**Trailbot Critical Design Review Report**

Team 14: Brandon Christler, Michael Hitchcock, Garrett Monast, Yifan Wu

ECE 4900

December 7th, 2018

# Table of Contents

[**Table of Contents**](#_lv3wnilzn0p7) **2**

[**List of Figures**](#_ddq6fz89upaa) **2**

[**List of Tables**](#_x3gho02hv5jx) **2**

[**Introduction**](#_8kz30gjkr8j0) **4**

[**Technical**](#_gg4p3t8wjcts) **4**

[Requirements and Specifications](#_hckiiioxv95l) 4

[Alternate Design Features](#_nr5hppgqzwor) 6

[Design](#_voqqxj47nvj4) 7

[Alternate designs or solutions](#_fqjuengrwccf) 11

[Regulation](#_yvu2kv2mkp45) 11

[Implementation](#_vu684su4mpbv) 12

[Test and Validation](#_wqh1d0i8r5yh) 15

[**Project Management**](#_io6yb814mm5x) **17**

[**Recommendations**](#_1xgtwsgwrs4t) **21**

[**Conclusion**](#_2zxr15li3eln) **21**

[**Bibliography and References**](#_g7pqmbkgo72b) **22**

[**Appendix: Project Code**](#_7lgo7up5cb0i) **23**

# List of Figures

# 

NOT POSSIBLE IN GOOGLE DOCS. WILL COMPLETE IN WORD AFTER CONTENT IS COMPLETE.

# 

# List of Tables

NOT POSSIBLE IN GOOGLE DOCS. WILL COMPLETE IN WORD AFTER CONTENT IS COMPLETE.

**Executive Summary**

Trailbot was developed as a consumer product to ease the process of going out for a run. After multiple rounds of brainstorming and research, Team 14 settled on a ground based robot to provide functionality to a consumer while they ran. Based on a commercial remote control car chassis, a robot was developed that was able to follow a user wearing an infrared beacon while it provided additional functionality such as carrying small objects and illuminating a trail during low-light conditions.

This report summarizes the process of developing this product. The problem statement that spurred this project is discussed, followed by the technical requirements developed to overcome the problem. The design is then discussed, including the processes used in the development and analysis of various aspects of the project, such as the constraints and relevant standards. The implementation and testing processes and results are covered before moving on to the group’s project management, conclusion, and final recommendations.

A brief review of the team’s project management concerns touching on risks, challenges, and budget naturally leads to a conclusion where the ultimate results of the project are discussed. Although all user tracking features including obstacle detection were successfully implemented and tested independently, when they were integrated together for a final prototype all functionality was severely impeded with direction and speed controls lagging significantly from the desired behavior. Although accessory functionality was successfully implemented, the time constraints of the project combined with the difficulties in developing an effective tracking system resulted in a cutting of features from what was initially intended. Despite these setbacks, the team remained optimistic about the potential of the Trailbot project; with more time and development resources, Trailbot could become a successful product.

# 

# Introduction

Trailbot was created to solve the following problem statement:

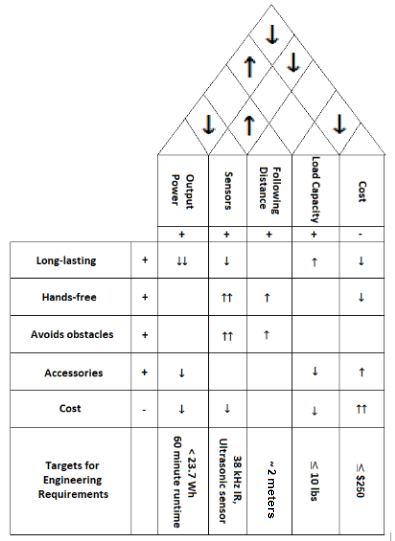
“Jogging is meant to be healthy and relaxing, but the challenges of busy schedules and the ever increasing number of personal gadgets to carry has made it hard to just get out onto the trail.”

In order to develop a prototype capable of solving these issues, Team 14 designed and built a robot capable of following a user while providing convenient functionality. This report details the technical challenges involved, from developing requirements to testing a final prototype, as well as relevant project management methods used to do so. In the end, this report will summarize the entirety of the project to develop the product known as Trailbot and review the successes and failures of the effort.

# Technical

## Requirements and Specifications

The requirements of the Trailbot design included a minimum runtime of one hour, a standard duration for a jog, a speed comparable to that of the average distance runner, approximately 25 kph, a range keeping accuracy within a fifth of a meter, and the ability to provide light to the user. It must also be safe for its user, those nearby, and its environment. Finally, it must be able to withstand reasonable outdoor conditions and provide enough protection to its electronics to ensure reliable function. All of these features would be achieved for a total cost of under $250. Figure 1 below shows the House of Quality for the design with its desirable engineering and marketing requirements along the top and left side as well as its quantitative technical requirements along the bottom. Table 1 gives a brief overview of the requirements for the design, and the features that were determined through testing which satisfy those requirements.



*Figure 1: House of Quality for Design*

*Table 1: Requirements and Specifications*

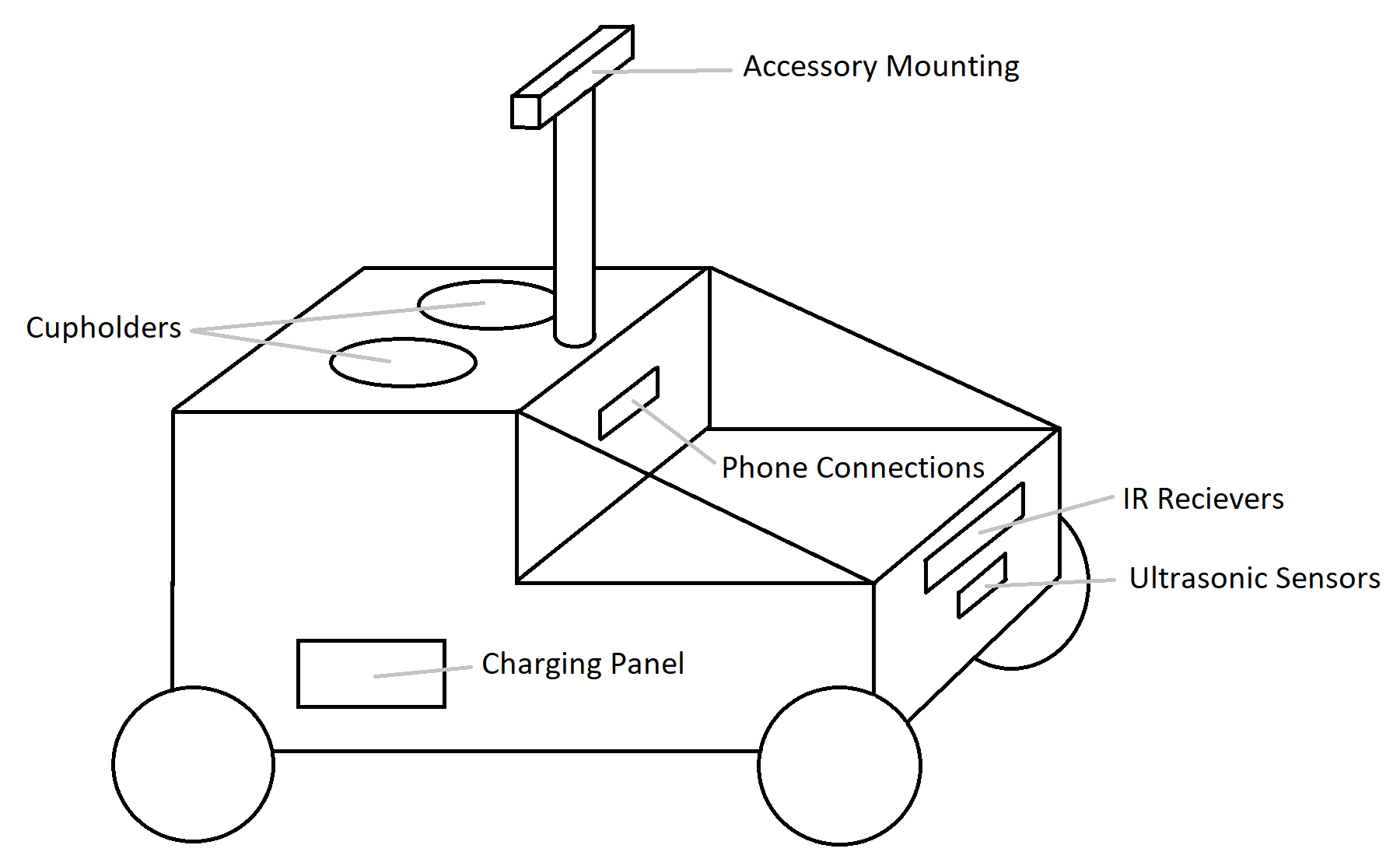
|  |  |
| --- | --- |
| **Requirements** | **Specifications** |
| Battery life long enough for average run time | Runtime ≥ 1 hr |
| Must match speed of average distance runner | Target max speed of 25kph |
| Accessory Functionality | Body must be easy to modify, repair, and mount to  Provide illumination sufficient to avoid hazards on night runs |
| Tracking ability | Maintain distance of ~2m, ±0.2m |
| Cost | Maximum cost of $250 |
| Safety | Easy operation; design must mitigate shock damage and collision with user and surroundings. |
| Environmental | No damage to environment, no disturbances to animals |
| Durability | Withstand average outdoor conditions |

## Alternate Design Features

The few design features that ultimately were changed were the quantity and types of sensors used for tracking and obstacle detection and the removal of the phone charging capability as a function. Team 14 found that in order to better track the user, a more accurate primary tracking sensor and a greater number of secondary tracking sensors directed at several angles were required. In addition, more ultrasonic sensors were needed to cover more space around the rover where obstacles could be. Furthermore, an IR beacon was developed to communicate 38-kHz transmitters and receivers with specific IR light. Finally, the group removed the feature of phone-charging due to complexity and logistical constraints. In summary, the features of a unidirectional IR receiver array and a single ultrasonic sensor for obstacle detection were ultimately rejected in addition to the phone charging capability originally included in the design requirements.

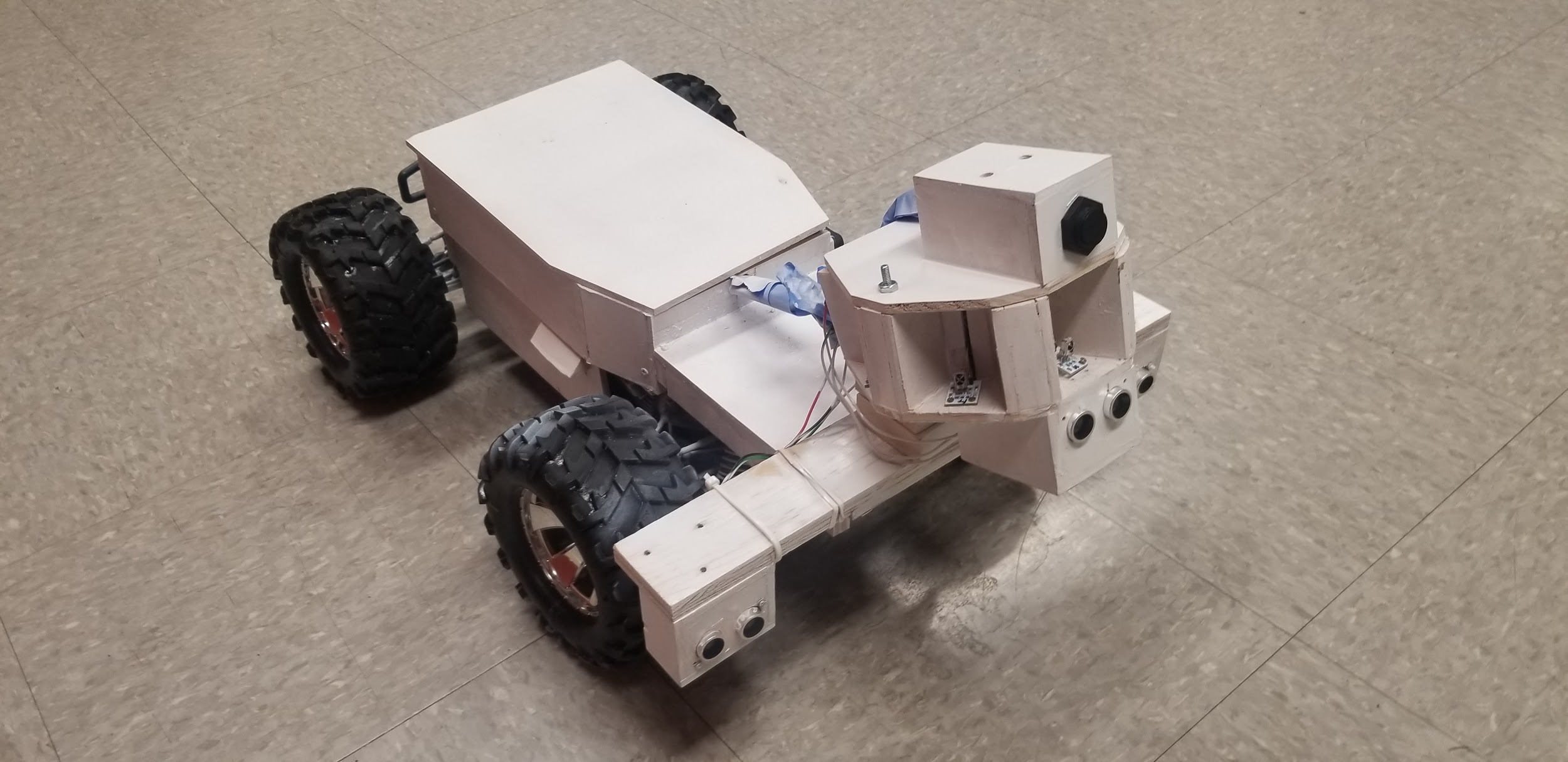
## Design

The group began the design process with the chassis obtained from the class workspace. The body design was based on the dimensions of the chassis, the requirements, and desired functions of the prototype. The body would include a pole mount on which LED lights would be mounted, a compartment for small items, and a cup holder. The body would also include a housing for the sensors which developed as the project progressed.



*Figure 2: Original Prototype Design*

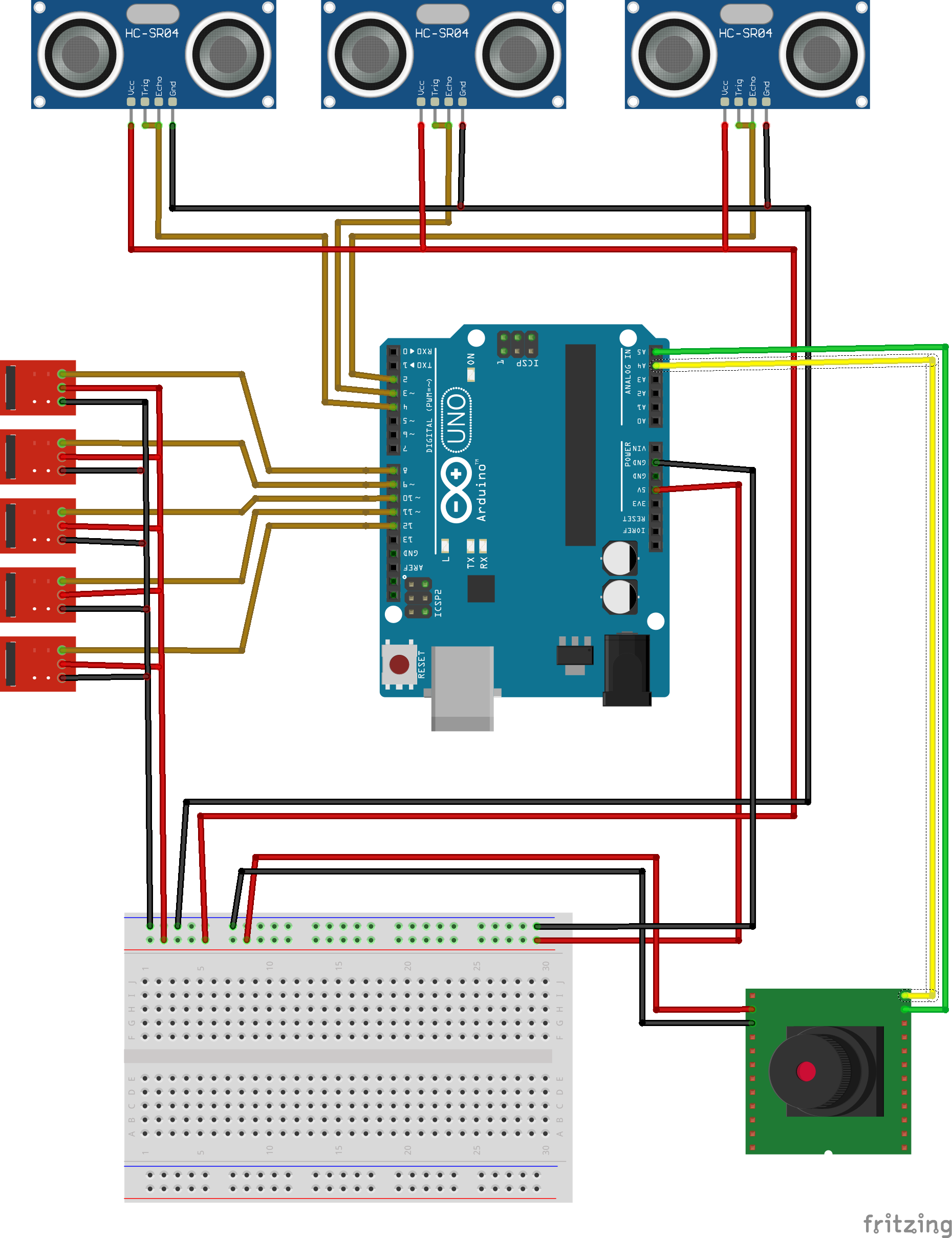
The original design proposal, shown by Figure 2, had an IR receiver array in one direction only and a single ultrasonic sensor for obstacle detection. The receivers would be housed in the front portion of the body near the bottom of the body looking out towards the front of the vehicle. The final design, shown by Figure 3, included five 38-kHz IR receivers arranged in a 180° array designed to assist a single IR camera with a 33° x 26° cone of vision. The IR camera was implemented as the primary tracking device and the IR array as the secondary device. Three ultrasonic sensors were used in the final design, one in the IR housing and two mounted above each of the front wheels. The second and third ultrasonic sensors were added to better detect obstacles to the sides of the bot.



*Figure 3: Prototype Without Accessory Functions*

In addition to the sensors, the major electronics of the design included one brushless DC motor, one servo attached to the steering column, and a single Arduino Uno which served as the microcontroller for all of the movement and sensor systems. The Uno, with a screw terminal mounted to ensure more reliable connections, was planned to be housed in the lower portion of the body for the original design, but ended up in a new compartment between the upper and lower portions of the body, which housed the accessories and motor system respectively.

Figure 4 gives the hardware block diagram which shows the integration of all of the sensors onto the microcontroller. Note the five 38-kHz IR receivers on the top, the three ultrasonic sensors on the right, and the single IR camera in the lower left corner. All of the IR receivers and the ultrasonic sensors interfaced with the Uno using digital pins, while the IR camera required the use of two analog pins. Power was shared among all of the sensors from the Uno’s 5V pin.



*Figure 4: Functional Hardware Block Diagram*

Figure 5 below shows the software flowchart of the prototype. The path began with a general startup, followed by an update of the readings obtained from the ultrasonics. If no obstacles were detected, the microcontroller read data from all IR sensors to determining the relative position of the IR source to the bot’s front and subsequently adjusts the steering angle if necessary. Steering, particularly when the IR source is outside of the camera’s view, is determined by weighting the response of the IR receivers proportional to their deviance from center; receivers on the left and right of the housing will trigger a sharper turn than those looking more closely towards the center.Distance was estimated using a combination of the middle ultrasonic sensor, which will trigger a stop if the user is too close, and the Y-coordinate of the IR source. If the user is far away, the IR source appears to the camera to be lower than a specified Y-coordinate and Trailbot will increase speed. Conversely, if the user is too close, but not so close as to trigger the ultrasonic, the IR source will appear to be higher than a specified Y-coordinate and the bot would slow down.



*Figure 5: Software Flowchart*

The primary hardware design considerations for the prototype were battery life, reliability, and the ability to mount accessories. The secondary considerations were weight and dimensions as it would likely have to be carried at times by the user as well as its ruggedness. The major considerations for the software side of the design were tracking ability and obstacle detection. The software path, specifically for the ultrasonic sensors, used an Arduino library called Newping, which mitigated the delay caused by particular commands inherent to the standard Arduino ultrasonic libraries. This library uses interrupts, rather than delays, to respond to events detected by the ultrasonics. In addition to the obstacle detection function, the group intended to implement a Proportional-Integral-Derivative method of control, to provide smoother responses to the user’s movements. PID control would takes in the error between a set point and the process variable and make corrections based on the the proportional, integral, and derivative terms. This implementation would have improved the smoothness of steering, but was dropped as time constraints forced compromises into the design.

## Alternate designs or solutions

At the beginning of the project, Team 14 originally designed a quadcopter that would follow the user while providing other functionality. Due to the nature of aerial vehicles, quadcopters rely on complex control methods and require expensive electronic devices in order to stay in the air. In addition to these examples, weight is a significant constraint that further hinders the vehicles capability to solve the problem statement by limiting the number of sensors, battery life, and possibilities for accessory mounting. Upon completion of a prototype, it became clear that a quadcopter was beyond both the budget and capabilities of the group. As a result, Team 14 decided to change the design of the Trailbot from a quadcopter to a rover like ground-based vehicle.

Upon further research of ground-based vehicles capabilities vs quadcopter capabilities, it became clear that the ground-based design was superior for the scope of the group’s project. One of the largest advantages the ground-based device had was the ability to hold significant amounts of weight. This allows the design to have better control due to more sensors and a significantly longer battery life. Another advantage the ground-based vehicle had over the quadcopter was fewer complex controls. Due to the vehicle being on the ground, the vehicle only needs to control the speed and orientation compared to the aerial vehicles yaw, pitch, roll, and height. Although the ground base-vehicle had more advantages, there was one area where the quadcopter performed significantly better. Due the quadcopters being lightweight and having extra degrees of freedom, it was able to maneuver faster with more possibilities than the ground-based vehicle. Despite this one advantage, the ground-based vehicle was considered the better option and was determined to be the design moving forward. A summary of the advantages and disadvantages of the two designs can be located within table 2 below.

*Table 2: Characteristics and Advantages of Quadcopter vs Ground-Based Vehicle .*

|  |  |  |
| --- | --- | --- |
|  | **Drone** | **Rover** |
| **Maneuverability** | **X** |  |
| **Weight Limit (for sensors)** |  | **X** |
| **Accessory Mounting** |  | **X** |
| **Battery Life** |  | **X** |
| **Cost** |  | **X** |
| **Complexity** |  | **X** |

## Regulation

At the beginning of the project, the group committed time to thoroughly researching possible standards that might apply to the Trailbot or projects like it. Despite careful research, the group was not able to find any relevant standards that the project would need to meet independent of those already satisfied by the components purchased. Due to the apparent lack of relevant standards, the group did not set any standards that needed to be satisfied upon the completion of the project. For a final, manufactured product there would be a responsibility to ensure that Trailbot meet basic standards for electronics as laid out by the FCC in the United States and similar standards in the EU and other regulatory environments.

## Implementation

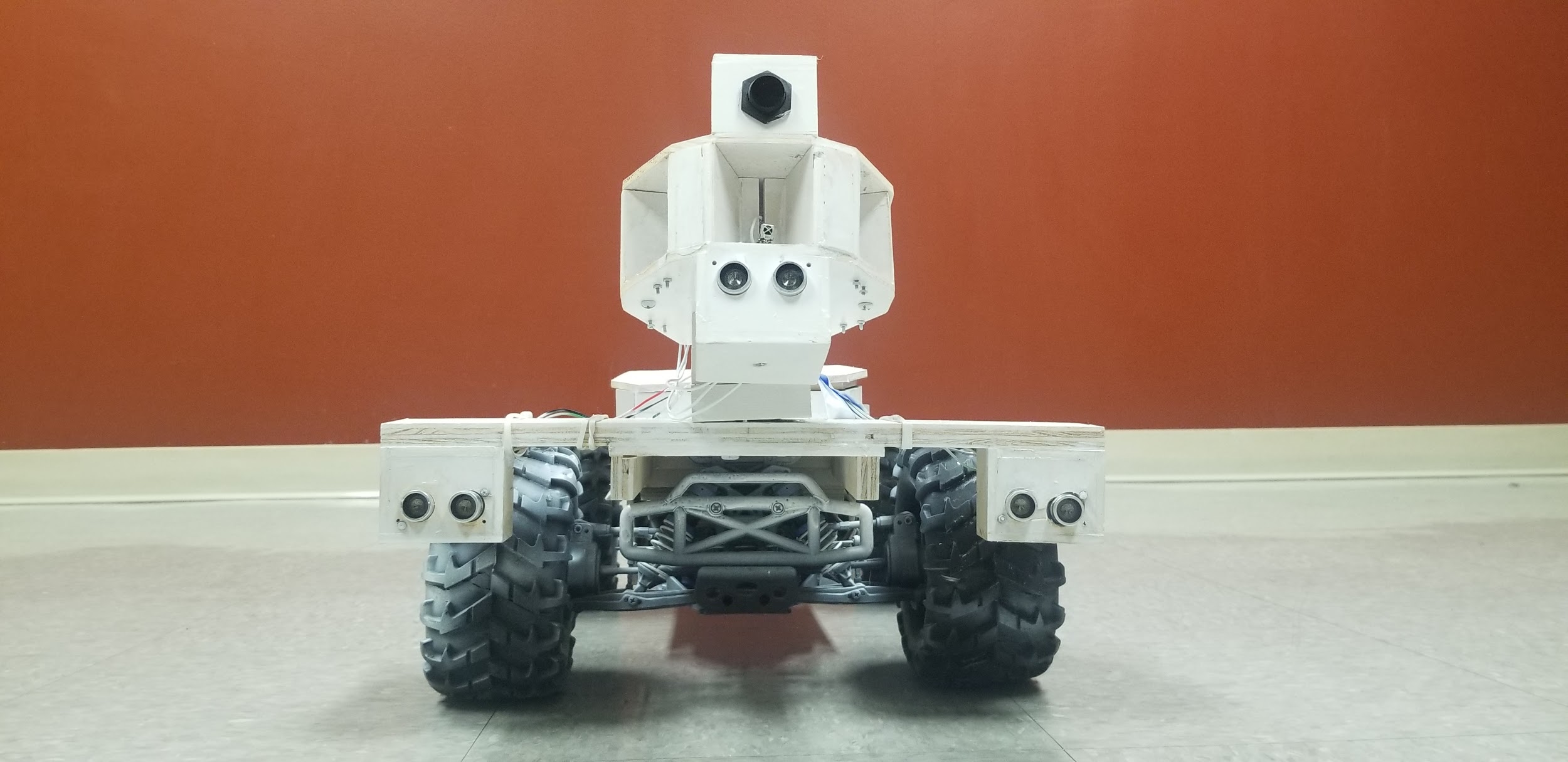
Trailbot is a user-following, obstacle-detecting, item-carrying, light-providing rover robot, which provides a solution to fitness enthusiasts who jog or run along paved tracks or trails anytime from dawn to dusk, and who do not wish to be encumbered by their myriad of small personal belongings such as water bottles, keys, and phones.

The prototype is designed on a solid, rugged chassis and consists of three major sections constructed primarily with plywood. The lower section houses the brushless DC motor, the electronic speed controller, the steering servo, and the battery. The middle section along with the sensor housing holds the electronics including the Arduino Uno, five 38-kHz IR receivers, three ultrasonic sensors, and a single IR camera. The upper portion of the body includes the small item compartment, a bottle or cup holder, and pole mount for LED lights.

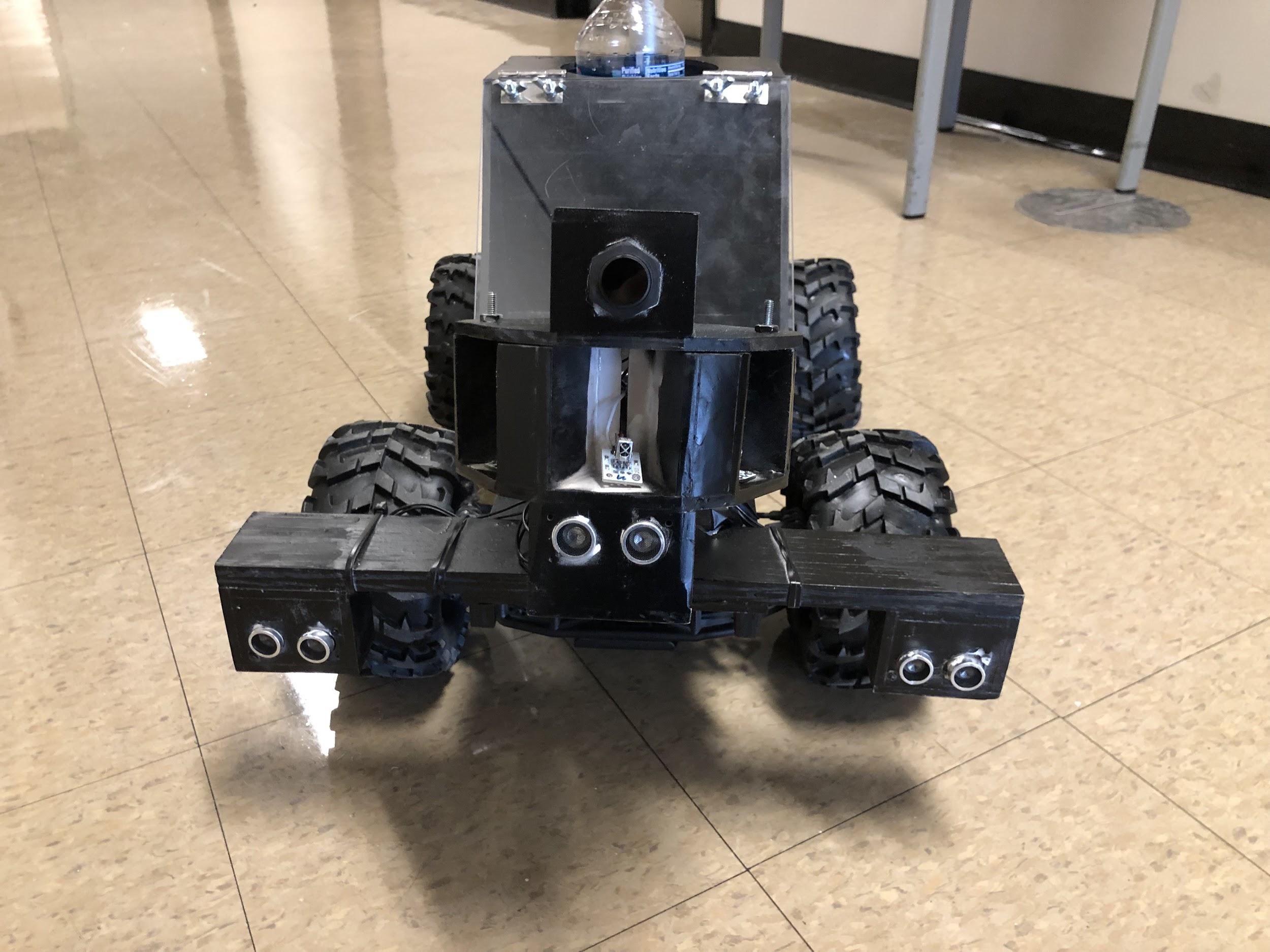
Trailbot is intended for use on paved tracks and trails and follows the user via an infrared beacon mounted on their waistband. The beacon has two IR emitters: a constant-emission LED and a 38-kHz LED. The combination of these two allow the beacon to transmit to both the IR camera and the IR receiver array mounted on the rover resulting in more reliable communication. While the body of the prototype is constructed using plywood, the finished product would have a body made of molded plastic. In addition, the chassis of the finished product would be more forgiving of the mounted load, and would maintain its unloaded balance. Figures 6 and 7 below are images of the prototype, before paint and accessory mounting 一 Figures 8 and 9 show the final prototype.



*Figure 6: Side view of prototype, pre-paint and pre-accessories.*



*Figure 7: Front view of prototype, pre-paint and pre-accessories.*



*Figure 8: Front view of final prototype.*



*Figure 9: Final Trailbot prototype with accessories mounted.*

The technical specifications of the prototype include:

* Dimensions: L x W x H = 0.660m x 0.381m x 0.356m (~0.5m with accessories added)
* Weight: 8.62kg
* Microcontroller Model: Arduino Uno
* Sensors

→ IR Camera

Product Code: RB-Dfr-553

Cone angle: 33° x 23°

→ IR Receivers: Velleman 1838, 38-kHz

→ Ultrasonic Sensors

Model: HC-SR05

Max Range: 4.5m

* Battery Supply: 14.8V, 5200mAh

## Test and Validation

From the very beginning, testing has been an important procedure that was crucial for the development of the Trailbot. Once the chassis and sensor station were completed, the group started to develop testing methods to verify the ability of the Trailbot to meet all of the original design specifications. The testing completed can be broken down into four categories: sensors, movement, sensors controlling movement, and final testing.

The first group of tests isolated the sensor station away from the movement of the Trailbot. These tests were mainly done at the workbench, and informed the group on the capabilities and limitations of the sensors being used. First, the ultrasonic sensors were tested to verify their accuracy and maximum range. After this, the infrared camera was tested to determine the range that the infrared beacon needed to be within to properly communicate with the camera. Finally, the group tested the infrared receivers to confirm they could communicate with the beacon and that they were isolated enough to be effective at locating the user. This group of testing provided valuable data that helped shape the end result; the results of the infrared receiver array tests confirmed that the receivers were not isolated enough, and that a better housing needed to be made. Another discovery made during this group of testing was the ultrasonics sensors could not reliably detect a distance over one and a half meters. This directly impacted the obstacle detection programming and the rate of braking when an obstacle is detected.

The second group of tests isolated the movement of the Trailbot away from the sensor station. These tests were essentially the programing of the vehicle to perform a specific motion so that commands could be associated with specific movements. The first test isolated the speed of the vehicle to verify the vehicle can achieve the required speed and understand which speed commands are equivalent to the desired speed. From this test, the group was able to confirm the Trailbot could easily achieve the desired speed of 25 kph. The second test isolated the steering of the vehicle, in order to determine the possible range of the steering commands along with the value necessary to maintain a straight line of movement.

The third group of testing utilized the sensors to control the movement of the Trailbot. These tests allowed the group to fine tune the programming so that the vehicle could follow the user and adjust its speed according to the user. The first test focused solely on the vehicle following the user. This test gave valuable information on the algorithm used to compute the steering command based on where the user was. The second test completed focused solely on the vehicle being able to adjust speeds according to the user, and provided data on how fast the vehicle should speed up or slow down based on the beacon’s coordinates.

The fourth and final group of testing combined all aspects of the Trailbot, and was used to verify all of the predetermined requirements and specifications. At the end of the project, all of the requirements and specifications in Table 3 were met except the desired tracking ability. The Trailbot’s ability to calculate the steering angle and speed based on the location of the beacon worked to a degree; however, the code will need further revision in order for the Trailbot to follow the user effectively enough to fully meet the requirements.

*Table 3: Requirements for the design, and features determined through testing.*

|  |  |
| --- | --- |
| **Requirements** | **Features** |
| Runtime ≥ 1 hr | Testing determined a continuous use time of more than two hours. |
| Target max speed of 25 kph | Brushless DC motor achieves 45 kph easily. |
| Accessory Functionality | Plywood body facilitates modification and construction. |
| Tracking ability: be able to follow at a distance of approx. 2m ±0.2m | 1 IR camera, 5 IR receivers to locate and gauge distance from beacon from 0m to 10m ±0.2m |
| Safe for user and surroundings | 3 Ultrasonic sensors prevent collisions. |
| Durability | Sturdy chassis and plywood body make the design capable of withstanding typical outdoor conditions. |

# 

# Project Management

Figure 10 below is a Gantt Chart detailing the timeline of the project. Table 4 below that shows the distribution of tasks among the group members. The milestones of the project, as seen in the Gantt Chart, are concentrated near the end of the project timeline, and include:

1. Achieve robot mobility functions, including sensors, **Nov. 10**.
2. Install carrier and peripherals, to complete the body, **Nov. 24**.
3. Verify all functions in comprehensive operational test, including all components, **Dec. 1**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 | W11 | W12 | W13 | W14 | W15 | W16 |
|  | 21-Aug | 27-Aug | 3-Sep | 10-Sep | 17-Sep | 24-Sep | 1- Oct | 8-Oct | 15- Oct | 22- Oct | 29-Oct | 5-Nov | 12-Nov | 19-Nov | 26-Nov | 3-Dec |
| Design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Chassis |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Controller |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Software |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sensors |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Additional Functions |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Build |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Chassis |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Controller |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sensors |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Test |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Movement |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Following |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Obstacle detection |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Additional Functions |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Present |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Records |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Presentation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Milestones** |  |  |  |  |  |  |  |  |  |  |  | **11/10** |  | **11/24** | **12/1** |  |

*Figure 10: Gantt Chart*

*Table 4: General distribution of tasks*

|  |  |
| --- | --- |
| Design Rover Body | Michael, Garrett, Brandon, Yifan |
| Design Beacon | Michael, Garrett, Brandon, Yifan |
| Design Software | Michael, Garrett, Brandon, Yifan |
| Design Electronics | Michael, Garrett, Brandon, Yifan |
| Build Rover Body | Michael |
| Design/Build Electronics | Michael, Garrett, Brandon, Yifan |
| Build Beacon | Garrett |
| Develop Software | Brandon, Yifan |
| Design Additional Features | Michael, Garrett |
| Initial Testing | Brandon |
| Final, Comprehensive Testing | Michael, Garrett, Brandon, Yifan |
| Reports/Presentations | Michael, Garrett, Brandon, Yifan |
| Documentation | Michael, Garrett, Brandon, Yifan |

At the beginning of the project, the group assigned Michael to the construction of the body, Garrett to research electronics and their implementation, and Brandon and Yifan to work on the software. These three processes could largely be done in parallel, so that no one section would have to wait for the completion of another. As the semester progressed, several bottlenecks occurred resulting in delays to various portions of the project and new tasks came up which needed to be assigned. Upon build completion, all of the team members became involved in testing and validation as well as the revision of the code. Toward the end of the project timeline, Garrett took on the new task of the design and construction of the beacon while Brandon carried out the initial testing. With majority of the team working in parallel, additional team assignments were primarily completed by those whose work was at a slow period. Table 5 below gives a brief detailing of the progression of responsibilities.

*Table 5: Progression of responsibilities/assignments.*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Beginning** | **Middle** | **End** |
| **Body** | Michael | Michael | Finished |
| **Electronics** | Garrett | Michael | Finished |
| **Beacon** | Not Assigned | Garrett | Garrett |
| **Software** | Brandon, Yifan | All | Michael, Brandon, Yifan |
| **Testing** | All | All | Michael, Brandon, Yifan |

The resources for the project included materials, software, and workspaces. The group primarily used the Capstone workspace for team meetings and performed most of their designing, testing, troubleshooting, and coding there. Michael’s home workshop also proved to be a valuable resource particularly for the construction of the body. The group used several professionally-manufactured devices in the completion of the project, such as the Arduino microcontroller, ultrasonic sensors, and IR sensors. Common test equipment, such as multimeters and power supplies, were used to verify proper function and aid in troubleshooting. Soldering equipment was used to produce more reliable connections between components/devices. The other resources used, hardware, software, and reference, include common electronic components and tools, the Arduino IDE, and documentation provided from product manufacturers.

The team decided early on in the project that acquiring the funds, and later on the necessary parts, through the Capstone program would take valuable time, and that it would be prudent to fund the project personally. The final cost for the project, including neither an estimated value for the chassis nor the work-hours put in, can be estimated to be around $120. Equally distributed, this cost was not infeasible for the group members. The maximum budget for the class was never officially discussed, but the target cost for the prototype, including the chassis value, was set at $250, which provided a reasonable allocation for the chassis. The money went primarily towards electronics such as the microcontroller and the sensors as well as the screw terminal board and other minor components, and secondarily towards the materials for the body. No facility or software costs were necessary for the project.

The risks initially identified for this project included both logistical and safety risks. One logistical risk was that the team would not allocate the proper amount of time for any particular task, fall behind schedule, and ultimately sacrifice prototype functionality; another was that the materials needed, determined through research, would not be available, and that the team would therefore sacrifice prototype functionality. To mitigate these logistical risks, the team attempted to think ahead and execute proper planning to identify and order necessary parts and distribute tasks appropriately. Safety risks included electric shock to the group members while working on the prototype, and physical harm sustained through collision as a result of the potential erratic behavior from the prototype Trailbot 一 a sturdy chassis and a powerful motor made this risk of harm very real. To mitigate these safety risks, the group worked carefully with the electronics, ensuring power was disconnected before handling them directly, and performing many steering and tracking tests without incorporating speed; such bench testing proved essential to ensure safety and reliability.

There were many issues, major and minor, that occurred throughout the project. The major problems encountered, and their solutions, include:

* **Unreliable connections through fragile splicing and connectors.**

→ Screw terminal board attached to Uno.

* **IR receivers don’t differentiate between pure IR beacon and natural light.**

→ 38-kHz transmitters and receivers used to communicate with specific IR light.

* **IR receivers are not well-isolated; beacon signal activates all receivers.**

→ Solid housing built for IR receivers, to isolate and direct beacon signals.

* **Processing time required for US pulses causes significant delays in bot’s response time to user movements and obstacle detection.**

→ NewPing library used to run processes concurrently, avoiding lag.

* **Ultrasonic sensors sent and received pulses at different rates.**

→ Solved by using not only a consistent model, but a consistent manufacturer.

* **Sensors produced data with outliers.**

→ PID control and interval averaging used to reduce outlier impacts.

# Recommendations

For moving forward from prototype to a consumer product, Team 14 recommends a variety of actions. These actions can broadly be defined as short term steps for improvements in additional rounds of prototyping and long term changes necessary to facilitate a positive consumer experience.

For additional prototyping, there were many lessons learned by Team 14 that should be incorporated into future prototypes for Trailbot. The most significant of these recommendations would be to allow more time for research and design so that problems can be anticipated prior to construction. Specific problems that should be considered during this extended research and design period should include, at a minimum, cable and connection management, chassis weight and suspension limitations, and sensor accuracy and appropriate application. By focusing on these three areas of improvement for the next stage of prototyping, severe problems involving rebuilds, poor performance, and unresponsive control from Trailbot may be mitigated or resolved.

Once additional prototyping and testing has been conducted, several factors will need to be considered in the transition to a mass-produced consumer product. The most obvious piece to be done differently would be the body; in order to reduce weight and cost while improving weather resistance and aesthetics, the body would be made from injection molded plastic rather than wood and would be decorated in a variety of colors and patterns. Another important factor to be changed from prototypes to a final product would be the battery charging procedure. In the current prototype, the battery has to be manually disconnected from the system and attached with two connections to a separate battery charger, but for a final product a simple, one plug system would be necessary.

# Conclusion

Although Trailbot proved to be an ambitious undertaking for the skills and experience of Team 14, the team managed to achieve its goals on a basic level while developing their engineering and team-working abilities. Trailbot failed to perform to the level targeted initially, but each necessary subsystem functioned well in independent testing which leads Team 14 to believe that with further research, testing, and refinement, the various subsystems can be better integrated in order to fully achieve the requirements laid out in this report. Despite the variety of setbacks that occured over the course of the Trailbot project, Team 14 remains confident that theTrailbot concept is both feasible and valuable.

# Bibliography and References

I sure hope we can come up with some of these.

# Appendix: Project Code

#include <Servo.h>

#include <Wire.h>

#include <IRCam.h>

#include <NewPing.h>

IRCam irCam;

Servo esc;

Servo steering;

const unsigned long KILL\_TIMER = 30000;

//US sensor assignment

#define SONAR\_NUM 3 // Number of sensors.

#define MAX\_DISTANCE 250 // Maximum distance (in cm) to ping.

#define PING\_INTERVAL 40 // Milliseconds between sensor pings (29ms is about the min to avoid cross-sensor echo).

#define US\_PIN1 2

#define US\_PIN2 3

#define US\_PIN3 4

//IR sensor assignment

//These sensors are active LOW and should be read with !digitalRead()

#define IR\_PIN1 8 // ir sensor 1 (left most) on pin 5

#define IR\_PIN2 9 // ir sensor 2 on pin 6

#define IR\_PIN3 10 // ir sensor 3 (center) on pin 8

#define IR\_PIN4 11 // ir sensor 4 on pin 7

#define IR\_PIN5 12 // ir sensor 5 (right most) on pin 10

//servo pin assignments

#define STEERING\_PIN 5 // steering servo pin

#define ESC\_PIN 6 // esc pin

int center = 72;

int ir\_ang = center;

int steer\_ang = center;

int max\_left = 140;

int left\_max = max\_left;

int max\_right = 20;

int right\_max = max\_right;

int ir\_ypos = 0;

int cur\_speed = 1550;

unsigned long timer = 0;

unsigned long speed\_change\_time = 0;

unsigned long speed\_delay = 250;

int ir\_rec\_array[6] = {0, 0, 0, 0, 0, 0}; //holds 0 or 1 based off of digital read of ir recievers

int last\_rec\_ir\_ang = center;

unsigned long last\_camera\_time = 0;

unsigned long pingTimer[SONAR\_NUM]; // Holds the times when the next ping should happen for each sensor.

unsigned int cm[SONAR\_NUM]; // Where the ping distances are stored.

uint8\_t currentSensor = 0; // Keeps track of which sensor is active.

unsigned long brake\_timer = 0;

bool brake\_complete = true;

unsigned brake\_delay = 500;

int obstacle1 = 20;

int obstacle2 = 20;

int obstacle3 = 20;

//data smoothing

const int numReadings = 5; //number of readings held for smoothing

int us1\_readIndex = 0;

int us1\_readings[numReadings];

int us1\_total = 0;

int us2\_readIndex = 0;

int us2\_readings[numReadings];

int us2\_total = 0;

int us3\_readIndex = 0;

int us3\_readings[numReadings];

int us3\_total = 0;

int range\_us1 = 100; // stores SMOOTHED range detected by us1 - should be target!

int last\_range\_us1 = 100;

int range\_us2 = 100; // stores SMOOTHED range detected by us1 - should be target!

int last\_range\_us2 = 100;

int range\_us3 = 100; // stores SMOOTHED range detected by us1 - should be target!

int last\_range\_us3 = 100;

//ultrasonic sensors

NewPing sonar[SONAR\_NUM] = { // Sensor object array.

NewPing(US\_PIN1, US\_PIN1, MAX\_DISTANCE), // Each sensor's trigger pin, echo pin, and max distance to ping.

NewPing(US\_PIN2, US\_PIN2, MAX\_DISTANCE),

NewPing(US\_PIN3, US\_PIN3, MAX\_DISTANCE)

};

void setup() {

// put your setup code here, to run once:

esc.attach(ESC\_PIN);

steering.attach(STEERING\_PIN);

esc.writeMicroseconds(1500);

steering.write(center);

irCam.begin(); // begin reading IR camera

Serial.begin(9600);

pingTimer[0] = millis() + 75; // First ping starts at 75ms, gives time for the Arduino to chill before starting.

for (uint8\_t i = 1; i < SONAR\_NUM; i++) // Set the starting time for each sensor.

pingTimer[i] = pingTimer[i - 1] + PING\_INTERVAL;

for (int thisReading = 0; thisReading < numReadings; thisReading ++) { // start smoothing at zero

us1\_readings[thisReading] = 0;

us2\_readings[thisReading] = 0;

us3\_readings[thisReading] = 0;

}

Serial.println("Starting program...");

esc.writeMicroseconds(cur\_speed);

}

void loop() {

for (uint8\_t i = 0; i < SONAR\_NUM; i++) { // Loop through all the sensors.

if (millis() >= pingTimer[i]) { // Is it this sensor's time to ping?

pingTimer[i] += PING\_INTERVAL \* SONAR\_NUM; // Set next time this sensor will be pinged.

if (i == 0 && currentSensor == SONAR\_NUM - 1) oneSensorCycle(); // Sensor ping cycle complete, do something with the results.

sonar[currentSensor].timer\_stop(); // Make sure previous timer is canceled before starting a new ping (insurance).

currentSensor = i; // Sensor being accessed.

cm[currentSensor] = 0; // Make distance zero in case there's no ping echo for this sensor.

sonar[currentSensor].ping\_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to look for echo).

}

}

//IR recievers read

ir\_rec\_array[1] = !digitalRead(IR\_PIN1); // read ir 1

ir\_rec\_array[2] = !digitalRead(IR\_PIN2); // read ir 2

ir\_rec\_array[3] = !digitalRead(IR\_PIN3); // read ir 3

ir\_rec\_array[4] = !digitalRead(IR\_PIN4); // read ir 4

ir\_rec\_array[5] = !digitalRead(IR\_PIN5); // read ir 5

//IR camera read

irCam.update(); //required to update camera readigs

//calculates ir angle from camera

if (irCam.p1.x > 0 && irCam.p1.x <= 1022) {

last\_camera\_time = millis();

for (int i = 0; i <= 36; i++) {

if (irCam.p1.x > i \* (1000 / 36) && irCam.p1.x <= (i + 1) \* (1000 / 36)) {

//last\_cam\_ir\_ang = ((center + (36 / 2)) - i);

ir\_ang = ((center + (36 / 2)) - i);

}

}

}

//ir angle from ir recievers

if (ir\_rec\_array[1] == 1) {

if (ir\_rec\_array[2] == 1) {

if (ir\_rec\_array[3] == 1) {

last\_rec\_ir\_ang = center + 15; //1,2,3

}

else {

last\_rec\_ir\_ang = left\_max - 10; //1,2

}

}

else {

last\_rec\_ir\_ang = left\_max; //1

}

}

else if (ir\_rec\_array[5] == 1) {

if (ir\_rec\_array[4] == 1) {

if (ir\_rec\_array[3] == 1) {

last\_rec\_ir\_ang = center - 15; //5,4,3

}

else {

last\_rec\_ir\_ang = right\_max + 10; //5,4

}

}

else {

last\_rec\_ir\_ang = right\_max; //5

}

}

else if (ir\_rec\_array[3] == 1) {

if (ir\_rec\_array[2] == 1) {

if (ir\_rec\_array[4] == 1) {

last\_rec\_ir\_ang = center; //3,2,4

}

else {

last\_rec\_ir\_ang = center + 15; //3,2

}

}

else if (ir\_rec\_array[4] == 1) {

last\_rec\_ir\_ang = center - 20; //3,4

}

else {

last\_rec\_ir\_ang = center; //3

}

}

else if (ir\_rec\_array[2] == 1) {

last\_rec\_ir\_ang = center + 30; //2

}

else if (ir\_rec\_array[4] == 1) {

last\_rec\_ir\_ang = center - 30; //4

}

// put your main code here, to run repeatedly:

if (brake\_complete == true) {

if (millis() - last\_camera\_time > 250) { //using ir camera

//this code turns to the left

//the further the ir angle is from the currently set steering angle, the larger the turn increments are

if (ir\_ang > steer\_ang) {

if (ir\_ang - steer\_ang >= 30 && steer\_ang < (max\_left - 10)) {

steer\_ang = steer\_ang + 10;

steering.write(steer\_ang);

}

else if (ir\_ang - steer\_ang >= 15 && steer\_ang < (max\_left - 5)) {

steer\_ang = steer\_ang + 5;

steering.write(steer\_ang);

}

else if (ir\_ang - steer\_ang > 0 && steer\_ang < (max\_left - 1)) {

steer\_ang = steer\_ang + 1;

steering.write(steer\_ang);

}

}

//this code turns to the right

//the further the ir angle is from the currently set steering angle, the larger the turn increments are

if (ir\_ang < steer\_ang) {

if (steer\_ang - ir\_ang >= 30 && steer\_ang < (max\_right + 10)) {

steer\_ang = steer\_ang - 10;

steering.write(steer\_ang);

}

else if (steer\_ang - ir\_ang >= 15 && steer\_ang < (max\_right + 5)) {

steer\_ang = steer\_ang - 5;

steering.write(steer\_ang);

}

else if (steer\_ang - ir\_ang > 0 && steer\_ang < (max\_right + 1)) {

steer\_ang = steer\_ang - 1;

steering.write(steer\_ang);

}

}

}

else { //if using ir recievers

//this code turns to the left

//the further the ir angle is from the currently set steering angle, the larger the turn increments are

if (last\_rec\_ir\_ang > steer\_ang) {

if (last\_rec\_ir\_ang - steer\_ang >= 30 && steer\_ang < (max\_left - 10 )) {

steer\_ang = steer\_ang + 10;

steering.write(steer\_ang);

}

else if (last\_rec\_ir\_ang - steer\_ang >= 15 && steer\_ang < (max\_left - 5)) {

steer\_ang = steer\_ang + 5;

steering.write(steer\_ang);

}

else if (last\_rec\_ir\_ang - steer\_ang > 0 && steer\_ang < (max\_left - 1)) {

steer\_ang = steer\_ang + 1;

steering.write(steer\_ang);

}

}

//this code turns to the right

//the further the ir angle is from the currently set steering angle, the larger the turn increments are

if (last\_rec\_ir\_ang < steer\_ang) {

if (steer\_ang - last\_rec\_ir\_ang >= 30 && steer\_ang > (max\_right + 10)) {

steer\_ang = steer\_ang - 10;

steering.write(steer\_ang);

}

else if (steer\_ang - last\_rec\_ir\_ang >= 15 && steer\_ang > (max\_right + 5)) {

steer\_ang = steer\_ang - 5;

steering.write(steer\_ang);

}

else if (steer\_ang - last\_rec\_ir\_ang > 0 && steer\_ang > (max\_right + 1)) {

steer\_ang = steer\_ang - 1;

steering.write(steer\_ang);

}

}

}

//speed + range control begin here...

ir\_ypos = irCam.p1.y;

if (ir\_ypos > 1022) {

ir\_ypos = 0;

}

if (ir\_ypos > 0 && millis() - speed\_change\_time >= speed\_delay && brake\_complete == true) {

Serial.println("Speed loop started!");

if (ir\_ypos > 400) { //y > means trailbot is closer to target than desired

if (cur\_speed >= 1530) {

cur\_speed = cur\_speed - 5;

speed\_change\_time = millis();

}

esc.writeMicroseconds(cur\_speed);

Serial.print("IR Y coordinate is: ");

Serial.print(ir\_ypos);

Serial.print(". Decreasing speed to ");

Serial.print(cur\_speed);

Serial.println(" to open distance.");

}

else if (ir\_ypos < 350 ) { //y < means trailbot is farther from target than desired

if (cur\_speed <= 1600) {

cur\_speed = cur\_speed + 1;

speed\_change\_time = millis();

}

esc.writeMicroseconds(cur\_speed);

Serial.print("IR Y coordinate is: ");

Serial.print(ir\_ypos);

Serial.print(". Increasing speed to ");

Serial.print(cur\_speed);

Serial.println(" to close distance.");

}

else if (brake\_complete == true) {

esc.writeMicroseconds(cur\_speed);

Serial.println("IR Y coordinate within accepable bounds");

}

}

else {

esc.writeMicroseconds(cur\_speed);

}

}

else {

esc.writeMicroseconds(1500);

steering.write(center);

delay(10);

}

}

void echoCheck() { // If ping received, set the sensor distance to array.

if (sonar[currentSensor].check\_timer())

cm[currentSensor] = sonar[currentSensor].ping\_result / US\_ROUNDTRIP\_CM;

}

void oneSensorCycle() { // Sensor ping cycle complete, do something with the results.

// The following code would be replaced with your code that does something with the ping results.

// for (uint8\_t i = 0; i < SONAR\_NUM; i++) {

// Serial.print(i);

// Serial.print("=");

// Serial.print(cm[i]);

// Serial.print("cm ");

// }

// Serial.println();

last\_range\_us1 = cm[0];

last\_range\_us2 = cm[1];

last\_range\_us3 = cm[2];

//US1 Smoothing

if (last\_range\_us1 != 0) {

us1\_total = us1\_total - us1\_readings[us1\_readIndex];

us1\_readings[us1\_readIndex] = last\_range\_us1;

us1\_total = us1\_total + us1\_readings[us1\_readIndex];

us1\_readIndex = us1\_readIndex + 1;

if (us1\_readIndex >= numReadings) {

us1\_readIndex = 0;

}

range\_us1 = us1\_total / numReadings;

//Serial.print("Smoothed range us1: ");

//Serial.println(range\_us1);

}

//US2 Smoothing

if (last\_range\_us2 != 0) {

us2\_total = us2\_total - us2\_readings[us1\_readIndex];

us2\_readings[us2\_readIndex] = last\_range\_us2;

us2\_total = us2\_total + us2\_readings[us2\_readIndex];

us2\_readIndex = us2\_readIndex + 1;

if (us2\_readIndex >= numReadings) {

us2\_readIndex = 0;

}

range\_us2 = us2\_total / numReadings;

//Serial.print("Smoothed range us2: ");

//Serial.println(range\_us2);

}

//US3 Smoothing

if (last\_range\_us3 != 0) {

us3\_total = us3\_total - us3\_readings[us3\_readIndex];

us3\_readings[us3\_readIndex] = last\_range\_us3;

us3\_total = us3\_total + us3\_readings[us3\_readIndex];

us3\_readIndex = us3\_readIndex + 1;

if (us3\_readIndex >= numReadings) {

us3\_readIndex = 0;

}

range\_us3 = us3\_total / numReadings;

//Serial.print("Smoothed rangevus3: ");

//Serial.println(range\_us3);

}

//Serial.print("US1: ");

//Serial.print(range\_us1);

//Serial.print(", US2: ");

//Serial.print(range\_us2);

//Serial.print(", US3: ");

//Serial.println(range\_us3);

if (millis() - brake\_timer > brake\_delay) {

brake\_complete = true;

}

if (range\_us1 > 0 && range\_us1 < obstacle1) {

Serial.print("Obstacle center at ");

Serial.println(range\_us1);

esc.writeMicroseconds(1501);

esc.writeMicroseconds(1000);

brake\_complete = false;

}

else if (range\_us2 > 0 && range\_us2 < obstacle2) {

Serial.print("Obstacle left at ");

Serial.println(range\_us2);

esc.writeMicroseconds(1501);

esc.writeMicroseconds(1000);

brake\_complete = false;

}

else if (range\_us3 > 0 && range\_us3 < obstacle3) {

Serial.print("Obstacle on right at ");

Serial.println(range\_us3);

esc.writeMicroseconds(1501);

esc.writeMicroseconds(1000);

brake\_complete - false;

}

}