its weight when submerged in water. These design specification are achieved with tapered-cylindrical wheels as it provides enough ground contact to achieve traction while maintaining its maneuverability. Furthermore, the tapered design achieves significant fluid displacement in water. The tapered cylindrical wheel is connected to the chassis by a four linkage mechanism at the center of the wheel thus significantly lowering the center of mass.

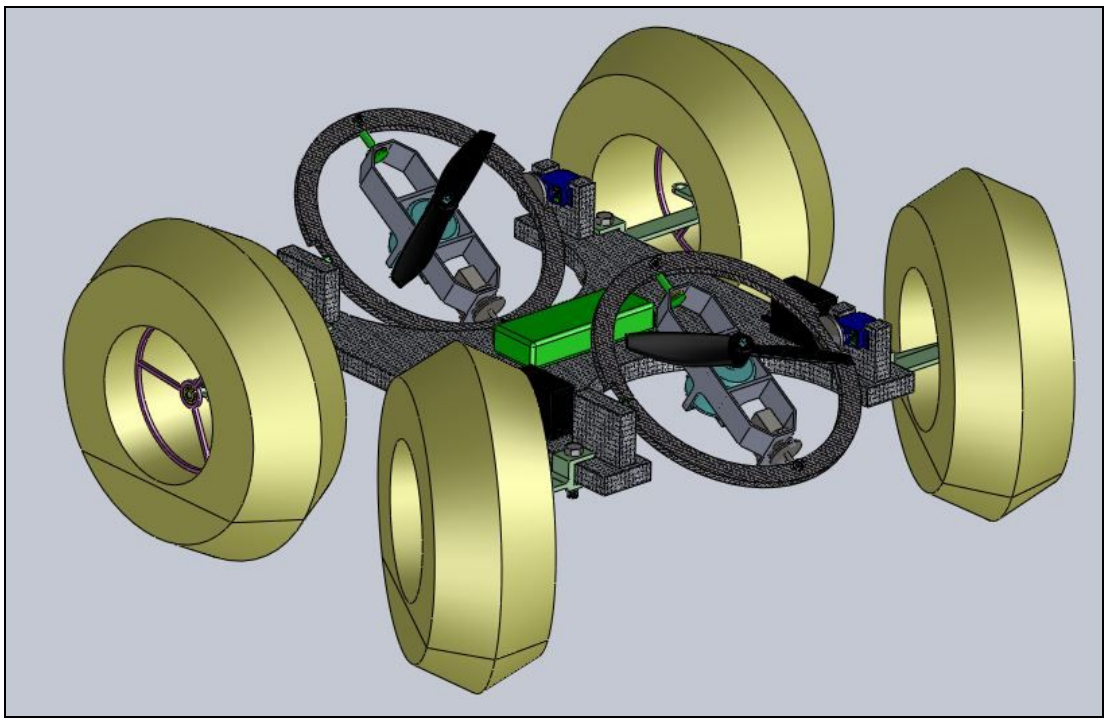


Figure 15: Tapered cylindrical wheels on the side. The black line on the wheels indicates the level at which water level would reach assuming final weight of 2.0kg

Steering mechanism

The steering mechanism was significantly more complex compared to the previous design. The steering mechanism consisted of a four bar linkage, where the motor drove link two. Our particular interest was to accurately predict the rotation angle of linkage 4 since it corresponds to the cylindrical wheel. This analysis was achieved using Newton-Raphson numerical technique. In this way, based on specific rotation of link 2, the rotation of link 4 can be tracked. Future analysis included the torque required to drive the linkage and were to be done once the RPM and the motor

were chosen. The steering mechanism maintains the center of rotation of the wheel fixed just as in the previous model.

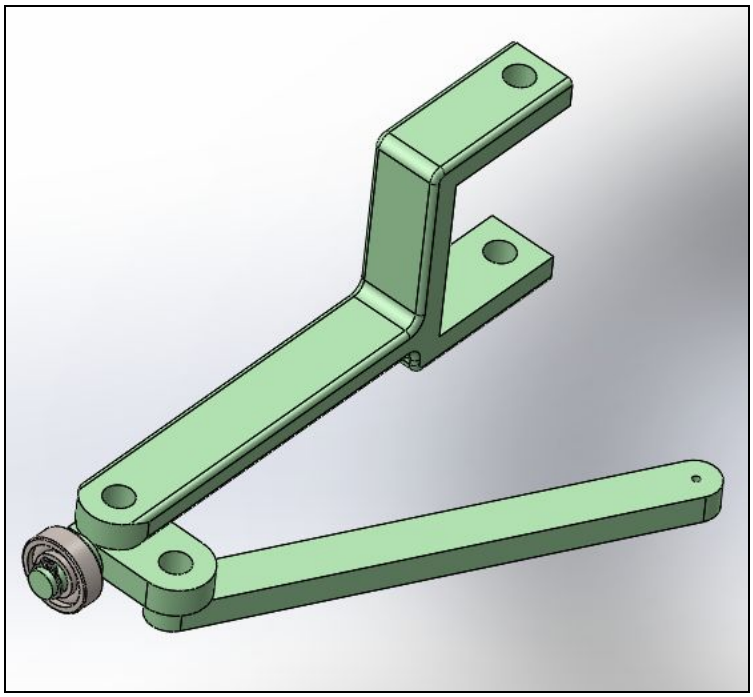


Figure 16: Four-bar linkage steering mechanism

Chassis

The chassis shape was changed to a more rectangular version as a rectangular shaped chassis is used to accommodate the components in a similar manner. It has the propeller slots and sufficient surface area to mount the components.

Figure 17 : First 3D printed chassis mold (the final design was much larger than this version)

4.2 Selected Design

Due to the significant amount of increase in available/effective thrust that results in lowering the center of mass, the second model was selected over the first design. The tapered cylindrical wheels were replaced by fully cylindrical wheels for easier manufacturability while maintaining the same properties as the first model. They provided a significant amount water displacement and keep a balance between surface contact to achieve maneuverability and friction to achieve wall climbing. The only difference is that this increased the complexity of the steering mechanism. The selected steering mechanism is modeled as a four bar linkage. In order to calculate the rotation of the links based on an input angle on the driver link, Newton-Raphson numerical technique is used. In addition, in order to optimize the angle of rotation of the wheel, optimization is performed by varying the lengths of the links.

4.3 Preliminary Analysis

4.3.1 Free body analysis of vehicle on wall

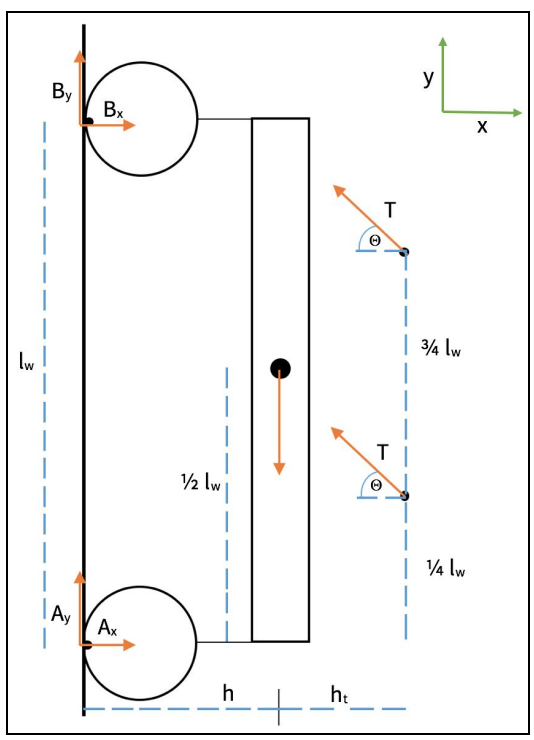


Figure 18: Free body analysis of still vehicle on wall

Variables used in free body diagram:

l_w = wheelbase length

h = distance of center of mass of vehicle from wall

h_t = height of point of thrust application (by propellers) above center of mass

Θ = angle of thrust applied with respect to negative x-axis

A_x, A_y = reaction forces by the wall at lower wheel point of contact

B_x, B_y = reaction forces by the wall at upper wheel point of contact

Assumptions:

1. The center of mass of vehicle is in the center of the chassis.
2. Both the propellers apply equal thrust in the same direction.
3. The force analysis is performed for the case when the vehicle is standing still on the wall, that is, zero velocity in both directions.
4. The analysis is also for the case in which the vehicle is just about to fall off from the wall, to get an idea of the minimum thrust needed to keep it on.

Performing free body analysis in the x-, y-, and rotational directions:

$$\Sigma F_x = A_x + B_x = 2T\cos\theta$$

$$\Sigma F_y = A_y + B_y + 2T\sin\theta = mg$$

$$\Sigma M_A = -B_x L_w - mgh + l_w T\cos\theta + 2T(h + h_t)\sin\theta = 0$$

As assumption number 4 states, the vehicle is just about to fall off the wall. Hence, the reaction forces at point A can be taken as equal to zero. Specifically,

$$B_x = 0\,,\ B_y = 0$$

Solving the three equations for T, A_x, and A_y:

$$T = \frac{mgh}{l_w\cos\theta + 2(h+h_t)\sin\theta} \qquad A_x = 2T\cos\theta \qquad A_y = mg - \frac{2mgh}{\frac{l_w}{\tan\theta} + 2(h+h_t)}$$

4.3.2 Abaqus Analysis

In this model, a representation of the deadweight is simulated. The deadweight created by the mounted components is assumed to act as a concentrated 2 lbf at the center of the vehicle.

Also, the joints are assumed to be fixed. From the results, the deflection is found to be 0.04 inches which is a tolerable amount for the given dimensions of our model. Furthermore, as seen from Figure 11, it occurs at the center as expected.

Material	: Foam	Modulus of Elasticity	: 125 psi
Thickness	: 1 inch	Load	: 2 lb-f (at center)
Boundary conditions	: Four ends fixed		

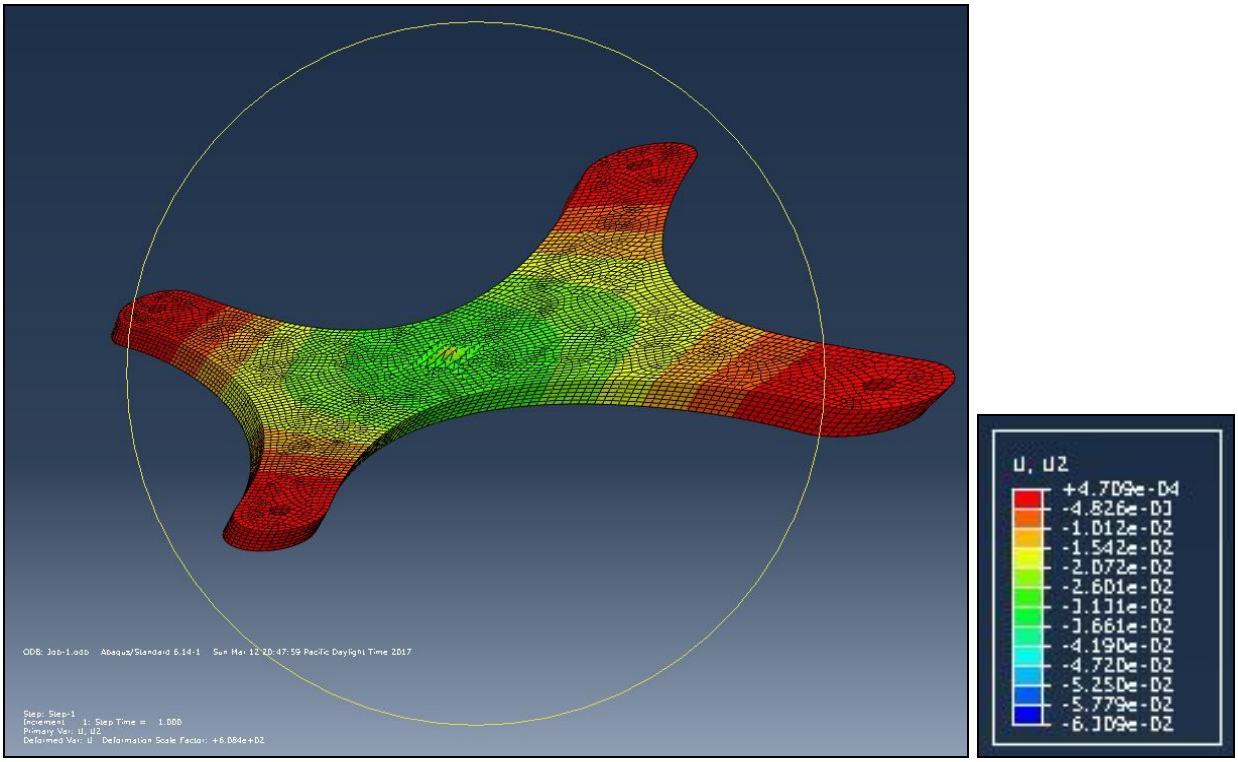


Figure 19: Abaqus analysis on chassis and legend

4.4 CAD Models

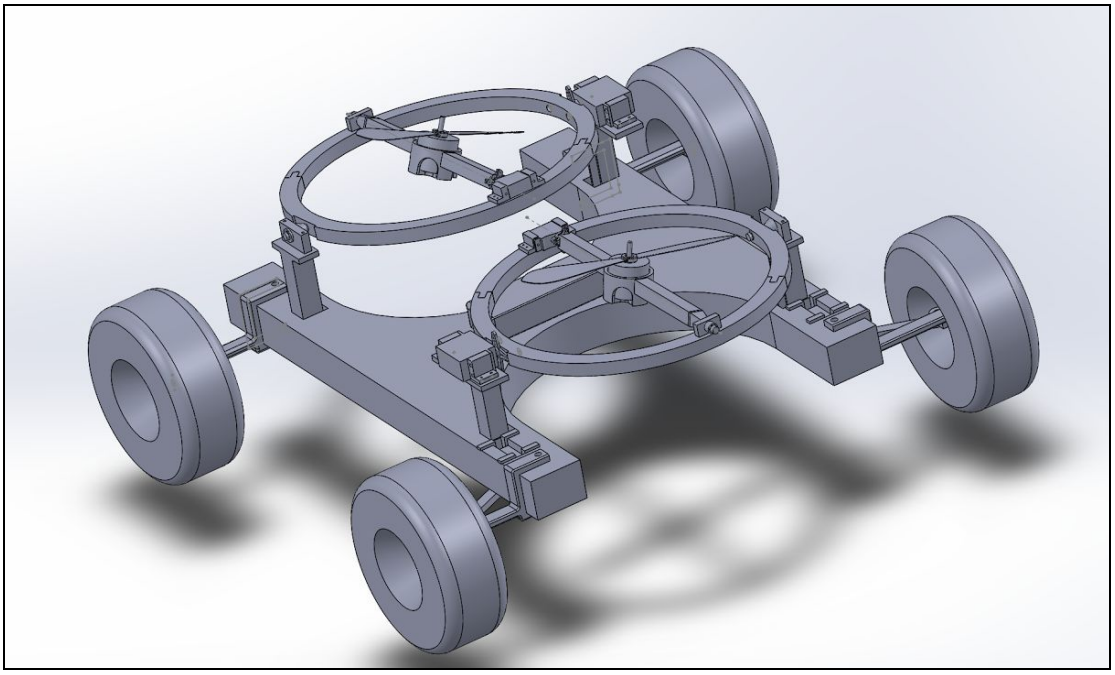


Figure 20: Completed CAD model - Isometric view

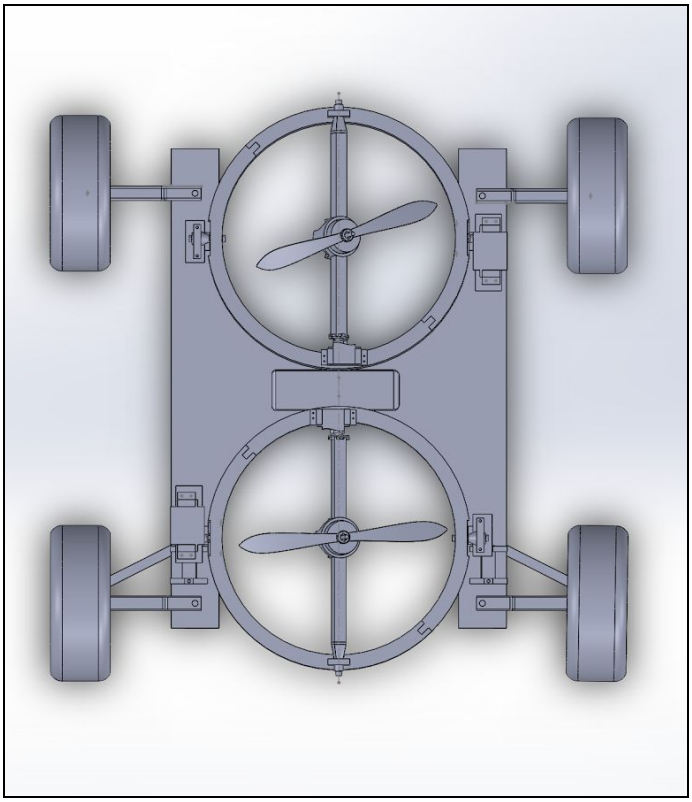


Figure 21: Completed CAD model - Top view

5 Detailed System Design

5.1 Overview of Detailed System

In the following sections, we will detail the several aspects of our robot. In particular, we will focus on giving an overview of our mechanical, electrical, and control designs.

The mechanical steering system consists of a four-bar linkage mechanism for each of the front wheels. Both of the front wheels are operated using parallel steering algorithm. Due to the physical constraints of steering linkages and the wheel, there is a small steering angle range, thus our robot could only perform wide turns. Due to the physical design of the steering mechanism, it was not possible to implement an Ackermann steering mechanism for more effective turning. However, the parallel steering algorithm provided some advantages. Not only is it relatively simple to program, but it was fairly responsive system and it is easy to implement. The Arduino was the microcontroller used to control the steering motor servos, the roll and pitch servos and the speed of the brushless motors. This was powered using a lithium polymer (Li-Po) battery with a discharge rating of 65 C. The brushless motors were controlled through the electronic speed controllers (ESC). The ESC functions to gradually increase the speed of the brushless motors, to control the direction of the rotation, and for low battery voltage cutoff.

5.2 Detailed Drawings

Propeller ring

The propeller ring dimensions are much bigger than what the 3D printers have capability for, therefore this component was split into two separate parts.

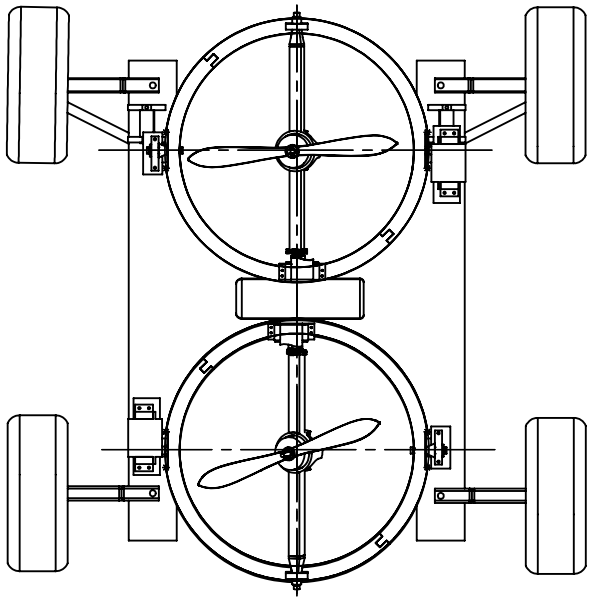
Propeller pillars

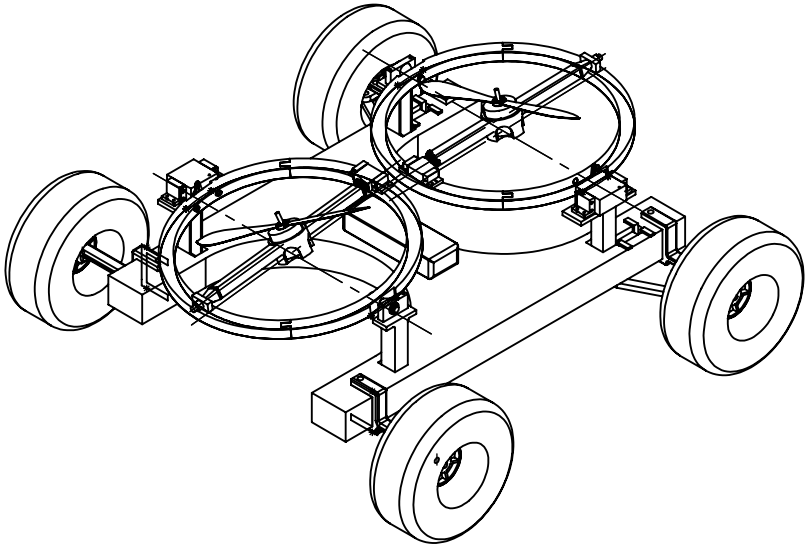
Since weight is a significant constraint for this project, several of the 3D-printed components were made hollow. The propeller pillars for example, are made such that the inside of it is hollow.

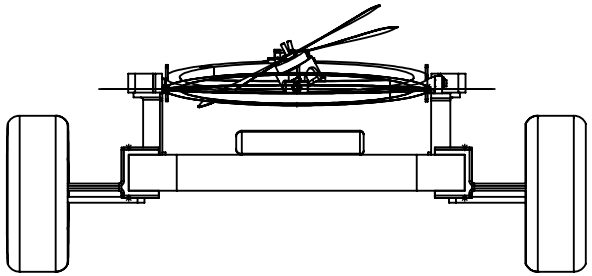
Propeller housing

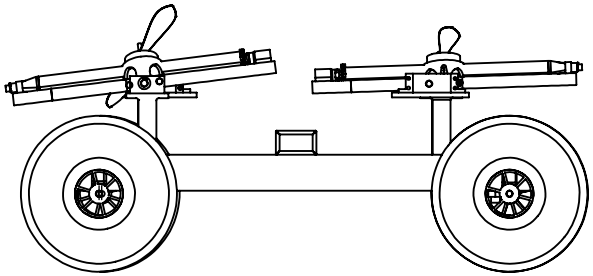
As seen from the design iteration process for the propeller housing, it is evident that the propeller housing underwent significant changes since its beginning phase. Although the first design was capable of providing a stable support for our chosen propeller, it was not optimal for the weight constraints as it had a lot of unnecessary weight and it was also very bulky. For this reason, in order to minimize the weight, a cylindrical housing that exactly encloses the propeller was chosen instead. In addition, due to to the constraints imposed by the 3D printer capabilities, the arms of the propeller housing were printed separately from the propeller housing itself. In order to make the attachment between the propeller housing and arms more secure, rectangular extrusions were placed at opposite ends across the propeller housing. The cross sectional area of the propeller housing arms were designed to attach into this cross section.

The following section contains engineering drawings of the full assembly of the robot along with sub assemblies in areas where significant design decisions were made.









		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

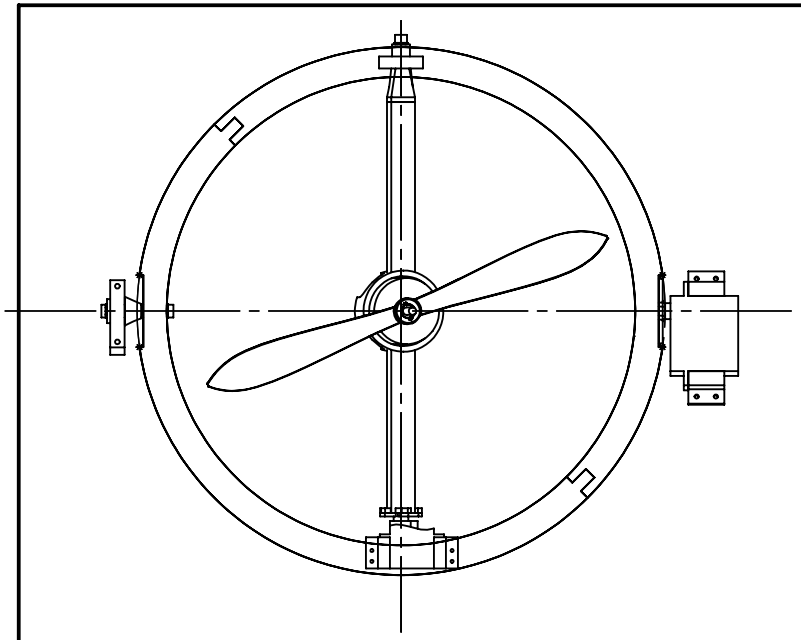
TITLE:

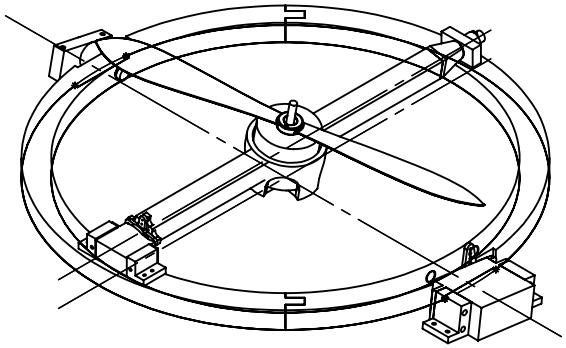
SIZE A | DWG. NO. Full Assembly | REV

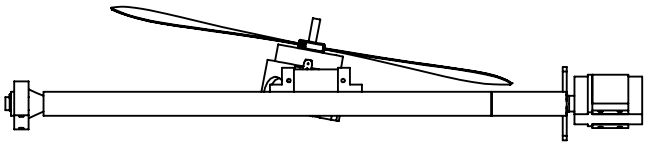
SCALE: 1:12 | WEIGHT: | SHEET 1 OF 1

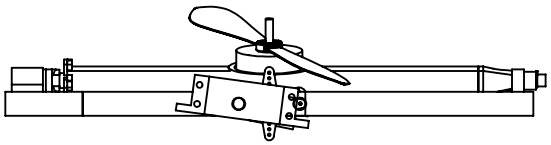
PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.









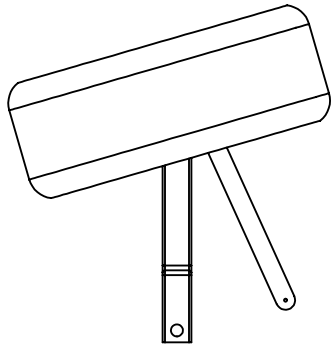
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

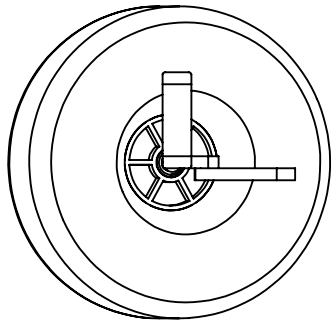
TITLE:

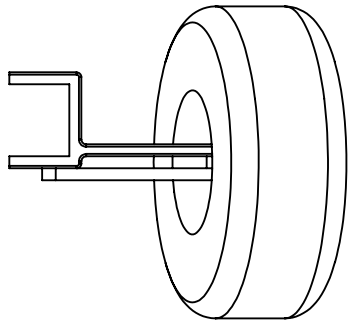
SIZE	DWG. NO.	REV
A	Propeller Housing Assembly	
SCALE: 1:8	WEIGHT:	SHEET 1 OF 1

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.







		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

TITLE:

SIZE	DWG. NO.	REV
A	Wheel Assembly	
SCALE: 1:8	WEIGHT:	SHEET 1 OF 1

5.3 Engineering Analysis

5.3.1 Steering Mechanism Link Design

Optimization Analysis: R1y													
R2	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127	0.0127
R3	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963	0.0963
R4	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150	0.0150
r1x	0.0949	0.0948	0.0949	0.0949	0.0949	0.0949	0.0949	0.0949	0.0949	0.0949	0.0948	0.0948	0.0949
r1y	0.0260	0.0250	0.0240	0.0230	0.0220	0.0210	0.0200	0.0190	0.0180	0.0170	0.0160	0.0140	0.0150
Angle Difference	121.7462	119.0648	117.4860	116.4626	115.8058	115.3876	115.1697	115.0926	115.1390	115.2844	115.5042	116.1419	115.8019

Table 1: Optimization Analysis for vector $R1_y$

From the analysis performed, it was determined that the angle of rotation for the wheel is maximized when the y component of link 1 is 0.026 meters.

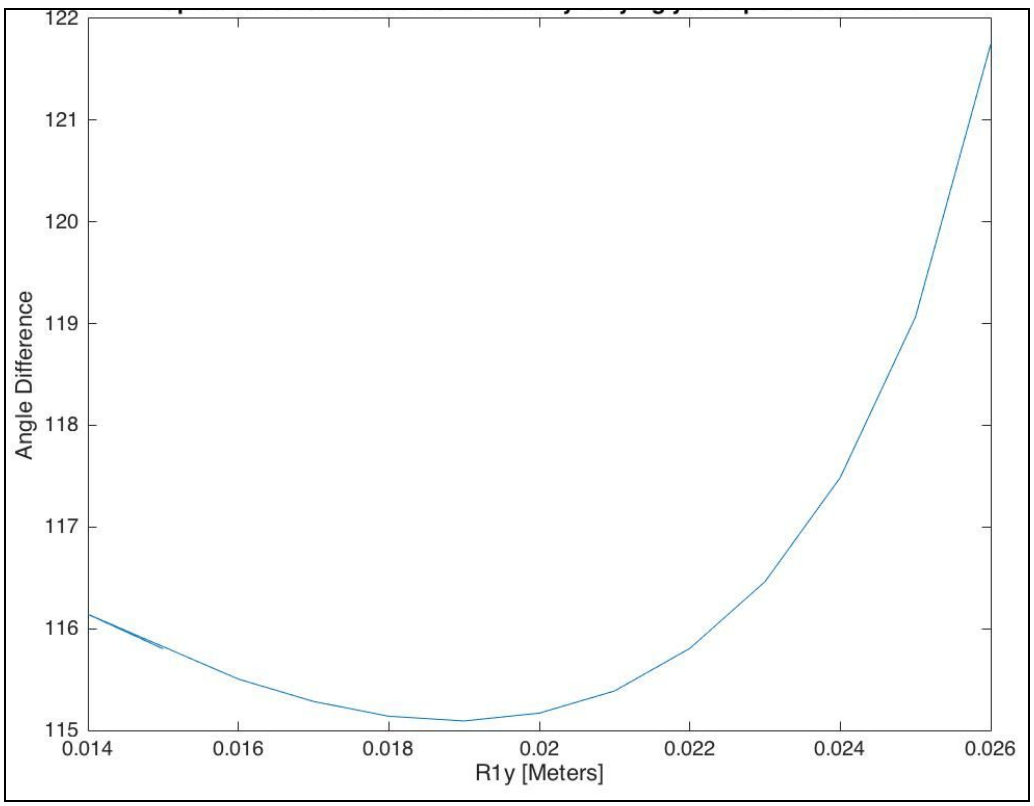


Figure 22: Optimization of rotation of the bar by varying y-component of Link 1

5.3.2 Buoyant Force Equations

For simplicity, the wheels are assumed to be cylindrical although in the actual design they are tapered. Furthermore, in order to achieve smooth water to ground transition, this equation is derived on the basis that 30% of the wheel is submerged.

$$\sum F_y:\ m_{chassis}g\ - 4F_{bouyant} + 4F_{wheel} = 0$$

$$m_{chassis}g - 4V_{disp}\rho_{water}g + 4V_{water}g = 0$$

$$4\rho_{foam}\pi r^2 t\ g + m_{chassis}g = 4(.30)4\rho_{water}\pi r^2 t\ g$$

$$r = \left(\frac{m_{chassis}}{t(.30\pi\rho_{water} - 4\pi\rho_{water})}\right)^{\frac{1}{2}}$$

$$r = \left(\frac{5m_{chassis}}{2t\pi(3\rho_{water} - 10\rho_{foam})}\right)^{\frac{1}{2}}$$

where:

$m_{chassis}$: the mass of the chassis and all the mounted components (excluding wheels)
ρ_{water} : density of water
ρ_{foam} : density of the foam

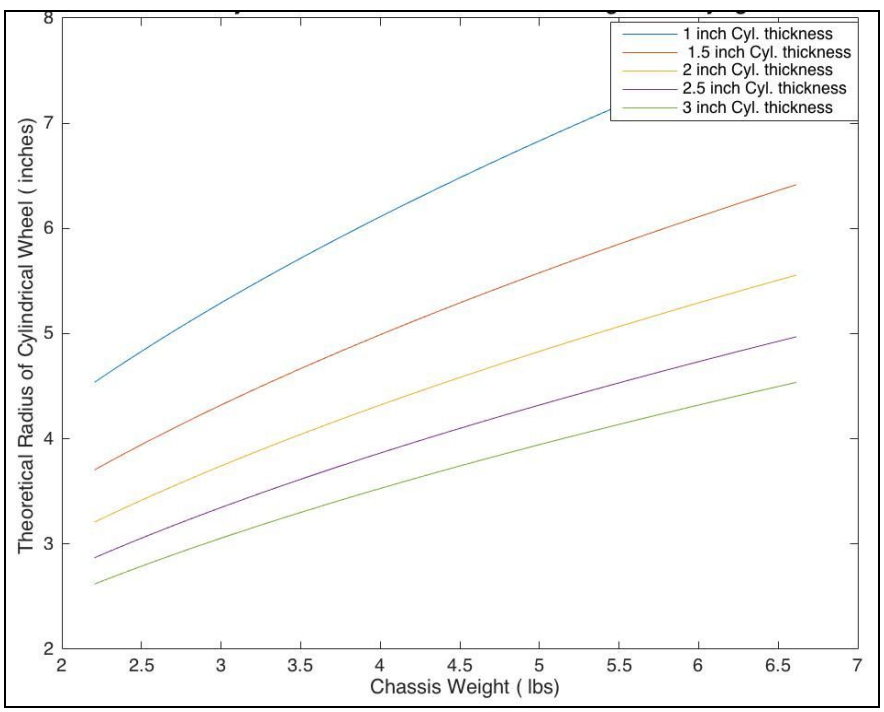


Figure 23: Theoretical radius of a wheel for given chassis weight with varying wheel thickness

As shown from the graph above, each plot represents a specified cylindrical wheel thickness. Once the chassis weight is experimentally determined, the plots can

be used to determined the appropriate radius for a specified thickness. Recall that the cylindrical wheel is assumed to be 30% submerged in water for a smooth water to land transition.

5.4 Electronics System Design

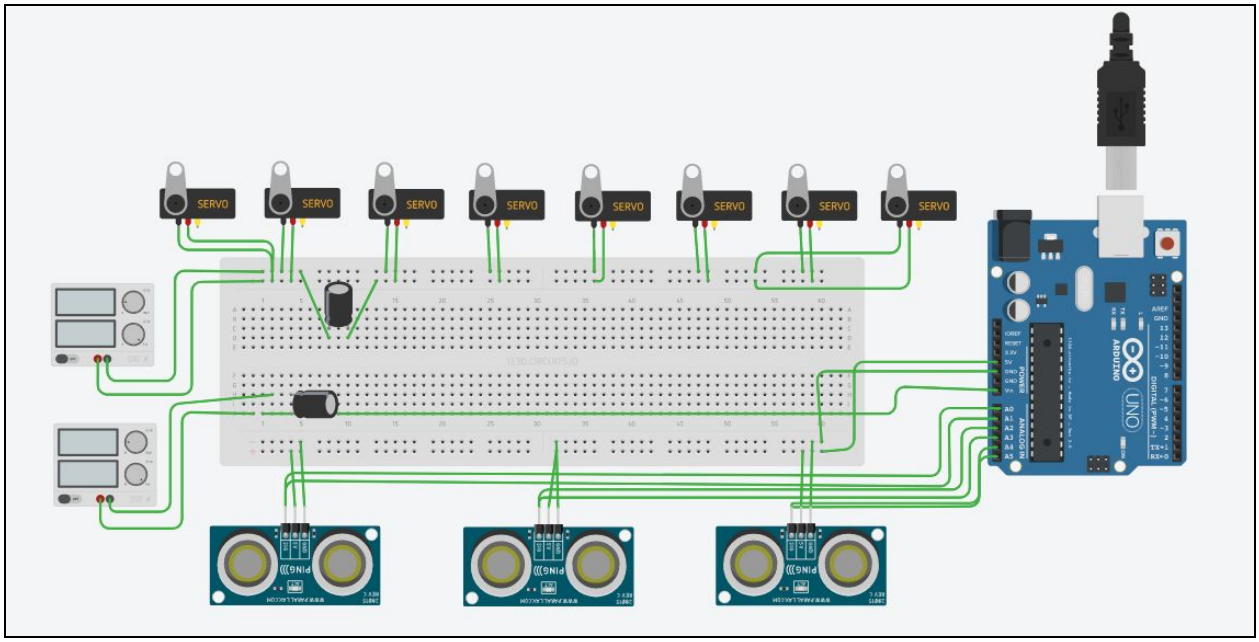


Figure 24: Electronics schematic detailing the objects connected to the Arduino

Electronics constituted a major part of the project, serving as the brain that controlled the propellers and the servos in order to achieve movement. As previously mentioned, the Arduino Uno was used as the microcontroller on board to perform the functions of controlling the steering servos, the roll and pitch servos, and the speed of the brushless motor. The Arduino was also responsible for collecting the distance data from the ultrasonic sensors. The analog pins on the Arduino were used to collect sensor data while the digital pins were used to send signal instructions to the various servos and brushless motors. All of the circuitry was powered using a 2700 mAh Lithium Polymer (Li-Po) battery with a discharge rating of 65 C manufactured by Turnigy.

As depicted in Figure 18, the roll servos were attached to pin 8 and pin 9 on the Arduino. Roll servos were used in this project to overcome the gyroscopic torque that we observed. Without the roll servos, we observed that the robot would often rotate without any input from the steering. Therefore, to overcome this effect, we placed the roll servos in order to prevent our robot from turning unexpectedly. The pitch servos were attached to pin 10 and pin 11. These servos would act to control the angular position of the propellers and affect the subsequent thrust generated. Along with this, pins 6 and 7 controlled the brushless motor's RPM. The brushless motors were

responsible for generating the torque of the robot. Lastly, the servomotors that were used for steering were attached to pin 4 and pin 5. Due to the absence of a physical Ackerman steering linkage, we concluded that it would be far too difficult to implement a virtual steering linkage. Along with the large tolerances that we had to consider for our steering mechanism, we decided that trying to implement a virtual Ackerman steering would be too difficult and lead to inaccuracies. Therefore, both servomotors were given angle values that corresponded to the final effect of the two servos turning in the same direction. For the ultrasonic sensors, all the sensors were connected through pins A0 to A5. These ultrasonic sensors were powered by the 5 V source in the Arduino. The Arduino was powered by a voltage regulator that would decrease the voltage from the battery to 11 V. Since the Arduino can safely handle 7 V to 12 V of external power to power it, 11 V was a decent amount of power to supply to the Arduino. All of the servo motors were controlled by the power supply that was supplied from the UBEC, another voltage regulator that can supply 5 V and 3 A.

The brushless motors were controlled through Electronic Speed Controllers (ESC) which received instructions from a digital pin on the Arduino. The ESCs had to be programmed for the following behaviours:

- Soft start, meaning a slow acceleration startup which prevents the motor from transmitting a large torque on the frame
- Direction of rotation, the motors were programmed to spin in opposite directions to cancel a net torque that would otherwise be produced. The propellers were also chosen to be opposite in the rotation
- Low battery voltage cutoff, since low voltages on a Li-Po cell can permanently damage it, the ESC has a built-in function to stop drawing current from the battery once it passes below a predetermined threshold

5.5 Control System Design

The robot was controlled through the microcontroller, Arduino uno. The skeleton code consisted of a main loop the split based on whether or not the robot was on the wall or off the wall. If the robot was not on the wall, then the code would split into two loops - whether the button for wall mount setup has been pressed or not

pressed. If the button for the wall mount has not been pressed, then the user will have complete control of the robot - both steering of the servomotor and speed of the propeller. This scenario was named region 4.

However, if the button is pressed, the robot can begin the wall mounting process. If the button is pressed, the code will split off into two possibilities which depend on the robot's distance from the wall. In only the first iteration upon entering the button is pressed portion of the code, the previous distance of the robot will be set by looking at the robot's current distance. Then, in the following iterations of the loop, the code will check the validity of the measured distance from the wall of the robot by comparing it to the previous distance. If the difference in the measured distance from the previous distance is not a valid distance, it will return a value of negative one. Whether or not the traveled distance is valid will be based on the formula - maximum speed of the robot times the number of seconds that passed since the last iteration of the code. The aforementioned portion of the code was to insure that the random spikes in values that the ultrasonic sensor is sometimes susceptible to would be eliminated in an efficient manner. Once a valid distance has been obtained, the next step is determined by whether the distance read is greater than the "far wall threshold" or less than or equal to the "far wall threshold." The "far wall threshold" is a variable that was given the value of 43 inches. As a side note, this value was chosen since past this point the ultrasonic sensors became increasingly more inaccurate when we tested them. If the distance to the obstacle is greater than the "far wall threshold," then the robot is in what we call region 3, meaning that it takes user inputted values for steering but does not take user input for the propeller speed. On the other hand, if the distance to the obstacle is less than or equal to the "far wall threshold," then the code proceeds to check whether or not a second button is pressed. If this second "enable wall mount" button is pressed, it indicates that the robot is allowed to mount the wall. If it is not pressed, but the first button, "enable wall mount setup" is still pressed, then the user maintains control of the steering and the robot will not attempt to mount the wall. However, if the second "enable wall mount" button is pressed and the robot is in region two, meaning it is within 43 inches from a wall, then the user loses control, and it autonomously drives itself toward the wall, proceeding to bump into the wall until both ultrasonic sensors read the same value within a margin of error. When both

ultrasonic sensors read the same value, this indicates that the robot is flush against the wall and ready to mount.

Once the robot is on the wall, the boolean logic that was used to monitor the status of the robot will turn true (is_on_wall = true). Then the code will run through the wall related portion of the code until an action reverses this. When on the wall, the back ultrasonic sensor will repeatedly check for the robot's distance to the ceiling. Once it is within the limit of the vicinity of the ceiling, the condition far_enough_from_floor will be satisfied. After this condition is satisfied, the propellers will rearrange itself in such a way that the robot can begin to safely descend the wall. Once it reaches a certain threshold from the floor, the robot will transition from the wall to floor. Most of the code related to the robot’s actions on the wall is autonomous due to the danger involved in allowing the user to control the robot’s movements on the wall.

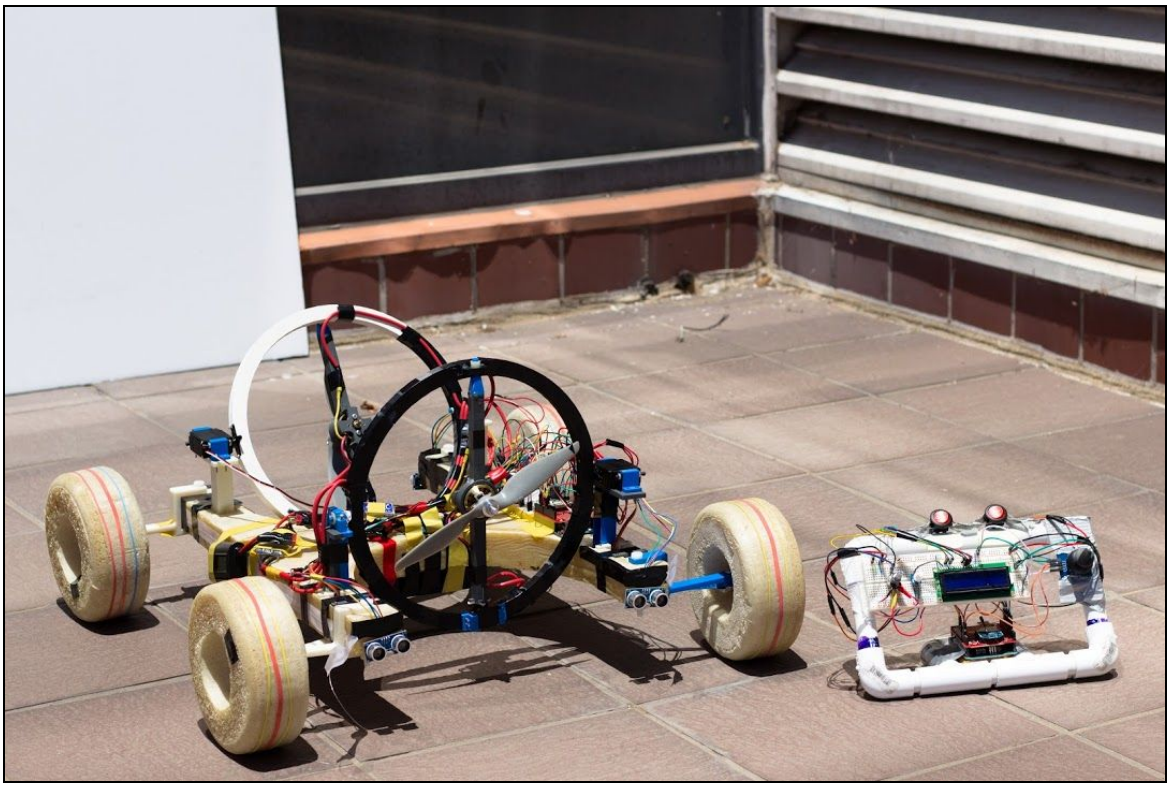


Figure 25 : Final product- robot and remote

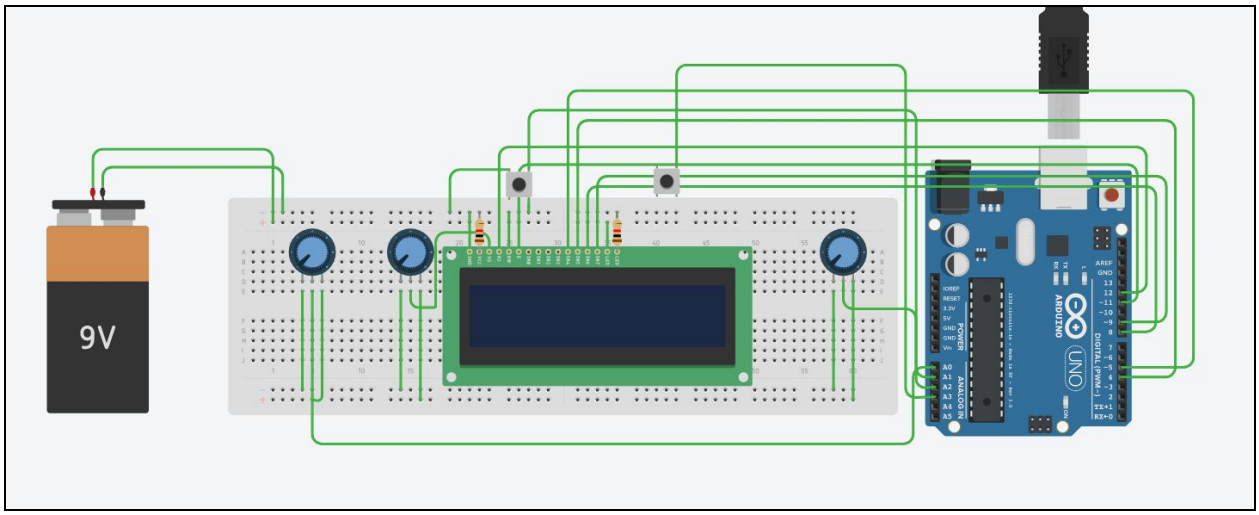


Figure 26: Electronic schematic of remote control

A remote (Figure 19 and 20) was built to control the robot wirelessly using the XBEE modules of the Arduino. The remote consisted of a joystick to control the steering servo angles. In the figure above, the potentiometer located on the right of the breadboard acts as a placeholder for the joystick. An LCD screen was used to display the region in which the robot is in. The LCD acts as a means of providing the user with some guidance as to what controls the user has over. Along with this, the LCD screen will act as a means to prompt the user for when the robot should climb on the wall. Along with an LCD screen, two potentiometers were placed on the remote - a potentiometer to control the brushless motor's RPM and another potentiometer to control the brightness of the LCD screen. Finally, two buttons were placed at the top of the remote. The first button, when on, allows the robot to enter the wall mount setup code. When off, the robot is not allowed to enter this portion of the code and the user is given full control of the remote unless the button status is changed. The second button acts as the wall mount initiation. In order for this button's input to be accepted, the distance between the robot and the wall needs to be at the minimum in which it can climb the wall. Along with this, the robot needs to be parallel, thus suggesting that the ultrasonic sensors must output similar values. When these aforementioned conditions are satisfied, then the second button input can be accepted. If the button is pressed, the robot will proceed to mount the wall. If not, the robot will ask the user, through the LCD screen, whether the user wants the robot to mount the wall.

Communication between the remote and the robot was controlled through XBee Series 1 Module. The XBee was connected to the Arduino through an XBee breakout

board from Sparkfun. With the breakout board, the XBee was supplied 3.3 V from the Arduino. We experienced several issues with the XBee Communication. One of the main problems was the shaking that we would experience when communicating the joystick position from the remote to the servo motor. One of the possible solutions for this was checking that enough power was supplied to the servomotor. After observing that using one 9 V battery to power two servo motors would increase the amount of jittering, we found that using one 9 V supply to control one servo motor would dramatically decrease the jittering. As the servo motor could only handle up to 5 V, the 9 V battery was sent through a linear regulator, 7805C. The 9 V batteries that were used in these preliminary design tests were not used in the final design of the robot.

Another possible solution that we found to limiting the amount that the servomotor would jitter was adding a capacitor in parallel to it. By adding a capacitor, the capacitor would act to smooth out the voltage that was sent to the servomotors. We decided to use a 470 μF capacitor for this application. We observed that using a 1000 μF capacitor would amplify the shakiness of the servomotor. A 100 μF capacitor slightly improved, but did not completely prevent the instability of the servomotor. Therefore, a 470 μF was the optimal choice for limiting the servomotors quivering.

The other solution that we used to reduce the shaking of the servomotor was experimenting with the delays in the code. Due to the conflicting timers in Software Serial and Servo library in Arduino, the servo motors were prone to twitching. As a result, adding a delay to better allow the servo motor to space and differentiate general signals from the XBee with specific signals for the servomotor, limited the twitching of the servo. Therefore, experimenting with the delays in the code significantly affected the magnitude of shaking that occurred in the servomotor. Another phenomenon that we observed is that the brushless motors did not respond to the inputs sent from the XBees unless a delay of 300 milliseconds was added.

One prominent issue that we discovered with the XBee was the sudden end of communication that would occur randomly. We noticed this problem more noticeably when more additions were added to the circuit. Without any added elements to the circuit, the code and communication would be able to run infinitely without any interruption. However, the addition of several circuit elements, such as servo motors, ESCs and ultrasonic sensors caused an increased occurrence of communication ending. We postulate that this might be attributed to the interference from the thinly

insulated wires that surrounded the XBee on the Arduino. Because this issue would disappear and appear in random intervals, it was not a replicable error. Therefore, this issue was never fully resolved. If we were to attempt using the XBee as a form of communication in the future, then we should consider ways to better insulate the XBee from forms of interference.

5.6 Materials

One of the design requirements for this project was the weight constraint in order to ease the motion of the robot. Therefore, the material used for manufacturing the robot had to be strong yet lightweight. Various materials were considered for this purpose:

- Carbon fibre and cardboard honeycomb composite
- Birch plywood
- Polyurethane pour foam

Due to its properties of high strength, low weight, and buoyancy in water, the polyurethane pour foam was used to manufacture the chassis and wheels. In order to manufacture parts using the foam, the foam mixture was poured into a 3-D printed mold and was set to cure for about two minutes. For the final chassis mold, the mold was not 3D printed because it would have had to have been printed in at least six different pieces and the print turnaround time was too long for our timeline. Therefore, we used plywood and sheet metal as seen in Figure X. On top of the plywood but not pictured, we 3D printed the base of the chassis with blocks in locations of where the pillars would go and where the steering servo motors would go.

Once the foam was mixed together, we had about one minute to mix it and pour it into the its respective mold before it started curing. As it cured, it expanded and took on the shape of the mold. The chassis along with its 4 wheels was manufactured this way.

Figure 27: Chassis mold

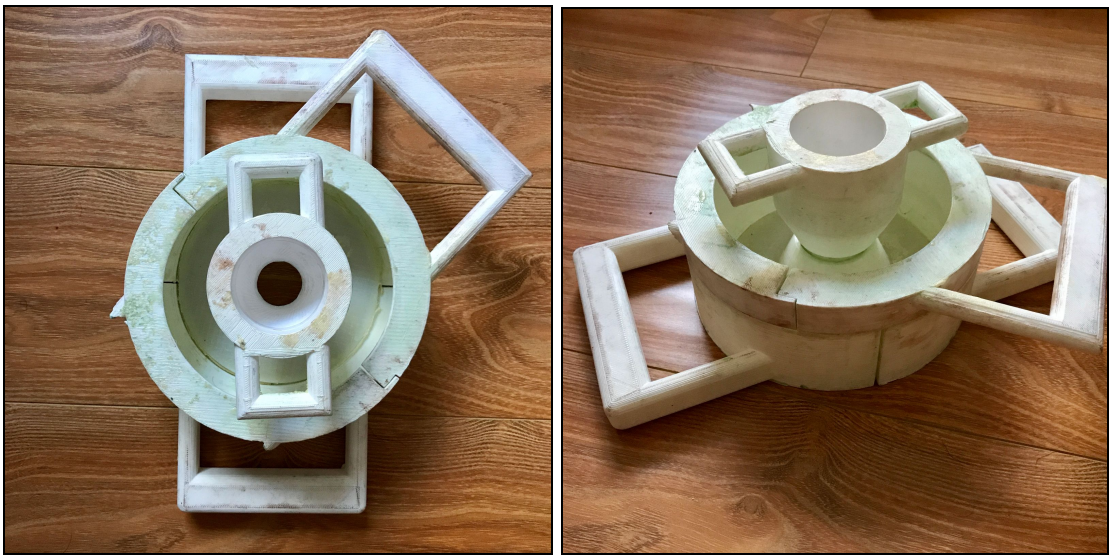


Figure 28: Wheel mold

All the other parts on the chassis (eg. Propeller mounting rings, mounting brackets, pillars etc.) were manufactured using a 3D printer which used Acrylonitrile Butadiene Styrene (ABS) plastic. The 3D printing method was chosen due to a high flexibility in realizable designs and short manufacturing time.

A variety of fasteners were used to join parts together, ranging from screws, nuts, bushings, and circlips. Wherever possible, weight was saved by using nylon screws and nuts. In other places, metal screws and nuts of small sizes (#0 and #1 sizes) were used, for the same reasons.

5.7 Design Reiterations

Wheel Mounting Link

The initial design of the Wheel Mounting Link had a loft which stood on its own on the axle. This was a badly designed part, and we discovered during testing of the robot that the link would shear off at the loft (as pointed in the image below). In order to fix this, the loft was moved to be flush with the flat surface, and the part performed better as expected.

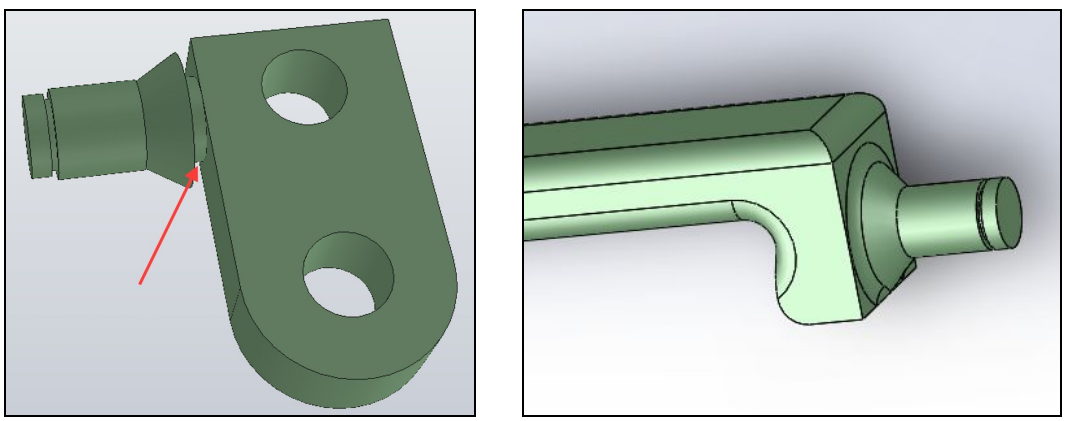


Figure 29 a. Old wheel mounting link b. Revised wheel mounting link

Brushless motor housing

The motor housing went through many iterations, as shown below. The initial designs planned to have the propeller roll motors on the inside of the housing, rotating against a fixed propeller ring (see Figure below). The later designs incorporated a simpler yet effective structure where the roll motor could be mounted onto the propeller ring with minimal support. The initial design for the propeller housing was a circular rod, however, FEA analysis demonstrated this design was not feasible since the rod would bend under a load of 8N. This part was redesigned in order to prevent the rod from displacing more than 1 mm from the initial position. This final design was

lightweight compared to the previous ones, and strong as evidenced by the FEA analysis shown in the figure below.

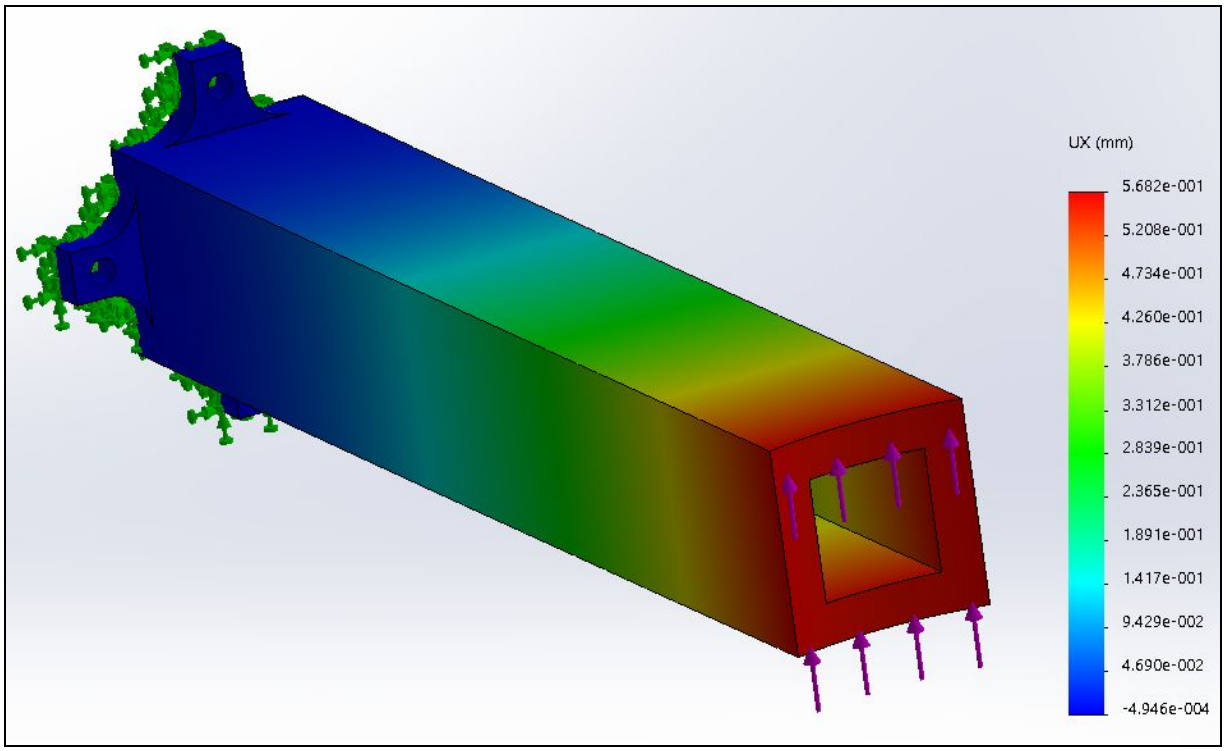


Figure 30: FEA performed using SolidWorks in order to determine the amount of bending due to a 10N force from the propeller. The analysis shows a half-millimeter deflection. This correlated with experimental observations

Chassis

Initial designs of the chassis were X-shaped in order to accommodate the propellers. We later realized that an H-shaped chassis would be easier to manufacture, while still satisfy the requirement of accommodating for the propeller. In addition, preliminary chassis designs were about 0.3 - 0.5 inches thick, whereas the final design was about 1.7 inches thick (since we decided to go with the pour foam as our material). Additionally, the length and width of the chassis changed as we increased our propeller size. The final length and width of the chassis was 20 inches by 16 inches.

Wheels

As shown in a previous section, the shape of the wheels was changed from chamfered cylinder to a normal cylinder. The considerations for wheel design were size, volume of water displaced when submerged, and weight. We also considered manufacturability. The diameter of the wheels was also a parameter that was iterated upon - going from 8 inches to 6.5 inches in height.

6 Summary and Discussion

Two important considerations for the project were that we had to lower the center of mass as much as possible in order to reduce the moment due to gravity in the vertical position. To realize that, we changed the design and placed the wheels at the sides of the chassis instead of under the chassis. We also chose to place the battery under the chassis to help lower the center of mass in addition to saving space on the top of the chassis.

The second consideration was to raise the centre point of the application of thrust in order to maximize the moment arm for the propeller thrust to effectively torque the robot and enable better control. In order to accomplish that the height of the pillars above the chassis was raised.

Being a mechanical system with fast moving parts, there are a few safety concerns. A main safety issue is a malfunctioning battery which in the case of a leak can expose human skin to corrosive chemicals. In a worse case, the malfunction can cause the battery to explode, which can be a major safety hazard. In order to protect from this, we have chosen a high quality battery with a good record of functioning without safety hazards and has been approved by members of the quadcopter and RC plane communities.

The other major safety concern is that there are two propellers on the robot that will be spinning at a very high rate and can be a hazard if accidentally touched. We initially aimed to have the robot climb on the wall, but after initial tests on the wall with the robot attached to fishing wire that was looped through two hooks on the wall, we discovered that this aim was quite dangerous. In one trial, the propeller sliced through the ABS plastic of the propeller ring. In another trial, the housing that hosted the roll servo loosened and separated from the propeller ring. At the time of these accidents, the speed for the brushless motors was half of the maximum RPM, therefore, these tests were deemed as unsafe. Another concern that we faced was the increased shaking of the pitch servo as the RPM increased. Therefore, increasing our RPM would lead to a potentially dangerous situation. Overall, due to the instability of our system, we thought it would be unwise to test the wall climb at the RPM's needed to simulate wall movement of the robot on the wall (which were greater than half the maximum RPM). However, we were successful in establishing a physical contact

between our robot and the wall as a result of the spinning propellers. If we were to repeat this experiment, we would find a way to further decrease the weight of the robot such that the RPM required to initiate the robot's wall movement would be decreased. Along with this, we would use improved servo motors that would help limit the jittering that we observed frequently with our current motors.

As is the case with remotely operated robots, our project also has the potential to be used as a surveillance device which comes with a plethora of ethical concerns. However, that is a distant concern, and until this idea is developed further, the potential for misuse is limited.

7 Conclusions and Recommendations

The initial objective of the project was to recreate the functionalities of the Disney VertiGo robot, with an extra ability: the ability to travel on water. The Surfer robot was designed to travel on irregular surfaces, walls, and water. Out of those three tasks only one was accomplished: irregular surfaces. Through a preliminary test on water with a density of approximately $1.0\,kg/m^3$, it was determined the robot would not be able to float on water. This test was performed without electronics mounted on it and a total weight of 1.6 kg.

The wall mounting was designed to be performed in three steps: 1) wall identification, 2) wall ascent, and 3) wall descent. The wall identification was performed using three ultrasonic sensors; two in the front and one in the back. The two sensors on the front were used to determine whether the robot was entirely perpendicular to the wall. This is performed by comparing the distance readings from both of the sensors; if the difference was greater than 0.8 inches, the robot would steer itself in order to align itself with the wall. Once the robot had touched the wall, it would wait for user input to begin mounting the wall. For the wall mounting, the initial plan was to employ a 6 DOF IMU in order to make the transition smooth. However, due to time constraints, the transition had to be simplified by establishing the initial and final angles and timing the movement of the propellers. These initial and final angles differed on the propellers. The front propeller had an initial angle of zero relatively to the horizontal and the back propeller had an angle of 90 degrees with respect to the

horizontal. During wall mounting, the front and back propellers would then move to an angle of 75 degrees with respect to the vertical. The transition from floor to wall could not be tested due to the fact that the required RPM to maintain the robot on the wall needed to be established. In order to determine the appropriate RPM to sustain the robot on the wall, the robot was placed vertically on the wall while the thrust was increased. This part of the testing proved to be unsafe to test due to jitteriness on the back pitch servo motor. At 50% of the full power, the servo motor would begin to harmonically oscillate up to a level deemed unsafe. This prevented the team from moving forward with testing. The jitter on the motor tried to be resolved with different approaches. These approaches concluded that the problem was not a mechanical problem with the servos or low current on the circuit. Initially, it was believed that the motors had been ruined or stripped by manually moving them. However, it was observed that the jitter would move to a different servo whenever the output pin on the Arduino was changed; therefore, it could be concluded that the jitter was not specific to one motor. Also, it was believed the load current was higher than what the UBEC could supply. In order to test this, all the servo motors were unplugged and the robot was tested; however, this did not stop the jitter either. The problem was narrowed down to a software problem. The proposed solution was to increase the friction on the joints on the propeller ring. A higher friction was achieved by placing Teflon tape in the joints. However, this had the opposite effect: it increased the jitter on the motor.

The testing on the wall had to be abandoned for time constraints. If more time had been available, the RPM on the propeller experiencing the jitter would have been limited to 50% of its full capacity while at the same time allowing for the front propeller to reach full capacity.

The team also experienced problems controlling the robot. These problems could be narrowed to the communication between the Xbees and the steering mechanism linkages. The loss of communication between Xbees seemed to be random and caused by a fluctuation in temperature and current draw. The communication experienced problems while tested outdoors when the temperature was 90 degrees Fahrenheit. The way to solve this problem is through the implementation of shielding.

The linkages on the steering mechanism were designed to move on a plane. However, some of the joints were loose allowing for some movement out of plane. This caused the wheels to attain negative camber. This negative camber, however, was

disproportionate among the wheels and allowed for uncontrolled steering. The problem tried to be resolved by adding friction in the joints of the links using Teflon tape and by decreasing the shaft diameter in order for the circlip to better fit in. Unfortunately, the camber could not be solved causing the robot to steer somewhat randomly when being propelled. The way to solve this in the future is to create parts with better tolerances to accommodate the steering links. Furthermore, this problem was attempted to be solved through washers, but this, too, proved ineffective since the washer thickness could not be controlled precisely.

References

1. Adarsh, Mohamed Kaleemuddin, Dinesh Bose, and Ramachandran. "Performance Comparison of Infrared and Ultrasonic Sensors for Obstacles of Different Materials in Vehicle/ Robot Navigation Applications." *IOP Science*. IOP Publishing Ltd, 2016. Web.
2. AeroMarine Products, Inc. "How to Use AeroMarine Polyurethane Pour-In-Place 2# Foam." (n.d.): n. pag. *AeroMarine Products*. AeroMarine Products Inc. Web.
3. AeroMarine Products, Inc.. "How to Use AeroMarine Polyurethane Pour-In-Place 4# Foam." (n.d.): n. pag. *AeroMarine Products*. AeroMarine Products Inc. Web.
4. AeroMarine Products, Inc. "How to Use AeroMarine Polyurethane Pour-In-Place 8# Foam." (n.d.): n. pag. *AeroMarine Products*. AeroMarine Products, Inc. Web.
5. AeroMarine Products, Inc.. "Liquid Urethane Expandable Pour Foam: Perfect For Boats." *Aeromarine Products Inc.* AeroMarine Products Inc, n.d. Web.
6. AeroMarine Products, Inc.. "Polyurethane Pour Foam For Insulation & Flotation." *Aeromarine Products Inc.* AeroMarine Products Inc, n.d. Web.
7. Alronzo. "Gyroscope." *Gyroscope*. SparkFun, n.d. Web.
8. Arduino. "Arduino - ArduinoBoardUno." *Arduino - ArduinoBoardUno*. Arduino, n.d. Web.
9. Beardsley, Paul, Dr. Roland Siegwart, Michael Arigoni, Michael Bischoff, Silvan Fuhrer, David Krummenacher, Dario Mammolo, and Robert Simpson. "VertiGo – A Wall-Climbing Robot including Ground-Wall Transition." *Disney Research » VertiGo*. Disney, 29 Dec. 2015. Web.
10. Buchmann, Isidpr. "What's the Best Battery?" *Basic to Advanced Battery Information from Battery University*. Isidor Buchmann, 21 Mar. 2017. Web.
11. Carpenter, Pete. "Fundamentals of Brushless RC Motors - Choose Wisely!" *Hooked on RC Airplanes*. Pete Carpenter, n.d. Web. 24 Mar. 2017.
12. Carpenter, Pete. "RC Airplane Propeller Size Guide." *RC Airplane World - Beginner's RC Flying Guide*. Pete Carpenter, n.d. Web. 24 Mar. 2017.
13. Călin, Dragoș-George. "Types of Sensors for Target Detection and Tracking." *Into Robotics*. Roboti, 20 Nov. 2013. Web.
14. Dee, Jordan. "Pulse-width Modulation." *Pulse-width Modulation*. SparkFun, n.d. Web.
15. Digi. "The Major Differences in the XBee 802.15.4 vs. the XBee ZB (S2) and (S2C)."

Kb. Digi International Inc., n.d. Web.

16. Earl, Bill. "About Servos and Feedback." *Adafruit Products*. Adafruit, n.d. Web.
17. Frenzel Lou Frenzel | Electronic Design, Lou. "What's The Difference Between IEEE 802.15.4 And ZigBee Wireless?" *What's The Difference Between... Content from Electronic Design*. Penton, 22 Mar. 2013. Web.
18. Goodrich, Ryan. "Accelerometers: What They Are & How They Work." *LiveScience*. Purch, 01 Oct. 2013. Web.
19. Gorilla Glue, Inc. "Gorilla Glue | Gorilla Glue." *Original Polyurethane Formula | Waterproof Glue*. Gorilla Glue, Inc, n.d. Web.
20. Jayant. "Arduino vs Raspberry Pi - Difference between the Two." *Circuit Digest*. Circuit Digest, n.d. Web.
21. Mahoney, Doug. "What's So Great About Brushless Motors?" *Popular Mechanics*. Hearst Communications, 30 Jan. 2015. Web. 24 Mar. 2017.
22. PRBA. "Types of Batteries." *PRBA Types of Batteries Comments*. Portable Rechargeable Battery Association, n.d. Web.
23. Raspberry Pi. "Raspberry Pi FAQs - Frequently Asked Questions." *Raspberry Pi*. Raspberry Pi, n.d. Web.
24. Reed, Frances. "How Do Servo Motors Work." *How It Works*. Jameco Electronics, n.d. Web.
25. REXCO. "Application Guide PARTALL® Paste #2." (n.d.): n. pag. *Rexco Mold Care*. REXCO. Web.
26. Schneider, Brian. "A Guide to Understanding LiPo Batteries." *Roger's Hobby Center*. Roger's Hobby Center, n.d. Web.
27. Seattle Robotics Society. "Introduction to Servo Motors." *Introduction to Servo Motors*. Northwestern University Mechatronics Design Laboratory, n.d. Web.
28. SparkFun. "XBee Buying Guide." *XBee Buying Guide - SparkFun Electronics*. SparkFun Electronics, n.d. Web.
29. Starlino. "A Guide To Using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications." *Starlino Electronics*. Starlino Electronics, 29 Dec. 2009. Web.

Appendix

The following is a code that will estimate the amount of torque we need for a servomotor. These values were based on our older model - spherical wheels with an inverted Y-Frame. No real values were put into these calculations as the dimensions of the robot were uncertain at the time. New torque calculations, resembling the logic in this code, will be made for our new model in the future.

```
% universal variables
density_pla = 4;
density_foam = 4;

%set up moment of inertia formulas
% vert - vertical component of y-frame
% hor - horizontal component of y-frame
% yball - rods connecting y-frame to ball
% sphere - spherical ball
% vert2 - 2 vertical rods underneath the y-frame

% moment of inertia for sphere (solid)
density_sphere = density_foam;
radius_sphere = 1;
volume_sphere = (4/3)*pi*(radius_sphere^3);
mass_sphere = density_sphere*volume_sphere;
inertia_sphere = (2/5)*mass_sphere*(radius_sphere^2);

inertia_sum = inertia_sphere;

% moment of inertia for vert (cylinder)
density_vert = density_pla;
radius_vert = 1;
height_vert = 1;
volume_vert = pi*(radius_vert^2)*height_vert;
mass_vert = density_vert*volume_vert;
inertia_vert = (1/2)*mass_vert*(radius_vert^2);

inertia_sum = inertia_sum + inertia_vert;
```

```
%moment of inertia for hor (cylinder)
density_hor = density_pla;
radius_hor = 1;
length_hor = 1;
volume_hor = pi*(radius_hor^2)*length_hor;
mass_hor = density_hor*volume_hor;
inertia_hor                                                                =
((1/4)*mass_hor*(radius_hor^2))+((1/12)*mass_hor*(length_hor^2));

inertia_sum = inertia_sum+inertia_hor;

%moment of inertia for yball (cylinder with empty cylinder of length =
%sphere diameter cut out)
density_yball = density_pla;
radius_yball = radius_hor;
length_yball = length_hor;
%inertia for complete cylinder
volume_yball_outer = pi*(radius_yball^2)*length_yball;
mass_yball_outer = density_yball*volume_yball_outer;
inertia_yball_outer                                                        =
((1/4)*mass_yball_outer*(radius_yball^2))+((1/12)*mass_yball_outer*(length_y
ball^2));
%inertia for cylinder of length = diameter of sphere
volume_yball_inner = pi*(radius_yball^2)*(radius_sphere*2);
mass_yball_inner = density_yball*volume_yball_inner;
inertia_yball_inner                                                        =
((1/4)*mass_yball_inner*(radius_yball^2))+((1/12)*mass_yball_inner*((radius_
sphere*2)^2));
%final inertia
inertia_yball = inertia_yball_outer - inertia_yball_inner;

inertia_sum = inertia_sum + inertia_yball;

%moment of inertia for vert2
density_vert2 = density_pla;
radius_vert2 = 1;
length_vert2 = 1;
volume_vert2 = pi*(radius_vert2^2)*length_vert2;
```

```
mass_vert2 = volume_vert2*density_vert2;
%inertia (approximate distance from center axis of cylinder to center of
%sphere as radius of sphere)
inertia_vert2              =              (1/2)*mass_vert2*(radius_vert2^2)+
(mass_vert2*(radius_sphere^2));

inertia_sum = inertia_sum + inertia_vert2;

%approximate velocity profile (as linear for now?)
%time values
t_array = [0 1 2 3 4];
%corresponding angle values
theta = [0 20 30 35 50];
theta_approx = polyfit(t_array,theta,2);
angular_vel = polyder(theta_approx);
angular_accel = polyder(angular_vel);


%set up torque equations

%friction forces? should we leave out for now?

torque = inertia_sum * angular_accel;
```

The following is a code used to acquire the data in Table 1. This gives the optimized length of the links for our steering design. This also produces the plots of the angles as a function of the input angle.

```
clear all
close all
clc

%Setting fixed parameters (units:mm)
r2 = 12.7*(1/1000) ;
%r3 = 111.7*(1/1000);
```

```
r3=96.28*(1/1000)
r4 = 15*(1/1000);
r1x = 94.85*(1/1000);
%r1y=17.84*(1/1000);
r1y=15*(1/1000);
%setting up storage
rr = 100;
theta2_maximum = pi;
theta2_minimum = 0;
theta2 = linspace(theta2_minimum ,theta2_maximum,rr);
t3 = zeros(1,rr);
t4 = zeros(1,rr);
%initial guesses
t3_1 = 200*pi/180;
t4_1 = 250*pi/180;

%setting for loop
for jj = 1:rr;

%Set dx and dy to a randomly large number
dx1 = 100;
dx2 = 100;
D = [dx1; dx2];

%Set up variables for iterations
n = 0;

while norm(D,1) > 10^(-6);
    n = n+1;
    %Evaluate f1, f2
    f1=r2*cos(theta2(jj))+r3*cos(t3_1)+r4*cos(t4_1)+r1x
    f2 = r2*sin(theta2(jj)) + r3*sin(t3_1) + r4*sin(t4_1)+r1y;
    F = [f1;
        f2;];

   %Evaluate Jacobian
   J11 = -r3*sin(t3_1);
   J12 = -r4*sin(t4_1);
   J21 = r3*cos(t3_1);
```

```
    J22 = r4*cos(t4_1);
      J = [J11 J12;
         J21 J22;];

    %return dx, dy
    D = -J\F;
    dx1 = D(1);
    dx2 = D(2);

    %update x_1, y_1
    t3_1 = t3_1 + dx1;
    t4_1 = t4_1 + dx2;

    if n > 100
        disp ('loop not converging')
        break;
    end
end

%Extract x_1 and y_1 for solution / convert to radians
t3(jj) = t3_1;
t4(jj) = t4_1;

end

figure(1)
plot(theta2*(180/pi), t3*(180/pi))
xlabel('Theta 2 ( degree)')
ylabel('Theta 3 ( degree)')

figure(2)
plot(theta2*(180/pi),t4*(180/pi))
xlabel( 'theta 2 (Degrees) ')
ylabel('theta 4 ( degrees)')

maxM=max(t4*(180/pi))
minM=min(t4*(180/pi))

Mag=maxM-minM
```

```
ZZZ=[r2 r3 r4 r1x r1y Mag]'
```

The following is the final iteration of the code. In the code, the portion detailing the wall climbing portion of the code was blocked out due to potential hazards of the robot in this mode. Because of these hazards, we tested the wall climb portion using the same code as the one we used for the robot when it was on the floor. With this, we can place the robot on the wall and then begin to increase the RPM of the brushless motors slowly to ensure nothing dangerous occurs. In the future, given increased safety precautions, we can test the robot without these measures.

```
// declare global variables


// ADD A CONDITION WHERE THE DISTANCE TO THE WALL MUST BE LESS THAN THE THRESHOLD
AT LEAST TWICE FOR SAFETY!

// include libraries
#include <Servo.h>
#include <SoftwareSerial.h>

SoftwareSerial xbee(2,3);
// define pin numbers and variables for ultrasonic sensor
extern const int ULTRASONIC_TRIG_FRONT_LEFT;
extern const int ULTRASONIC_ECHO_FRONT_LEFT;
extern const int ULTRASONIC_TRIG_FRONT_RIGHT;
extern const int ULTRASONIC_ECHO_FRONT_RIGHT;
extern const int ULTRASONIC_TRIG_BACK;
extern const int ULTRASONIC_ECHO_BACK;
extern const float ULTRASONIC_DELAY_MS;
extern const float ULTRASONIC_DISTANCE_ERROR_CLOSE;
extern const float ULTRASONIC_DIFFERENCE_ERROR;
extern const float WALL_MOUNT_DISTANCE;
extern float previous_distance;
extern int angle_propeller_back;
extern int angle_propeller_front;

extern const float FLOOR_MOUNT_DISTANCE;

// declare variables that set up ranges for robot's distance to the wall
const float FAR_WALL_THRESHOLD = 43;

// declare variables that  up ranges for robot's distance to floor
const float FAR_FLOOR_THRESHOLD = 3;
const float NEAR_FLOOR_THRESHOLD = 0;
int increment_tries = 0;

bool is_on_wall = false;
```

```
bool enable_wall_mount_setup = false;
boolean old_enable_wall_mount_setup = false;
bool enable_wall_mount = false;
bool far_enough_from_floor = false;
bool undergoing_wall_mount = false;
bool set_up_region_4 = false;

void setup() {
  ultrasonic_setup();
  xbee.begin(9600);
  Serial.begin(9600);
}

void loop() {
  float distance_to_obstacle;

  // this is the MAIN IF/ELSE LOOP that considers two conditions - on the floor or
not on the floor
  // when robot is on the floor condition is satisfied, this part of the loop will
run
  // IF ROBOT IS NOT ON WALL
  if (!is_on_wall) {
    old_enable_wall_mount_setup = enable_wall_mount_setup;
    enable_wall_mount_setup = check_button_status('s');

    // this is the SECOND MAIN IF/ELSE LOOP that considers two conditions - whether
the button is on or off (on - robot can begin mounting wall process / off - robot
should not begin mounting wall process)
    // IF BUTTON IS NOT PRESSED
    if (!enable_wall_mount_setup) {     // if button is not on the wall and button
has not been pressed, everywhere is region four
      if (set_up_region_4 == false) {
        orient_propeller_angle('P', 90, 0);
        orient_propeller_angle ('R', 170, 163);
        lcd_message ("LB4");
        set_up_region_4 = true;
      }
      if (set_up_region_4 == true) {
        region_four();                    // region_four() = orient_steering +
orient_speed + vertical_propeller
      }
    }
    // IF BUTTON IS PRESSED
    else {
      set_up_region_4 = false;
      distance_to_obstacle = get_raw_min_distance(ULTRASONIC_TRIG_FRONT_LEFT,
ULTRASONIC_ECHO_FRONT_LEFT, ULTRASONIC_TRIG_FRONT_RIGHT,
ULTRASONIC_ECHO_FRONT_RIGHT);
      if (distance_to_obstacle > FAR_WALL_THRESHOLD) {        // if button is
pressed, we allow only steering inputs from user when robot is beyond is region 2
        lcd_message("LB3");
        region_three();                                       //
```

```
set_min_speed_propeller and orient_steering
      }
      else if (distance_to_obstacle <= FAR_WALL_THRESHOLD ) {
        enable_wall_mount = check_button_status('m');
        if (enable_wall_mount == false) {
          region_two();       // set_min_speed_propeller
          lcd_message("LB2");
          if (get_diff_distance(ULTRASONIC_TRIG_FRONT_LEFT,
ULTRASONIC_ECHO_FRONT_LEFT,
ULTRASONIC_TRIG_FRONT_RIGHT,ULTRASONIC_ECHO_FRONT_RIGHT) <= 1.5 &&
            distance_to_obstacle <= WALL_MOUNT_DISTANCE +
ULTRASONIC_DISTANCE_ERROR_CLOSE && distance_to_obstacle >= WALL_MOUNT_DISTANCE -
ULTRASONIC_DISTANCE_ERROR_CLOSE) {
            lcd_message("LMQ");
          }
        }
        else if (enable_wall_mount == true) {
          if (get_diff_distance(ULTRASONIC_TRIG_FRONT_LEFT,
ULTRASONIC_ECHO_FRONT_LEFT,
ULTRASONIC_TRIG_FRONT_RIGHT,ULTRASONIC_ECHO_FRONT_RIGHT) <= 1.5 &&
            distance_to_obstacle <= WALL_MOUNT_DISTANCE +
ULTRASONIC_DISTANCE_ERROR_CLOSE && distance_to_obstacle >= WALL_MOUNT_DISTANCE -
ULTRASONIC_DISTANCE_ERROR_CLOSE || undergoing_wall_mount == true){
            undergoing_wall_mount = true;
            lcd_message ("LMS");
            region_one(increment_tries);
            increment_tries += 1;
            if (increment_tries == 2) {
              is_on_wall = true;
            }
          }
          else {
            region_two();       // set_min_speed_propeller
            lcd_message("LB2");
          }
        }
      }
    }
  }

  // when robot is on the wall
  else if (is_on_wall == true) {
    lcd_message ("LWU");



//    if (far_enough_from_floor == false) {
//      orient_propeller_angle ('R', 167, 162);
//      move_up_wall();
//      move_up_wall_speed();
//      lcd_message("LWU");
//      int distance_to_obstacle_front =
```

```
get_raw_min_distance(ULTRASONIC_TRIG_FRONT_LEFT, ULTRASONIC_ECHO_FRONT_LEFT,
ULTRASONIC_TRIG_FRONT_RIGHT, ULTRASONIC_ECHO_FRONT_RIGHT);
//    }
//    else if (far_enough_from_floor == true) {
//      orient_propeller_angle ('R', 167, 162);
//      move_up_wall();
//      move_down_wall_speed();
//      lcd_message ("LWD");
//    }

//    distance_to_obstacle = get_distance(ULTRASONIC_TRIG_BACK,
ULTRASONIC_ECHO_BACK);
//    if (far_enough_from_floor == false){
//      lcd_message("LWU");
//      move_up_wall_speed();
//      roll_straight();
//      move_up_wall();
//      if (distance_to_obstacle < 2) {
//        far_enough_from_floor = true;
//      }
//    }
//    else {
//      move_down_wall_speed();
//      roll_straight();
//      move_down_wall();
//      if (distance_to_obstacle >= FLOOR_MOUNT_DISTANCE -
ULTRASONIC_DISTANCE_ERROR_CLOSE && distance_to_obstacle <= FLOOR_MOUNT_DISTANCE +
ULTRASONIC_DISTANCE_ERROR_CLOSE){
//        wall_to_floor();
//        is_on_wall = false;
//        far_enough_from_floor = false;
//      }
//    }
  }
}


// XBee and Steering Control
// Functions defined include:
// void steering_setup()      (setup the servomotors to the correct pins)
// float orient_steering()    (use the joystick commands sent from the XBee to
orient direction of servomotors)
//                            (parallel steering - servomotors turn to result in
same output angle)

// declare pins
const int SERVO_LEFT_PIN = 4;
const int SERVO_RIGHT_PIN = 5;


// declare classes
Servo servo_left;
```

```
Servo servo_right;
// declare variables

void detach_all_steering() {
  servo_left.detach();
  servo_right.detach();

}

void attach_all_steering() {
  servo_left.attach(SERVO_LEFT_PIN);
  servo_right.attach(SERVO_RIGHT_PIN);
}

void orient_steering(){
  detach_all();
  int joystick_input = get_joystick();
  int servo_angle_left = map(joystick_input, 0,1023,1000, 1333);  // the joystick
takes analog values from 0 to 1023. writeMicroseconds command takes 1000 to 2000 as
input. use the map function to assign our joystick input to our new range
  int servo_angle_right = map(joystick_input, 0, 1023, 1333, 1080);
  attach_all();
  servo_left.writeMicroseconds(servo_angle_left);
  servo_right.writeMicroseconds(servo_angle_right);
  delay(200);
  detach_all();
}

// 75, 15 for wall
void region_four() {
  orient_steering();
  orient_speed('F');
}

void region_three() {
  set_constant_speed("Minimum");
  orient_propeller_angle ('P', 90, 0);
  orient_propeller_angle ('R', 170, 163);
  orient_steering();
}

void region_two() {
  set_constant_speed("Minimum");
  orient_propeller_angle ('P', 90, 0);
  orient_propeller_angle ('R', 170, 163);
}

void region_one(int iteration) {
  floor_to_wall_speed();
  orient_propeller_angle ('R', 170, 163);
  floor_to_wall(iteration);
}
```

```
void wall_climb() {
  orient_propeller_angle('P', 75, 15);
  orient_propeller_angle ('R', 170, 163);
  orient_speed('W');
}

// PROPELLER FUNCTIONS
// declare pin numbers for the propellers
const int BRUSHLESS_FRONT_PIN = 6;
const int BRUSHLESS_BACK_PIN = 7;
const int PROPELLER_PITCH_FRONT_PIN = 10;
const int PROPELLER_PITCH_BACK_PIN = 11;
const int PROPELLER_ROLL_FRONT_PIN = 8;
const int PROPELLER_ROLL_BACK_PIN = 9;

// declare variables
const int VERTICAL_PITCH_FRONT = 90;   // note that these are dummy variables
const int VERTICAL_PITCH_BACK = 0;    // vertical = 90 / perpendicular to ground
const int HORIZONTAL_PITCH_FRONT = 180;
const int HORIZONTAL_PITCH_BACK = 1000;

// brushless motor duty cycle constants
const int MIN_PROPELLER_SPEED = 1100;
const int FRONT_TRANSITION_SPEED = 1400;
const int BACK_TRANSITION_SPEED = 1400;
const int DOWN_WALL_SPEED = 1000;
const int UP_WALL_SPEED = 1000;
const int PROP_TRANSITION_DELAY = 300;
const int PROP_TRANSITION_INCREMENTS = 5;

const float FLOOR_MOUNT_DISTANCE = 1;

// declare classes
Servo brushless_front;
Servo brushless_back;
Servo propeller_pitch_front;
Servo propeller_pitch_back;
Servo propeller_roll_front;
Servo propeller_roll_back;


void detach_all_propeller() {
  brushless_front.detach();
  brushless_back.detach();
  propeller_pitch_front.detach();
  propeller_pitch_back.detach();
  propeller_roll_front.detach();
  propeller_roll_back.detach();
}

void attach_all_propeller() {
```

```
  brushless_front.attach(BRUSHLESS_FRONT_PIN);
  brushless_back.attach(BRUSHLESS_BACK_PIN);
  propeller_pitch_front.attach(PROPELLER_PITCH_FRONT_PIN);
  propeller_pitch_back.attach(PROPELLER_PITCH_BACK_PIN);
  propeller_roll_front.attach(PROPELLER_ROLL_FRONT_PIN);
  propeller_roll_back.attach(PROPELLER_ROLL_BACK_PIN);
}

void attach_all () {
  attach_all_propeller();
  attach_all_steering();
}

void detach_all () {
  detach_all_propeller();
  detach_all_steering();
}

void orient_propeller_angle (char type_of_propeller, int front_propeller, int
back_propeller) {
  detach_all();
  attach_all();
  if (type_of_propeller == 'R') {       // ROLL
    propeller_roll_front.write(front_propeller);
    propeller_roll_back.write(back_propeller);
  }
  else if (type_of_propeller == 'P') {  // PITCH
    propeller_pitch_front.write(front_propeller);
    propeller_pitch_back.write(back_propeller);
  }
  detach_all();
  pinMode(PROPELLER_PITCH_FRONT_PIN,INPUT);
  pinMode(PROPELLER_PITCH_BACK_PIN,INPUT);
  pinMode(PROPELLER_ROLL_FRONT_PIN, INPUT);
  pinMode(PROPELLER_ROLL_BACK_PIN, INPUT);
}

void orient_speed(char floor_or_wall) {
  if (floor_or_wall == 'F') {
    detach_all();
    int pot_input = get_potentiometer();
    int brushless_speed_front = map(pot_input,0,1023,1000,1150);
    int brushless_speed_back = map(pot_input, 0, 1023, 1000, 1150);
    attach_all();
    brushless_front.writeMicroseconds(brushless_speed_front);
    brushless_back.writeMicroseconds(brushless_speed_back);
  }
  if (floor_or_wall == 'W') {
    detach_all();
    int pot_input = get_potentiometer();
    int brushless_speed_front = map(pot_input, 0, 1023, 1000, 1600);
    int brushless_speed_back = map(pot_input, 0, 1023, 1000, 1300);
```

```
    attach_all();
    brushless_front.writeMicroseconds(brushless_speed_front);
    brushless_back.writeMicroseconds(brushless_speed_back);
  }
  delay(300);
  detach_all();
}

void set_constant_speed (String type) {
  if (type == "Minimum") {
    detach_all_steering();
    detach_all_propeller();
    int brushless_min_speed = get_speed('M');
    attach_all_steering();
    attach_all_propeller();
    brushless_front.writeMicroseconds(brushless_min_speed);
    brushless_back.writeMicroseconds(brushless_min_speed);
    delay(300);
    detach_all_steering();
    detach_all_propeller();
  }
}

void floor_to_wall_speed() {
  detach_all_steering();
  detach_all_propeller();
  int brushless_speed = get_speed('F');
  attach_all_steering();
  attach_all_propeller();
  brushless_front.writeMicroseconds(brushless_speed);
  brushless_back.writeMicroseconds(brushless_speed);
  delay(300);
  detach_all_steering();
  detach_all_propeller();
}

void floor_to_wall(int iteration) {
// WITHOUT IMU
  attach_all_propeller();
  attach_all_steering();
  if (iteration == 0) {
    propeller_pitch_front.write(180);
    propeller_pitch_back.write(0);
  }
  else if (iteration == 1) {
    propeller_pitch_front.write(45);
    propeller_pitch_back.write(45);
  }
  delay(300);
  detach_all_propeller();
  detach_all_steering();
}
```

```
void move_down_wall_speed() {
  detach_all_steering();
  detach_all_propeller();
  int brushless_speed = get_speed('D');
  attach_all_steering();
  attach_all_propeller();
  brushless_front.writeMicroseconds(brushless_speed);
  brushless_back.writeMicroseconds(brushless_speed);
  delay(300);
  detach_all_steering();
  detach_all_propeller();
}

void move_down_wall() {
  attach_all_steering();
  attach_all_propeller();
  propeller_pitch_front.write(30);
  propeller_pitch_back.write(45);
  detach_all_steering();
  detach_all_propeller();
}

void wall_to_floor() {
  int variable;
  brushless_front.writeMicroseconds(variable);
  brushless_back.writeMicroseconds(variable);
  int angle_propeller_front = 0;      //initial position of front propeller on
floor before proceeding to mount on wall
  propeller_pitch_front.write(angle_propeller_front);
  int angle_propeller_back = 45;           // initial position of back propeller on
floor
  for(int i = 0; i < PROP_TRANSITION_INCREMENTS; i++) {
    propeller_pitch_back.write(angle_propeller_back);
    delay (PROP_TRANSITION_DELAY);
    angle_propeller_back += ((180-45) / PROP_TRANSITION_INCREMENTS);
  }
}

void move_up_wall_speed() {
  detach_all_steering();
  detach_all_propeller();
  int brushless_speed = get_speed('C');
  attach_all_steering();
  attach_all_propeller();
  brushless_front.writeMicroseconds(brushless_speed);
  brushless_back.writeMicroseconds(brushless_speed);
  delay(300);
  detach_all_steering();
  detach_all_propeller();
}
```

```
void move_up_wall() {
  attach_all_steering();
  attach_all_propeller();
  propeller_pitch_front.write(45);
  propeller_pitch_back.write(45);
  detach_all_steering();
  detach_all_propeller();
}


//Ultrasonic Sensor HC SR04
// Functions defined include:
//    void ultrasonic_setup   (used to setup the ultrasonic echo and trig pins)
//    float get_raw_distance  (used to send ultrasonic sound wave)
//                            (provides an unfiltered value for distance in inches)
//    float get_distance      (used to get distance from the robot on wall or
floor)
//                            (depends on whether !on_wall is true or false)

// define pin numbers for ultrasonic sensor
const int ULTRASONIC_TRIG_FRONT_LEFT = A0;
const int ULTRASONIC_ECHO_FRONT_LEFT = A1;
const int ULTRASONIC_TRIG_FRONT_RIGHT = A2;
const int ULTRASONIC_ECHO_FRONT_RIGHT = A3;
const int ULTRASONIC_TRIG_BACK = A4;
const int ULTRASONIC_ECHO_BACK = A5;

// Define constant values used throughout ultrasonic related operations
const float ULTRASONIC_DELAY_MS = 100;   // Delay period between each iteration of
get_distance function. MUST ACCOUNT FOR SKELETON LOOP TIME
const float MAX_SPEED = 2;                // Maximum operable speed of our robot in
inches per sec
const int MAX_TRIES_ULTRASONIC = 5;      // number of times ultrasonic sensor will
try to measure distance upon receiving invalid measure
const float ULTRASONIC_DISTANCE_ERROR_CLOSE = 0.8;
const float WALL_MOUNT_DISTANCE = 2.5;
const float ULTRASONIC_DIFFERENCE_ERROR = 1;

// measure the maximum distance robot can travel using MAX_SPEED
// MAX_SPEED will be measured by us (we can run robot at maximum operable speeds
and measure speed)
// MAX_SPEED must be in measured in in/s!
const float MAX_DISTANCE_TRAVELED = MAX_SPEED * (ULTRASONIC_DELAY_MS / 1000);

// define variables for use throughout the program (that need to be stored with
every iteration) (useful for calibrating)
float previous_distance;      //current distance to obstacle

void ultrasonic_setup(){
  pinMode (ULTRASONIC_TRIG_FRONT_LEFT,OUTPUT);   //define trigPin as an output as
it will send ultrasound waves
  pinMode (ULTRASONIC_TRIG_FRONT_RIGHT, OUTPUT);
```

```
  pinMode(ULTRASONIC_TRIG_BACK,OUTPUT);
  pinMode (ULTRASONIC_ECHO_FRONT_LEFT,INPUT);    //define echoPin as an input as it
will receive ultrasound waves from trigPin
  pinMode (ULTRASONIC_ECHO_FRONT_RIGHT,INPUT);
  pinMode(ULTRASONIC_ECHO_BACK,INPUT);
}


// NOTE : We decided to have 2 ultrasonic sensors in the front of the robot. I have
not had the time to add that to this code yet
// but I will get that done tonight after Mother's Day!
float get_raw_distance(int ultrasonic_pin_trig, int ultrasonic_pin_echo){
  float distance;
  // clear trigPin
  digitalWrite (ultrasonic_pin_trig,LOW);
  delayMicroseconds(2);

  //send ultrasound wave
  digitalWrite (ultrasonic_pin_trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(ultrasonic_pin_trig,LOW);

  //read the echopin. value returned measures time of TOTAL sound wave travel (from
being transmitted to being received)
  float duration = pulseIn (ultrasonic_pin_echo,HIGH);

  //convert values to distance; converting seconds to cm using speed of sound (340
m/s)
  //divide by two to get the time traveled from hitting obstacle to being received
  float distance_cm = (duration*0.034)/2;

  //convert to inches
  distance = distance_cm * 0.3937;
  return distance;
}

float get_distance(int ultrasonic_pin_trig, int ultrasonic_pin_echo) {
  float distance;

  // if distance traveled measured by Ultrasonic is greater than what our robot can
physically achieve, value is invalid
  // for loop will return -1 and code will re-run until valid measure is given
  for (int attempts = 1; attempts <= MAX_TRIES_ULTRASONIC; attempts++) {
    distance = get_raw_distance(ultrasonic_pin_trig, ultrasonic_pin_echo);

    // check if distance traveled measured by ultrasonic sensor is achievable by
our robot
    float distance_traveled = distance - previous_distance;

    //convert to positive number to account for robot traveling away or towards the
wall
    if (distance_traveled < 0){
```

```
      distance_traveled = distance_traveled * -1;
    }

    // if value measured by ultrasonic sensor is valid, the for loop will exit.
    if (distance_traveled < MAX_DISTANCE_TRAVELED * attempts) {
      previous_distance = distance;
      break;
    }
    // if distance is not valid then the loop will run again. if attempts =
MAX_TRIES_ULTRASONIC, then the loop will exit and print message indicating failure.
    else {
      distance = -1;
      delay(ULTRASONIC_DELAY_MS);
    }
  }

  if (distance == -1){
//    Serial.println("Invalid Distance Measurement");
    previous_distance = get_raw_min_distance(ULTRASONIC_TRIG_FRONT_LEFT,
ULTRASONIC_ECHO_FRONT_LEFT, ULTRASONIC_TRIG_FRONT_RIGHT,
ULTRASONIC_ECHO_FRONT_RIGHT);
  }

//  Serial.print("Distance:");
//  Serial.println(distance);
  return distance;
}

float get_min_distance(int ultrasonic_pin_trig_left, int ultrasonic_pin_echo_left,
int ultrasonic_pin_trig_right, int ultrasonic_pin_echo_right){
  float distance_to_obstacle_left = get_distance(ultrasonic_pin_trig_left,
ultrasonic_pin_echo_left);
  float distance_to_obstacle_right = get_distance(ultrasonic_pin_trig_right,
ultrasonic_pin_echo_right);
  float distance_to_obstacle = min(distance_to_obstacle_left,
distance_to_obstacle_right);
//  Serial.print( "Minimum Distance to Obstacle: ");
//  Serial.println(distance_to_obstacle);
  return distance_to_obstacle;
}

float get_raw_min_distance (int ultrasonic_pin_trig_left, int
ultrasonic_pin_echo_left, int ultrasonic_pin_trig_right, int
ultrasonic_pin_echo_right) {
  float distance_to_obstacle_left = get_raw_distance(ultrasonic_pin_trig_left,
ultrasonic_pin_echo_left);
//  Serial.println(distance_to_obstacle_left);
  float distance_to_obstacle_right = get_raw_distance(ultrasonic_pin_trig_right,
ultrasonic_pin_echo_right);
//  Serial.println(distance_to_obstacle_right);
  float distance = min(distance_to_obstacle_left, distance_to_obstacle_right);
//  Serial.print( "Minimum Distance to Obstacle: ");
```

```
//  Serial.println(distance_to_obstacle);
//  Serial.print("End of Get Raw Min Distance: ");
//  Serial.println(distance);
  return distance;
}

float get_diff_distance (int ultrasonic_pin_trig_left, int
ultrasonic_pin_echo_left, int ultrasonic_pin_trig_right, int
ultrasonic_pin_echo_right){
  float distance_to_obstacle_left = get_raw_distance(ultrasonic_pin_trig_left,
ultrasonic_pin_echo_left);
  float distance_to_obstacle_right = get_raw_distance(ultrasonic_pin_trig_right,
ultrasonic_pin_echo_right);
  float dist_diff = distance_to_obstacle_left - distance_to_obstacle_right;

  if (dist_diff < 0) {
    dist_diff = dist_diff * -1;
  }
  return dist_diff;
}

bool is_speed_zero(){
  return false;
}

// XBee Communication
const int DELAY_XBEE = 50;

int get_joystick() {
  xbee.println('j');
  delay(DELAY_XBEE);
  while (true) {
    if (xbee.available()){
      String raw_input = xbee.readStringUntil('\n');
      char id_char = raw_input.charAt(0);                 // look at the first
character in the string
      if (id_char == 'j') {                                // if the first
character is a j, we know that it came from the joystick
        int joystick_input = raw_input.substring(1).toInt();      // example: j526
-> raw_input.substring(1).toInt() converts string to int and discards j. end result
is -> joystick_input = 526
        return joystick_input;
      }
    }
  }
}

int get_potentiometer() {
  xbee.println('p');
  delay(DELAY_XBEE);
  while (true) {
    if (xbee.available()){
```

```
      String raw_input = xbee.readStringUntil('\n');
      char id_char = raw_input.charAt(0);                 // look at the first
character in the string
      if (id_char == 'p') {                                // if the first
character is a j, we know that it came from the joystick
        int pot_input = raw_input.substring(1).toInt();      // example: j526 ->
raw_input.substring(1).toInt() converts string to int and discards j. end result is
-> joystick_input = 526
        return pot_input;
      }
    }
  }
}

int get_speed(char identifier) {
  xbee.println('V' + String(identifier));
  //Serial.println("get speed called xbee");
  delay(DELAY_XBEE);
  while (true) {
    if (xbee.available()) {
      String raw_input = xbee.readStringUntil('\n');
      char id_char = raw_input.charAt(0);
      char second_char = raw_input.charAt(1);
      if (id_char == 'V') {
        if (second_char == 'M') {
          int brushless_min_speed = raw_input.substring(2).toInt();
          return brushless_min_speed;
        }
        else if (second_char == 'F') {
          int brushless_speed = raw_input.substring(2).toInt();
          return brushless_speed;
        }
        else if (second_char == 'C') {
          int brushless_speed = raw_input.substring(2).toInt();
          return brushless_speed;
        }
        else if (second_char == 'D') {
          int brushless_speed = raw_input.substring(2).toInt();
          return brushless_speed;
        }
      }
    }
  }
}


int get_button(char setup_or_mount) {
  if (setup_or_mount == 's'){
    Serial.println("button check");
    xbee.println('s');
    delay(DELAY_XBEE);
    while (true) {
```

```
      if (xbee.available()) {
        String raw_input = xbee.readStringUntil('\n');
        Serial.println("raw_input");
        char id_char = raw_input.charAt(0);
        if (id_char == 's') {
          int mount_button_input = raw_input.substring(1).toInt();
          return mount_button_input;
        }
      }
    }
  }
  else if (setup_or_mount == 'm'){
    xbee.println('m');
    delay(DELAY_XBEE);
    while (true) {
      if (xbee.available()) {
        String raw_input = xbee.readStringUntil('\n');
        char id_char = raw_input.charAt(0);
        if (id_char == 'm') {
          int mount_button_input = raw_input.substring(1).toInt();
          return mount_button_input;
        }
      }
    }
  }
}

bool check_button_status(char setup_or_mount) {
  if (get_button(setup_or_mount) == HIGH) {
    return true;
  }
  else {
    return false;
  }
}

void lcd_message(String input_message) {
  xbee.println(input_message);
}
```