Race of Doom

Design Document

Team 43

Client: Timothy Bigelow

Advisor: Timothy Bigelow

Team Members/Roles:

Andrew Kraft - Testing, Circuit Design

Jack Doe - Project Manager

Gavin Petrak - Team Organization

Jacob Nedder - Testing, Team Coordination

Peter Wissman - Client Interaction

Team Email: sdmay24-43@iastate.edu

Team Website: https://sdmay24-43.sd.ece.iastate.edu

Introduction

Problem Statement

In pursuit of revolutionizing transportation, our project aims to advance the development of self-driving vehicles, primarily focusing on enhancing key functionalities such as crash detection, prevention, and autonomous vehicle control. By harnessing similar technologies and innovative methodologies, we aim to ensure that self-driving vehicles not only detect and respond to potential collisions with precision but also possess the ability to to proactively prevent accidents from occurring.

Furthermore, our project extends beyond typical road obstacles, highlighting vehicles' need to navigate diverse and dynamic road environments. Our team is committed to equipping an autonomous steering system with the intelligence and adaptability to traverse many challenges, including a plethora of unexpected obstructions and obstacles. Through intensive testing and refinement, our team aims to instill decision-making capabilities within the RC car, enabling it to navigate complex scenarios with efficiency and safety as its primary objective. With this commitment to safety, reliability, and adaptability, we strive to redefine the possibilities of autonomous driving and pave the way for a safer and more efficient transportation and landscape.

Intended Users and Uses

The intended users of this project primarily include individuals and organizations with a vested interest in the advancement of autonomous vehicle technology. Specifically, our project caters to faculty and future participants involved in the Race of Doom, offering them valuable insights and advancements in self-driving vehicle development that can be integrated and improved upon into future project iterations. By providing a refined design and innovative solutions, we aim to contribute to the ongoing evolution of autonomous vehicles within the Race of Doom project context.

Furthermore, our project serves the broader community invested in the growth of self-driving vehicles. As the autonomous vehicle industry grows, creating diverse test cases, such as those utilized in the Race of Doom, becomes increasingly valuable. Even on a smaller scale, our project provides a plethora of useful data and insights that can be leveraged for further research and use in future Race of Doom projects. Thus, while our project may be localized to the context of the Race of Doom, its implications extend further, contributing to the advancement and development of self-driving vehicle technology.

Related Products

The recent rise of Tesla’s self-driving cars is likely the most well-known example of a related product. These engineers are attempting to allow fully automatic driving in any real-world scenario. Although Tesla brings some of the most advanced self-driving technology to the table, there are still plenty of bugs that prevent drivers from completely relying on the functionality of this software. Although we did not have nearly as high a budget as a company like Tesla, we were still able to work off of their previous knowledge and experience.

If we were to scale down the high aspirations of Tesla, another example of a group attempting to develop autonomous vehicles could be teams from the show BattleBots. Here, groups compete against each other to use sensors and traps to fight their opposing vehicle. Our project is similar to this example where we must race around a track as fast as possible. We could use ideas from a show like BattleBots and transform them into our designs for a race. However, we must also keep in mind budget is a large factor for a project like this, and we likely do not have the same funding as groups in the BattleBots show.

Revised Design

Requirements & Constraints

Team Requirements:

- Weekly meetings with the individual team, and periodic meetings between all Race of Doom teams.
- Modify an existing car with new specs so every car team starts at the same point.
- A functional car must be fully operational by the end of Semester 2.

Design Requirements:

- The project must stay within the given CPR E program's budget
- The RC Car shall autonomously steer away from obstacles on the track
- The RC Car shall stay within the bounds of the track
- The RC Car shall be protected from the track-hacking source
- The Driver shall control only the speed and acceleration of the RC Car
- The RC Car shall make multiple laps around the track

Constraints:

- Our car must cost less than $500 to build, modify, and develop. This is subject to change pending funding from Caterpillar.
- The car should be able to sense its environment using sensors installed on the vehicle.
- Cars should be somewhat autonomous: the car can only move forward or backward by user input, and steering will be determined by sensors on the car.
- Car communications security must be student-built to allow testing of the cyber security aspect of the project.
- Each obstacle for the car should be overcome as quickly as possible to win the race.

Engineering Standards

- IEEE 802.11 (Wi-Fi) Standards: IEEE 802.11 standards for Wi-Fi communication are relevant for remote control and data exchange between the remote control devices and the cars.
- ISO 6469 - Safety of Electrically Propelled Road Vehicles: While the vehicles are small in scale if they are electrically powered, ISO 6469 might still apply to address electrical safety aspects, especially if they use lithium-ion batteries or other electrical components.
- IEEE 1275 - Open Firmware Standard for Embedded Systems: If the remote control cars use embedded systems or microcontrollers, adherence to relevant firmware standards can be important for compatibility and reliable operation.

- Radio Frequency (RF) Standards: Depending on the communication technology used for remote control, there may be specific RF standards that apply to ensure proper signal transmission and interference avoidance.
- Electromagnetic Compatibility (EMC) Standards: EMC standards can be relevant to ensure that the operation of the remote control cars does not interfere with other electronic devices and vice versa.

Security Considerations

For the project, security issues played a small but significant role. While this vehicle did not handle sensitive data like finances or personal information, one of the core components of the Race of Doom project definition was the inclusion of elements in the track that attempted to mislead or otherwise challenge the cars, from the Faraday cage with hanging material that must be driven through despite registering as a wall to the ramp, which again registers the same sensory information as a wall, to a reflective metal plate in the track designed to disorient floor-based sensors.

To be effective, the team tested extreme cases to ensure the vehicle could still function. As an example, the LiDAR sensor was altered to be able to filter out false positives potentially caused by more reflective surfaces or conflicting artificial signals from draping material. Another challenge is ensuring control over the vehicle's acceleration is maintained in the presence of difficulties in track terrain such as bubbles and ledges. Finally, tests for the vehicle's ability to navigate in unusual terrain will be of utmost importance. The track will not be straight, so being able to turn corners, navigate enclosed passages, or come to a premature halt with unexpected timing will all be necessary.

Design Evolution

Over time developing this project and making our vision a reality, we encountered a few issues and hardships that forced us to adjust our design. First and foremost, the unexpectedly small size of our car called for us to take off the outer shell of the car and replace it with a platform that mounts our sensors. This platform was also important for us to include an external battery which was needed to run the Raspberry Pi. Due to evolving constraints and the discovery that the track team was going to use walls instead of tape for the boundaries, we scrapped the use of the photoelectric sensor, as the LiDAR sensor gave enough data for every problem.

Design Implementation

Detailed Design - Overview

The overall design had three main components to it: the hardware, software, and integration portions. The hardware focused on the connections between the physical components and software devices. The software focused on controlling the steering with the Raspberry Pi, which controlled the input and output data between the LiDAR and the steering motor. Finally, the integration portion connects

all the different hardware and software components together with a mount fastened to the top of the RC car. Figure 1 shows the complete design.

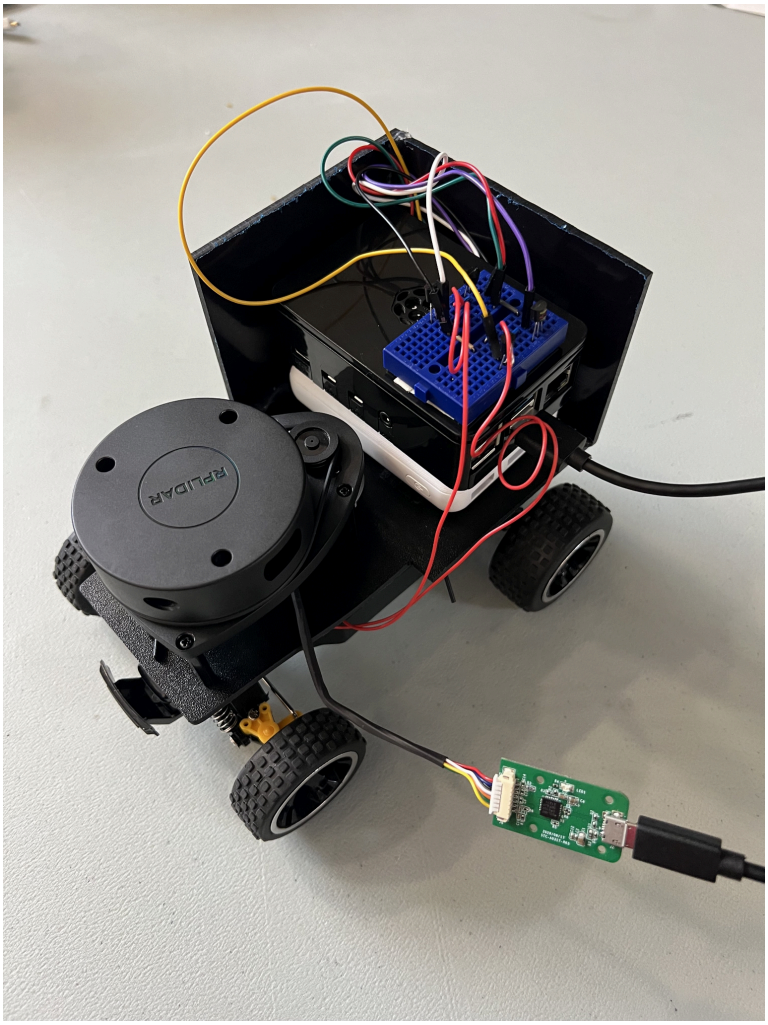


Figure 1: Full RC Car Design

Detailed Design - Hardware

The design hardware consisted of a Raspberry Pi 4, added battery pack, simple H-Bridge circuit, a LiDAR, and a mount to hold it all together. The added battery pack is to power the Raspberry Pi, which is then connected to the LiDAR through a USB port, and the H-Bridge circuit through the GPIO pins, as shown in the power systems diagram in Figure 2. All of these added devices are held together by the mount, more details about this in the "***Detailed Design - Integration***" section.

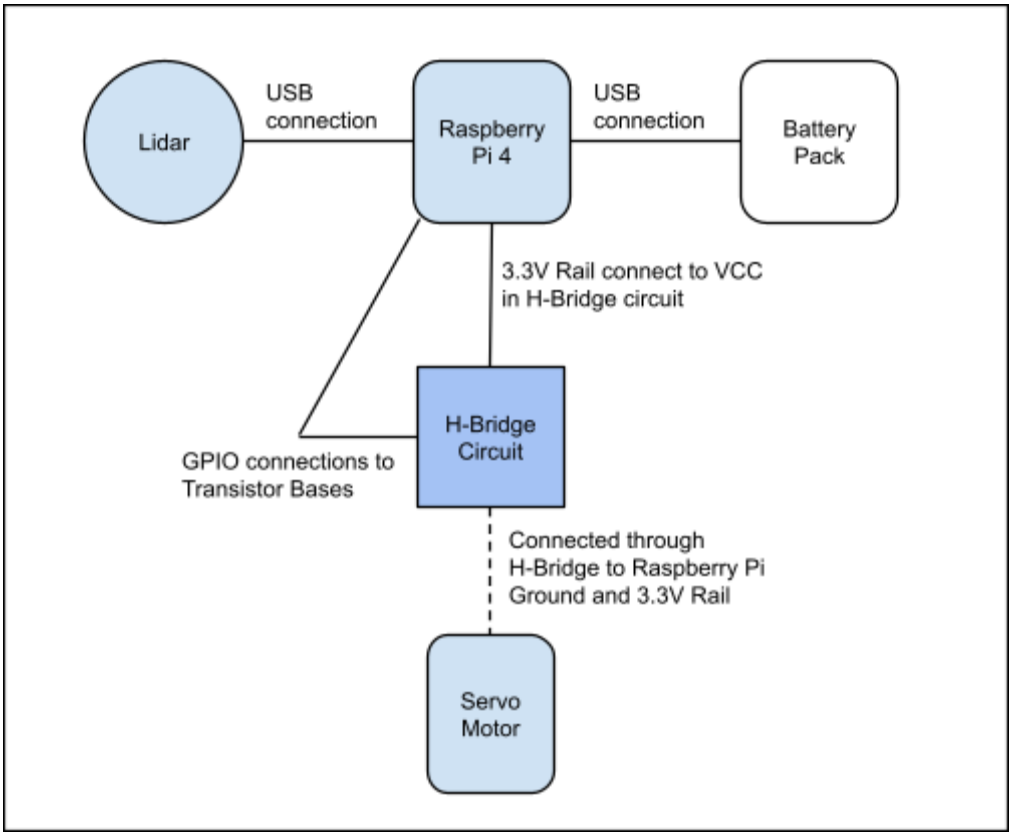
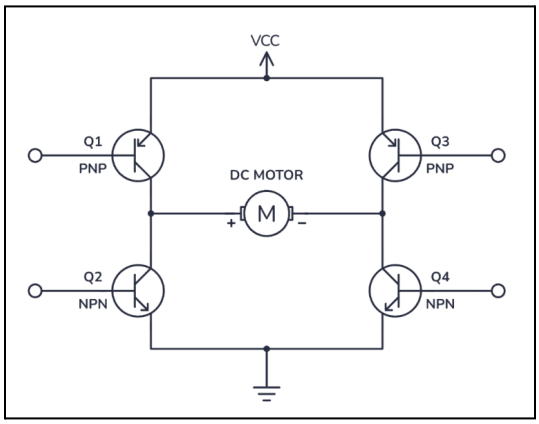
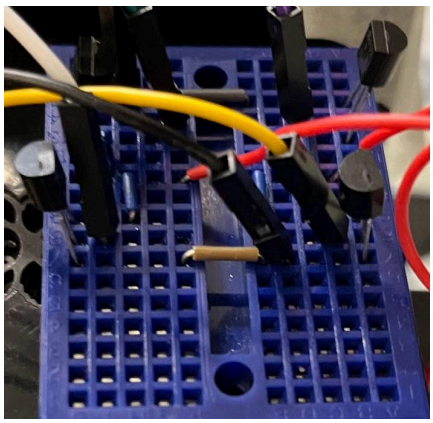


Figure 2: Power Systems Diagram

Through various testing, more details in "***Testing - Hardware***" section, we learned that an H-Bridge circuit would be best suited for our RC car because it connects the already installed Servo motor used for turning, to the Raspberry Pi GPIO pins. This circuit configuration would essentially allow power to the servo motor by turning on and off different GPIO pins on the Raspberry Pi, allowing easy control of the steering. The H-Bridge implemented consists of two 2222a NPN and two PN2907 PNP BJT's connected to the servo motor as shown in the circuit diagram in Figure 3. These particular BJT's had a low on/off power requirement allowing direct control using the Raspberry Pi GPIO pins. The turning configurations using the specific GPIO pins can be seen in the table.





	GPIO/BCM Pin	Turn Left	Turn Right	Straight
Q1 - PNP	13/27	1 - Off	0 - On	0
Q3 - PNP	33/13	0 - On	1 - Off	0
Q2 - NPN	35/19	0 - Off	1 - On	0
Q4 - NPN	11/17	1 - On	0 - Off	0

Figure 3: H-Bridge circuit and corresponding GPIO connections to Raspberry Pi

The GPIO pins produce 3.3V at ~16mA which was above the 15mA requirement for the PNP transistors and the 10mA NPN requirement. Having the power requirements met for both transistors saved the need to supplement power from other sources on the Raspberry Pi, increasing the overall battery time of the extra battery pack.

The connections made from the H-Bridge to the servo motor were connected directly by soldering wires to the already in place connections on the servo motor, as shown in Figure 4. This allowed for a stable connection that eliminates the chance of the wires disconnecting when in use. All the circuitry was connected through a small breadboard.

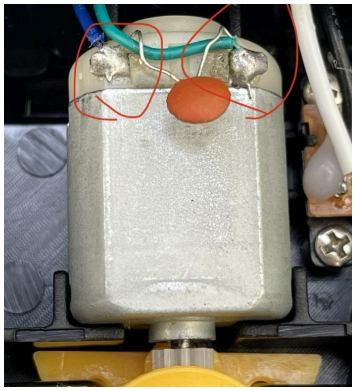


Figure 4: Initial Connections on Servo Motor

The design also consisted of changes relating to the suspension. The original suspension was not able to withstand the added weight of the Raspberry Pi 4, battery pack, simple H-Bridge circuit, LiDAR, and mount. To solve this, we glued small pieces of plastic to prevent the front axle from dipping down, as shown in Figure 5.

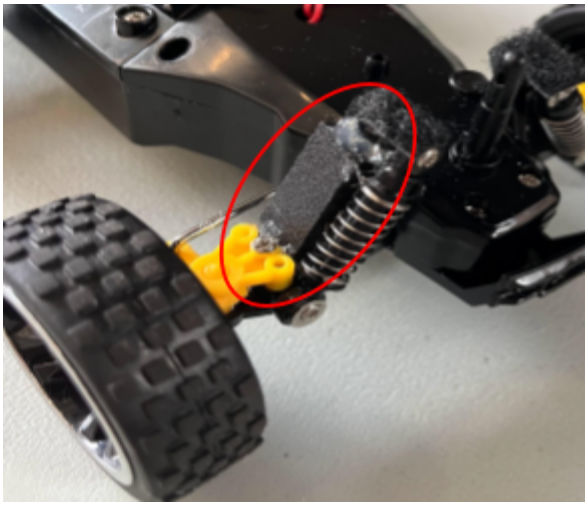
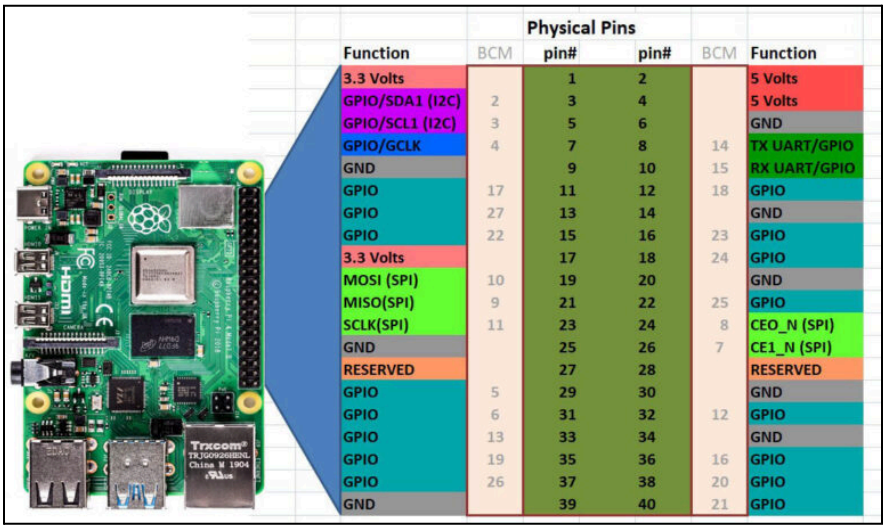


Figure 5: Added Suspension

The motor used to control the forward and backward motion of the RC car was not changed because the Criteria/Requirement specified that the user could control that. The connection between the RC car controller and the already installed circuit board was also not changed. This is because the “hacking” portion of the contest was dropped due to the questionable legality of remotely hacking into devices, so no change was needed.

Detailed Design - Software

The original plan for the software was to use Python on the Raspberry Pi to control the sensors, but the libraries for the LiDAR sensor were native to C++. Initially, the GPIO ports attached to the LiDAR were going to be used to figure out how to get information at a very low level. The pinout definitions in Figure 6 were used for the Raspberry Pi.



		Physical Pins			
Function	BCM	pin#	pin#	BCM	Function
3.3 Volts		1	2		5 Volts
GPIO/SDA1 (I2C)	2	3	4		5 Volts
GPIO/SCL1 (I2C)	3	5	6		GND
GPIO/GCLK	4	7	8	14	TX UART/GPIO
GND		9	10	15	RX UART/GPIO
GPIO	17	11	12	18	GPIO
GPIO	27	13	14		GND
GPIO	22	15	16	23	GPIO
3.3 Volts		17	18	24	GPIO
MOSI (SPI)	10	19	20		GND
MISO(SPI)	9	21	22	25	GPIO
SCLK(SPI)	11	23	24	8	CEO_N (SPI)
GND		25	26	7	CE1_N (SPI)
RESERVED		27	28		RESERVED
GPIO	5	29	30		GND
GPIO	6	31	32	12	GPIO
GPIO	13	33	34		GND
GPIO	19	35	36	16	GPIO
GPIO	26	37	38	20	GPIO
GND		39	40	21	GPIO

Figure 6: Raspberry Pi 4 Pinout

However, it was discovered that there was a missing USB daughter board connected to the LiDAR that was necessary to gain access to the LiDAR's header files. This was a crucial piece of the puzzle as it had necessary functions dealing with scanning processes, which would have been out of the scope of our project if we needed to design them. Instead, along with this small USB daughterboard, we found an SDK that supplied a few example programs to understand the LiDAR's output data, and decided to build directly off these example programs to run the car.

It was important to relay information between the software and hardware sides when dealing with the circuitry that determined the car's steering. Maintaining the same GPIO ports throughout the project was crucial so the code would work reliably. Additionally, developing software for these GPIO ports was much easier now that we were working with C++ instead of Python.

Inside the code, a while loop was created which ran until we decided to end the program. This loop would be where the logic for steering would be located, and would use the data sets the LiDAR would provide as input data. Our team decided that assigning weights based on obstacle distances, location, and angle would give the best decision-making process. The most critical of these ranges were 20 degrees to the left and right of the front of the vehicle, and these ranges determined if the car needed to steer. Next, was the creation of many fields which were chosen by meeting criteria based on priority. For instance, if the LiDAR scans and hits something on the right, the car will turn to the left. Adjusting these variables was key to creating a cohesive system for autonomous steering. An example of this can be found in Figure 7.

```
        if (SL_IS_OK(op_result)) {
            drv->ascendScanData(nodes, count);
          int rightWeight = 0;
          int leftWeight = 0;
          int rampWeight = 0;
            for (int pos = 0; pos < (int)count ; ++pos) {
            if (((nodes[pos].dist_mm_q2/4.0f) < 500) && (nodes[pos].quality
!= 0) && (((nodes[pos].angle_z_q14 * 90.f) / 16384.f < 60) ||
((nodes[pos].angle_z_q14 * 90.f) / 16384.f > 300))) {

                // Printing scans where the result is valid, less than 0.5m
away, and in the 40 degree range in front of the car.

                if ((((nodes[pos].angle_z_q14 * 90.f) / 16384.f) > 340)) {
                  // Critical wall angle to the right.

                  if (((nodes[pos].dist_mm_q2/4.0f) <= 200)) {
              // Check if we are very near a wall to turn into.
                      leftWeight--;
                  } else {
                      leftWeight++;
                  }

                } else if (((nodes[pos].angle_z_q14 * 90.f) / 16384.f) <
20) {       // Critical wall angle to the left.

                  if (((nodes[pos].dist_mm_q2/4.0f) <= 200)) {
              // Check if we are very near a wall to turn into.
                      rightWeight--;
                  } else {
                      rightWeight++;
                  }
```

Figure 7: Example code of finding objects in the path of the RC car

Finally, depending on if the system decides to steer, signals were sent to our GPIO ports that control the steering circuit. Unfortunately with the simplistic nature of the original RC car, it could only turn

fully to the left or right based on the inputs to the servo motor. Figure 8 shows an example of the GPIO/BCM output to control the steering.

```
        if ((abs(leftWeight - rightWeight) <= 10) && (rightWeight != 0)
&& (leftWeight != 0)) {
            // If there are walls on both sides...
            // continue straight
            gpioWrite(17, 0);
            gpioWrite(27, 0);
            gpioWrite(13, 0);
            gpioWrite(19, 0);
            printf("Walls on both sides... stay straight\n");

          } else if (rightWeight > leftWeight) {
            // If there is much more wall presence on the right...
            // turn left
            gpioWrite(17, 0);
            gpioWrite(27, 1);
            gpioWrite(13, 0);
            gpioWrite(19, 1);
            printf("Turning Left - right: %d - left: %d \n", rightWeight,
leftWeight);

          } else if (leftWeight > rightWeight) {
            // If there is much more wall presence on the left...
            // turn right
            gpioWrite(17, 1);
            gpioWrite(27, 0);
            gpioWrite(13, 1);
            gpioWrite(19, 0);
            printf("Turning Right - right: %d - left: %d \n", rightWeight,
leftWeight);
```

Figure 8: Example code of assigning GPIO signals for straight movement, left, and right turns

Detailed Design - Integration

The final part of the design was combining the hardware and software components. The two integral parts that needed to be connected are the following:

1. Connecting the Raspberry Pi pins to the Breadboard.
2. Attaching the Raspberry Pi, Portable Battery, and the LiDAR to the mounted plastic on our RC car.

For the Breadboard, we needed to assign pins on our Raspberry Pi that we will use to supply power to our H-Bridge circuit. Connections were made with simple male to female wires that would connect the specified pin on the Raspberry Pi to the correct location on the breadboard. This would allow the software to control the simple DC motor that was used for steering the vehicle. More specifications on the H-Bridge circuit from Figure 9 can be found in "***Detailed Design - Hardware***" section of this design document.

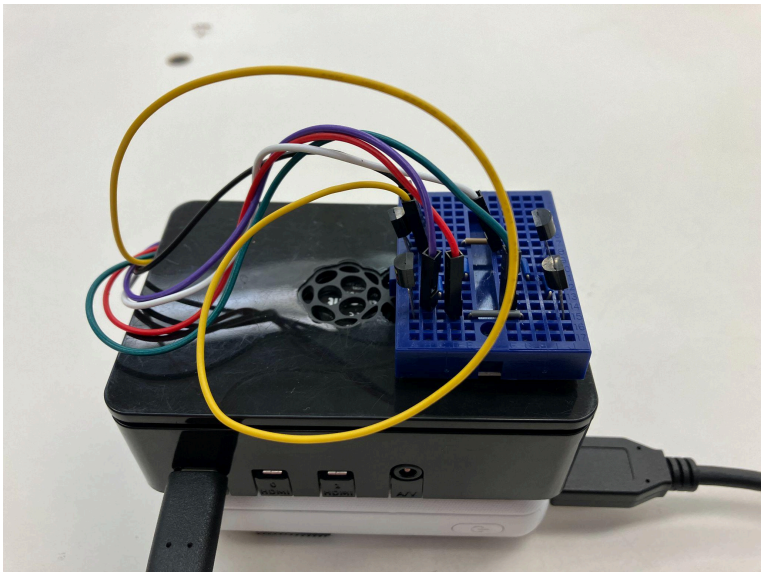


Figure 9: H-Bridge Circuit

In order to mount all the pieces of our design together, we opted to buy a 1⁄8 inch thick sheet of ABS Plastic. After taking measurements of each major component that needed to be mounted: the LiDAR, raspberry pi, and portable battery, we mapped out the locations on the plastic sheet of where everything was going to go. Afterwards, a section of the plastic was cut to fit on top of the vehicle. Along with this, walls were cut and attached using glue to provide some extra protection and stability for the battery and Raspberry Pi, as shown in Figure 10.

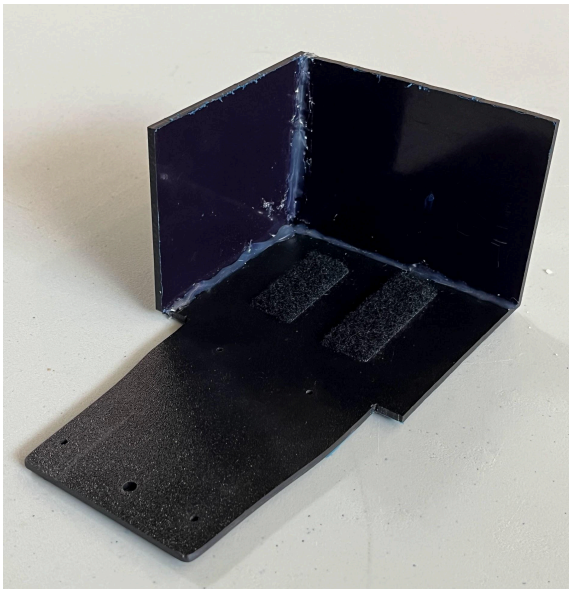


Figure 10: Mount for the components

Once the plastic was cut, we needed to choose how to attach the plastic to the car as well as attach the components to the plastic. Because of the smaller nature of the RC car, we didn't have a lot of options when it came to mounting the plastic. The final mounting structure ended up being connected to the suspension supports and the middle portion of the RC car, as shown in Figure 11. In order to securely

fasten the plastic to the vehicle, velcro was used on both the car and the plastic. A hole was also drilled into the plastic that fit around a vertical section rc car that was sticking out of the top. This helps prevent the plastic from sliding forward off the RC car and helps with keeping the mount fastened securely.

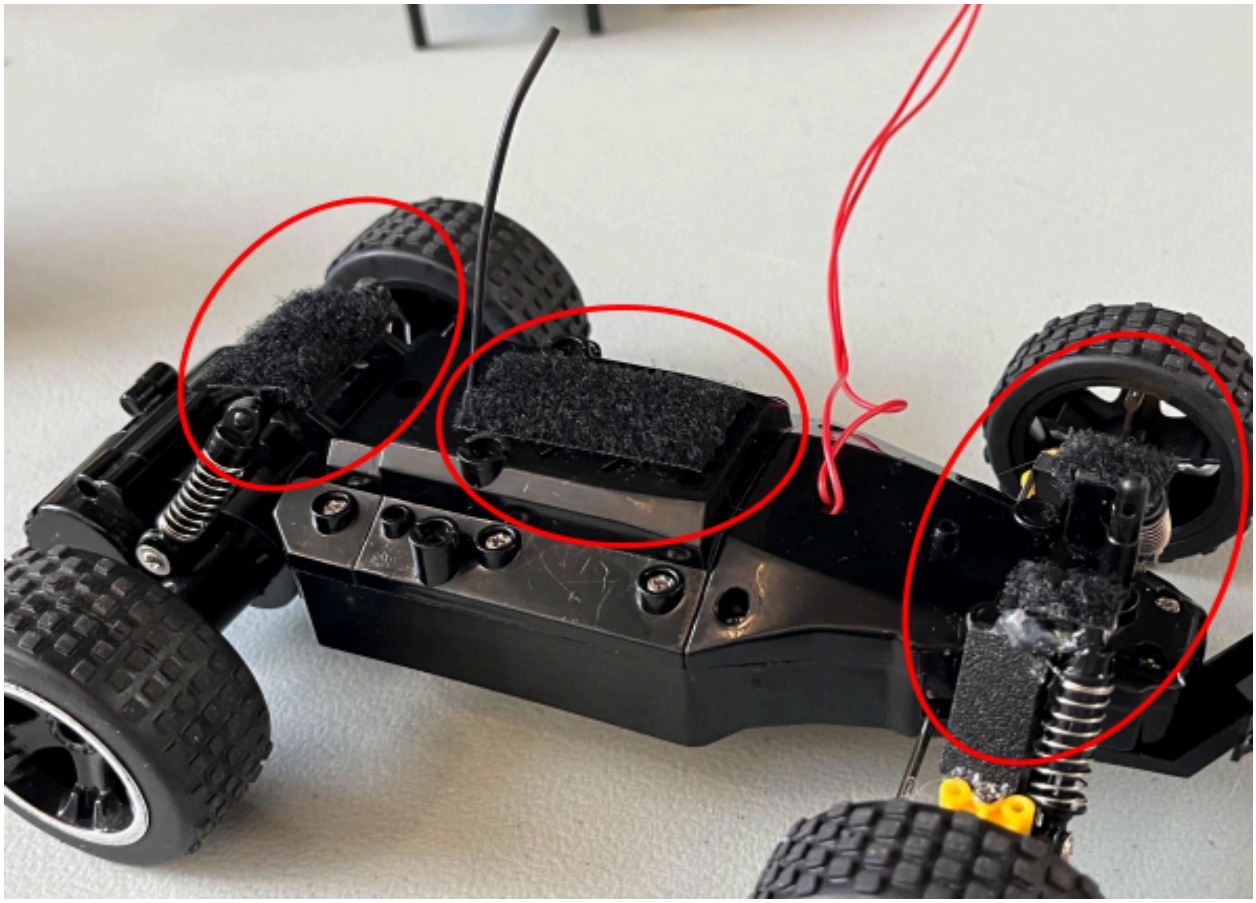


Figure 11: Velcro connections for the mount

Attaching the components to the plastic was slightly different but more of the same. Velcro was again used to attach the Raspberry Pi and battery together, as well as the battery to the base of the mount, as shown in Figure 12. This was found to be the easiest method, as drilling holes into the battery or Raspberry Pi was impossible. Instead of glue, velcro allowed for easy disassembly to modify the internal components separately.

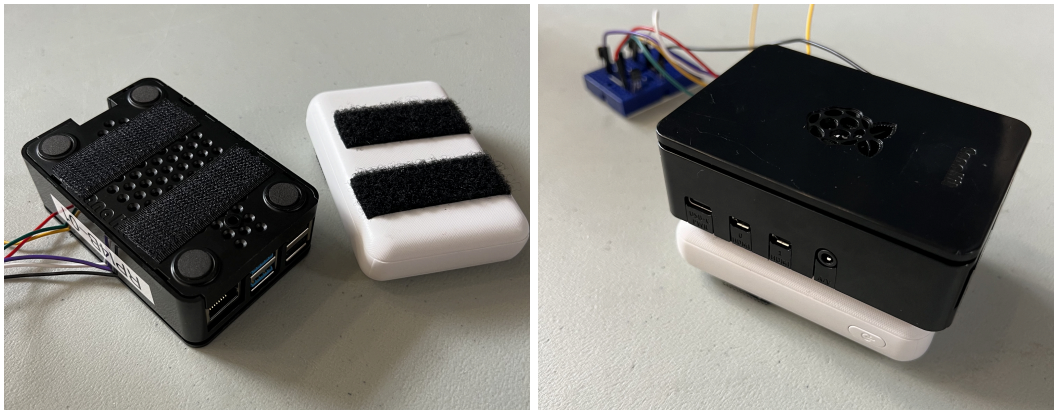


Figure 12: Velcro to attach Raspberry Pi to battery

The last component, the LiDAR, required a different mounting process. Because of the cost of the LiDAR, it's best to securely fasten it to the mount. The LiDAR happened to have a built-in stand which had holes we could use to screw into. So it was decided to drill holes into the mount in order to securely fasten the LiDAR to the mount. The LiDAR fastened to mount is shown in Figure 13.

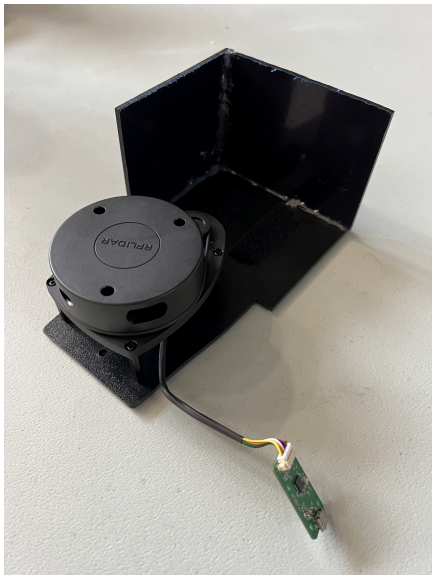


Figure 13: LiDAR fastened on the Mount

Overall, the mounting of the components works surprisingly well. The only downside is the plastic is a little shaky due to the mounting points on the car being too close to the center of gravity. This fact was unavoidable in our approach to mounting as the car is just too small to mount it anywhere else. Figure 14 shows all components fully mounted.

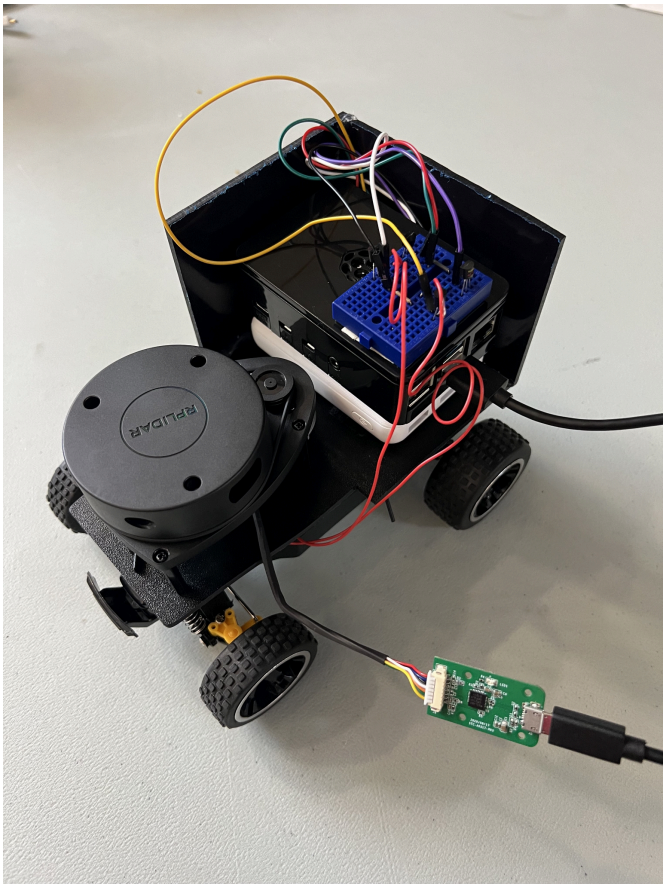


Figure 14: All components fully mounts on the RC car

Testing

Testing - Hardware

Process:

The hardware portion of the final design consisted of various physical components installed, and an H-Bridge circuit connected to different pins on the Raspberry Pi and Servo motor. Testing all of these components was done alongside the design process and implementation stages of our work.

For all physical components made (suspension upgrades, and mount with all devices, etc), testing was mainly done by applying different degrees of force/weight relative to what the RC car may experience when being used on the track. There was no special method for this; we applied force through hands and various objects, such as a notebook lightly hitting the RC car, to simulate this. Testing was done this way because this iteration of our RC car will be exclusively used in a semi-controlled environment, and there is a very low chance of large amounts of force being applied directly to the RC car. Various other tests were done if a problem arose, more information listed in the “Results” section.

To implement/verify that the H-Bridge circuit worked properly the following steps were taken:

1. Test what input voltage and current was needed to activate the servo motor, and in what connection. Use a multimeter and voltmeter in tandem with varying voltages and currents.

    - A minimum of 3V at 300mA is needed to activate the Servo to be able to turn. Only two wires are connected to the servo, and that one side needs to be grounded to turn. From this, and some research, it is concluded that an H-Bridge circuit configuration is needed, since it would allow us to control the base of the transistors with the GPIO pins from the Raspberry Pi and ground the circuit as needed.

2. To test the H-Bridge circuit, a voltmeter acted as the batteries and GPIO pins, while LED's acted as the motor for turning left and right. The volt meter was set to the same power requirements stated on the battery packs (3.3V at 2A) that came with the car, and the GPIO pins (3.3V at ~16mA) as stated on the datasheet referenced for the Raspberry Pi.

3. Verifying that this connection worked for our purposes, the GPIO pins, servo motor, and original batteries went back into the circuit and once again tested to confirm that everything worked properly.

    - After these connections were verified, soldering was done to connect wires to the Servo motor so connections inside the RC car were stable.

Finally, when testing the final product, we tested the maximum speed we were able to achieve, the total weight and turning capabilities.

Results:

Various Tests Done:

	Testing needed	Conclusions/Results
Suspension upgrades	1) Effects on suspension with added component and mount weight	1) Suspension dipping in the front with weight applied Solution: Used plastic to prevent wheels from dipping when final product was in use
Mount	1) Total amount of force that can be applied when mount glued together 2) Physical effects mount may have has on other components	1) All pieces are securely connected. No change needed. 2) Wheels sometimes hit the edge of the mount when fully turned Solution: Used a file to shave down edges to prevent contact
Power requirements	1)Servo motor in H-Bridge circuit 2) Transistors in H-Bridge circuit 3) Final Design of RC car	1) Servo takes 3V-5V at 500mA to fully turn. Need a H-Bridge connection, with one side grounded Solution: Used Raspberry Pi 3.3V pin

		connected in H-Bridge with a ground pin 2) All transistors turn on/off with GPIO pins connected to the base. No change needed 3) All devices work together when both batteries are on. No change needed
Soldering	1) Test if connections will stay when in use	1) No connection lost. No change needed

Final Results Before and After Modifications added:

Before Results	After Results
Top Speed:1.5 m/sec	Top Speed: 0.78 m/sec
Turning Radius: 1.143 m	Turning Radius: 1.27 m
Total Weight: 689.46 grams	Total Weight: 1176.77 grams

Testing - Software

The testing portion for software involved communication and control of GPIO ports as well as gathering data from the LiDAR sensor and using it to detect the environment around the system. Testing the software for autonomous steering recognition came with much trial and error, which brought continuous improvement with every tweak added to the code.

Firstly, the LiDAR was tested for its range, accuracy, and reliability. Boards or hands were used to act as the obstacles that would present themselves on the track. A variety of situations were handled, such as detecting walls only to the left or right, as well as cases where walls are present on both sides of the car. Each of these scenarios would send text output to the terminal, which is where the LiDAR's detection behavior could be observed. Based on this output, adjustments were made to the software's weights, tolerances, and ranges.

Next came the steering mechanism, which was implemented in the code as soon as the circuit was completed. Inside each software request to steer, GPIO ports were asserted which sent the correct power to the hardware. During this time, the circuit started to get overwhelmed due to the amount of times these GPIO port assertions were happening. To resolve this, a sleep function was added, which limited the LiDAR scans to happen five times every second. Driving the car now needed to be slower to assess its environment properly, but this was much more important than attempting to code a broken steering circuit.

The most difficult part of testing came with recognizing the properties of the ramp. Due to the reliance on the LiDAR being enough to overcome all obstacles, this part of the project proved tricky to implement in software. In the end, it was decided to scan for a wall directly in front of the vehicle, and if

the distance from each end of this scanned “wall” was just about equal, the car determined that it was going straight toward the ramp and steering was shut off. This behavior continued for as long as the LiDAR scanned no walls around it, as that would imply it was above the track and on the ramp.

Broader Context

Area	Description	Examples
Public health, safety, and welfare	In our evolving society, self-driving cars and autonomous vehicles are becoming a reality more and more every day. With this emerging technology, creating additional test situations such as this senior design project can help find reliable solutions for public safety and welfare by minimizing hacking attempts and malfunctions.	Further developing knowledge in the world of automotive vehicles, reducing risks of malfunction and/or hacking from third parties.
Global, cultural, and social	At the moment, self-driving vehicles and the technology behind them are only available to a select few, such as the upper class and possibly farmers when it comes to automated farming.	Development of this test is a smaller scale to reduce risks of harming humans, animals, or the environment.
Environmental	The project will not use any gases or fossil fuels in general and will be run 100% on electricity. Utilizing the least amount of energy for optimization is a priority.	No usage of fossil fuels, and minimizing the amount of electricity used when the car is running is essential.
Economic	Even on a small scale, it is likely this experiment will cost at least $1000. Reasons like this are why automated vehicles are still in their infancy. Trying to cut down on these costs as much as possible without sacrificing safety features is the biggest goal of the project	Minimizing cost allows for further development of automated vehicles heading in a consumer-level market, but all safety requirements must still be met.

Cost Effectiveness

The total cost of the design was ~$217.39, as shown in Figure 15. Comparing this to the total cost of the RC car designed by Sdmay24-06, our design was $347.18 cheaper. This is maily due to our base RC car is 10x cheaper ($20 compared to $200), while our sensor was more expensive.

Component	Description (Taken from website where it was bought)	Cost (Website bought from)
Raspberry Pi 4	RASPBERRY PI 4 B 2GB	$45.00 (From ETG)

Tecnock RC Racing Car	2.4GHz High Speed Remote Control Car, 1:18 2WD Toy Cars Buggy for Boys & Girls with Two Rechargeable Batteries for Car, Gifts for Kids (White)	$19.99 (Amazon)
USB Battery Pack for Raspberry Pi	10000mAh - 2 x 5V outputs	$39.95 (Adafruit)
RPLiDAR	A1M8 2D 360 Degree 12 Meters Scanning Radius LiDAR Sensor Scanner for Obstacle Avoidance and Navigation of Robot	$99.99 (Amazon)
ABS Sheet	1/8 Inch Thick (3mm) - 12" x 8", Black Plastic Sheet Waterproof Rigid Thermoplastic Sheet, ABS Plastic Board for Sign, Craft, DIY Display Project (Pack of 1)	$5.97 (Amazon)
HiLetgo Mini Breadboard	HiLetgo 6pcs SYB-170 Mini Breadboard Colorful Breadboard Small Plates	$6.49 (ETG)
Circuit components		FREE (Students Own)
PN2907 PNP x2		
2222a NPN x2		
Male to Female Wires x6		
Normal Wires x6		
Physical Components		FREE (Students or ETG provided)
Glue		
1/4th screws x4		
Velcro		
Wire Connections		FREE (Students own or ETG provided or came with product)
USB A to 3.1C		
Micro USB to 3.1A		

Total		$217.39

Figure 15: Total Costs

Since the actual racing day is on (4/28/2024), one day after the Final Design Document is due, we are unable to come to solid conclusions/ranking in the last 24 hours, but based on what we have seen during the group testing date (4/27/2024) we can predict what might happen. Both teams have their pros and cons, as seen in Figure 16.

Sdmay24-43	Sdmay24-06
Pros: - Staying centered on the track - Able to drive through Faraday Cage - Able to consistently go around curves in track	Pros: - Able to go up/down ramp - Overall Faster speed - Easier access to internal structure/devices
Cons: - Slower and weaker motor - Unable to go up the ramp - Sometimes gets stuck going over bumps in track	Cons: - Unable to consistently go through Faraday Cage - Frequently runs into walls

Figure 16: Pros and Cons of both Designs

We predict that the overall times will be similar for both teams based on the Scoring Table in Figure 17. This is because both teams have a high chance of getting a Violation resulting to a Penalty. However, we believe that our design will score better because we have getting significantly less violations in the test runs.

Violation	*Penalty*
Touch or hitting the walls/objects	+5 sec
Knocking Down walls/objects	+10 sec
Minor Car/Track Adjustment	+20 sec
Major Car/Track Adjustment	+30 sec

Figure 17: Violation and Penalty table

Conclusion

Review Progress

There was much progress made this and last semester regarding our design. Starting with the previous semester, we began with an initial design of our car utilizing an unspecified RC car, a LiDAR Sensor, Photoelectric sensors, and an arduino for processing the autonomous functionality.

After deciding on our RC car model, we ordered parts for our initial design and changed the Arduino to a Raspberry Pi for more processing power. Furthermore, most of the first semester was spent planning our design, given the constraints and openness of the design process.

In the second semester of design, the software side of the team decided to implement our code for the LiDAR detection in C++ as the SDK for the LiDAR was in that language. We determined that the photoelectric sensors were redundant as the LiDAR did all our object detection sufficiently and more effectively. The hardware side of the team created and tested the H-Bridge circuit that was used to steer the RC car. They also built a chassis for all of the components as the size of the components was much larger than anticipated. For example, the external battery pack we purchased, the Raspberry Pi, breadboard, LiDAR sensor, and all the cables with the components needed to be mounted on the car.

After all the components were situated on the vehicle and working, we did integration testing of our design to make sure the car could move and detect objects properly. After that step was completed, the final demo on the Race Of Doom track was the last part of the project that needed to be completed.

Value of our Design

The design proved effective and functioned well, showcasing the integration of the various components, such as the LiDAR and Raspberry Pi. However, a significant challenge arose due to the size of the RC car, which hindered the car's ability to drive effectively and mount all of the components securely. This shows the importance of considering not only the high-cost components like the LiDAR and Raspberry Pi but also the function and size of the RC car.

The shape and size of the car play a pivotal role in its overall performance and functionality. A compact and appropriately sized design ensures optimal maneuverability and stability, leading to smoother operation and better integration of the components. In contrast, a bulky or small RC car can lead to difficulties in mounting essential components securely and may compromise the RC car’s ability to navigate the track effectively.

Therefore, when developing and designing autonomous RC car projects like Race of Doom, it is important to prioritize an optimal-sized RC car alongside the selection of components. This approach ensures that the design not only incorporates the right technologies but also optimizes the physical attributes of the car for enhanced performance and functionality. By carefully considering both factors, future iterations of the Race of Doom project can mitigate challenges related to size and component constraints, leading to a more successful autonomous RC car.

Next Steps for Race Of Doom

This was the first year that Race of Doom was available for seniors to choose as a project for Senior Design. Race of Doom aims to allow students to construct an RC car capable of autonomous operation. This will give the software students a challenging and engaging autonomy project while allowing electrical students to tackle the complexities of an embedded systems project. Since this was the first Race of Doom project ever held, the direction and requirements for the project were very loose and unspecified.

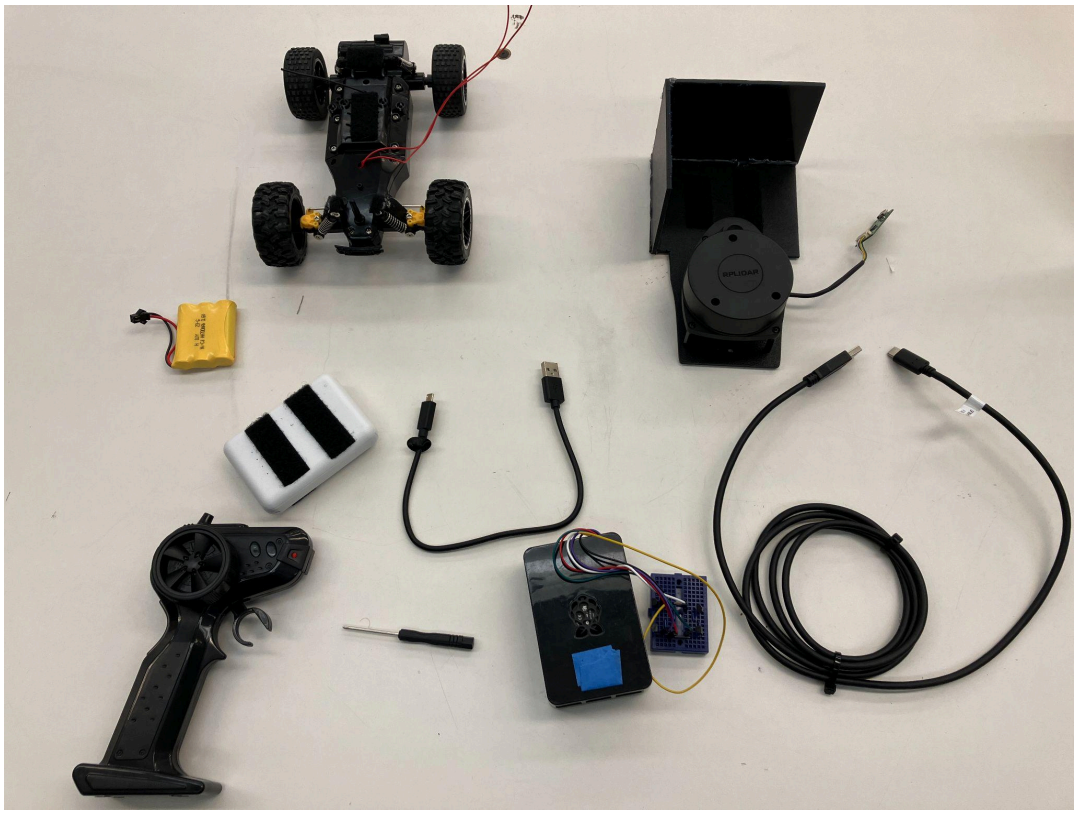
Future iterations of Race of Doom would allow other groups to look at our design and either enhance them or create their own vehicle based off of them. With access to our design, future teams would be able to avoid design flaws that our vehicle suffered from. Year after year, the designs for Race of Doom would be refined and would expand the requirements and specifications for each team to follow.

Appendices

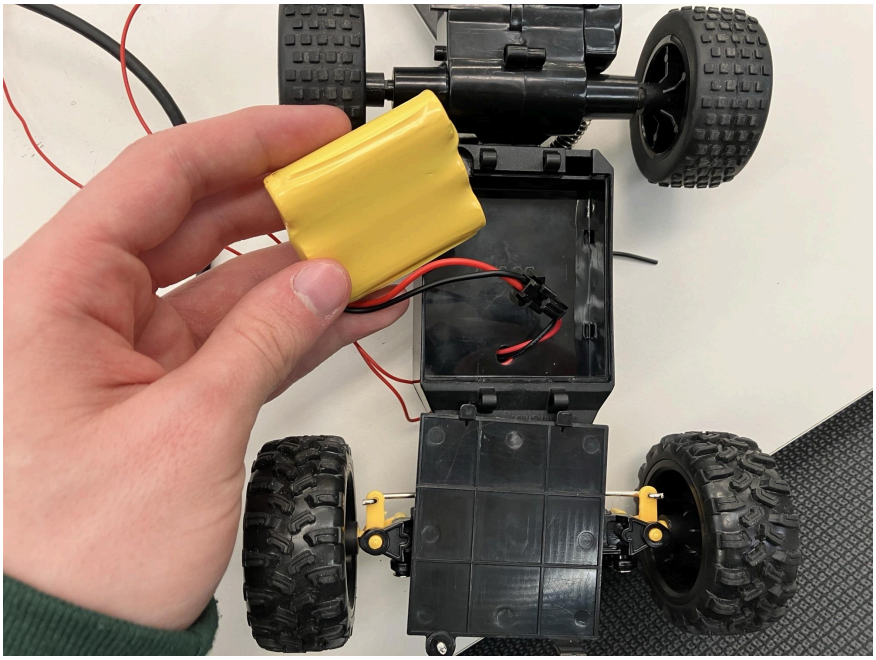
Appendix 1 – Operation Manual

Pictures correspond to Steps they are under

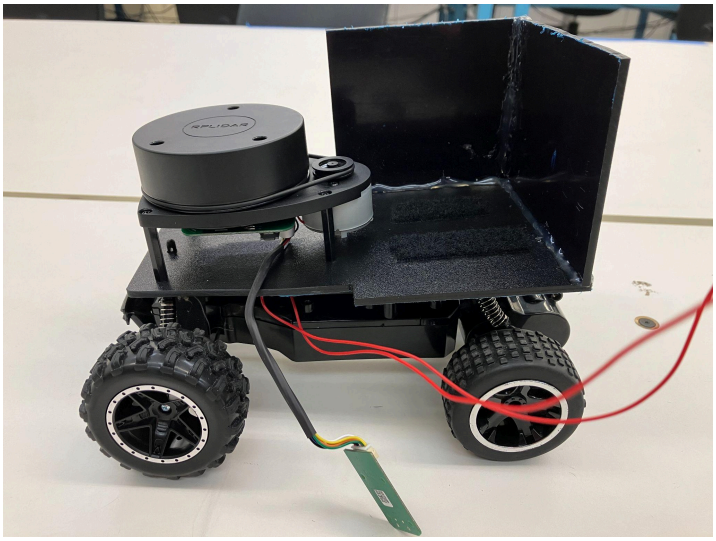
1. The car is stored disassembled for ease of transportation. First, gather all components and tools:
    a. Raspberry Pi (in case) with attached breadboard and wiring
    b. White external battery pack
    c. LiDAR sensor screwed into black plastic housing
    d. RC car body with attached red wires
    e. Yellow RC car battery pack
    f. USB to USB-C cable
    g. USB to micro USB cable
    h. RC car remote control
    i. Laptop (necessary for forming an ssh connection with the Raspberry Pi)
    j. Small Phillips head screwdriver



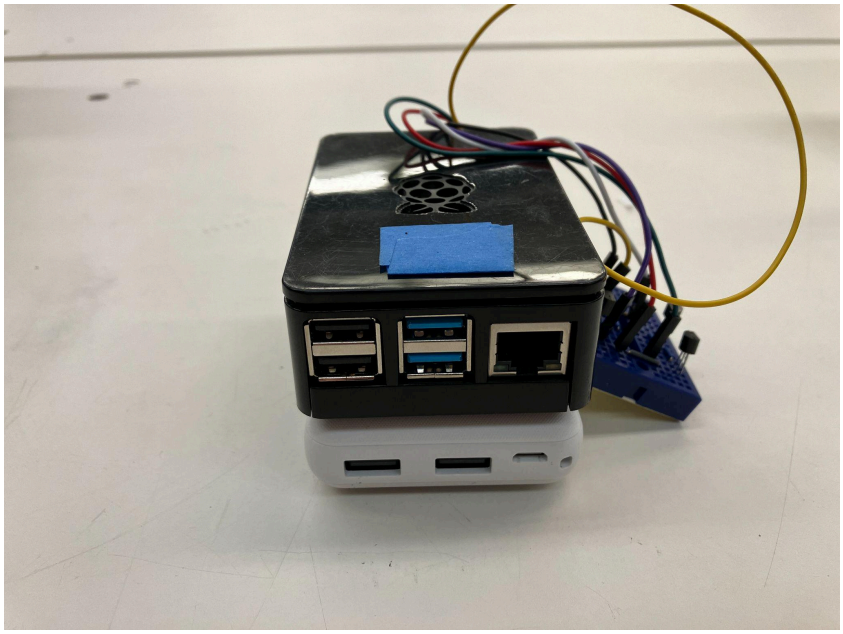
2. Then, put the components together
    a. Open the bottom of the RC car using a small phillips head screwdriver and connect the yellow RC car battery pack, closing the bottom and securing it again



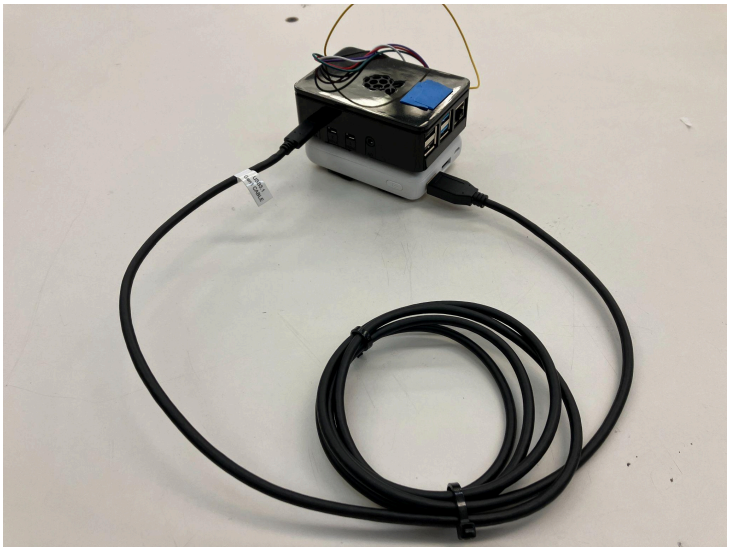
b. Attach the mount with the screwed in LiDAR sensor to the RC car body using the attached velcro, making sure to align the pegs at the front of the car body with the hole in the front of the black plastic



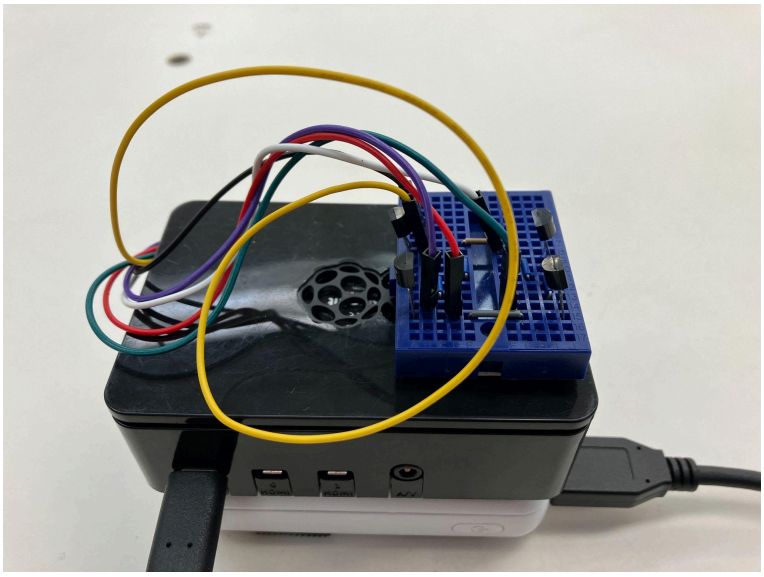
c. The Raspberry Pi is attached with Velcro to the white external battery pack. Ensure that the side of the Pi with USB outlets and the side of the battery pack with USB outlets face the same direction



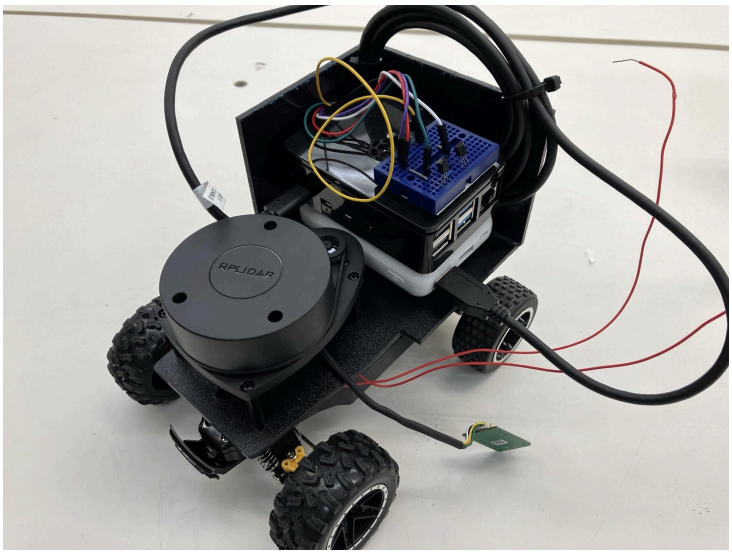
d. Connect the external battery pack to the Pi's power input (labeled on the case) using the USB to USB-C cable



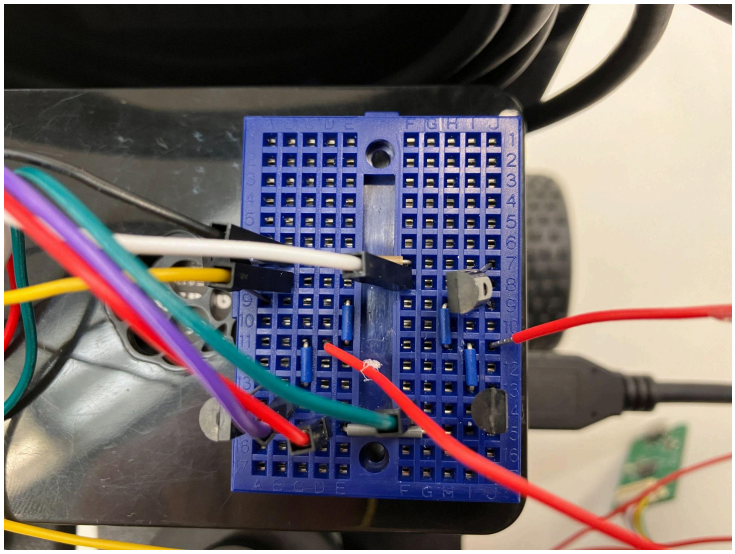
e. Attach the breadboard to the top of the Pi case using the pre-applied masking tape



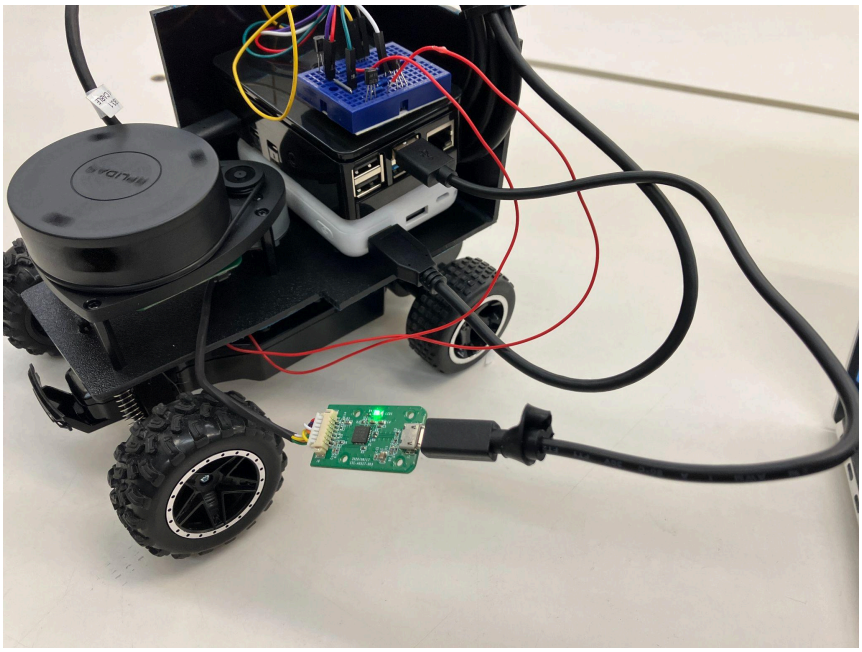
f. Attach the combined Pi, white external battery, and breadboard unit to the back of the black plastic housing using the attached velcro
    i. Ensure that the side of the Pi and battery pack module with USB outlets faces the open side of the black plastic housing
    ii. The excess slack on the USB to USB-C cable can be slotted behind the combined Pi/battery unit



g. Take the two red wires attached to the RC car body and connect them to the breadboard as shown in the image below



h. Connect the LiDAR sensor to the Raspberry Pi using the USB to micro USB cable



3. Power on the Pi by pressing the small power button located on the side of the white external battery pack

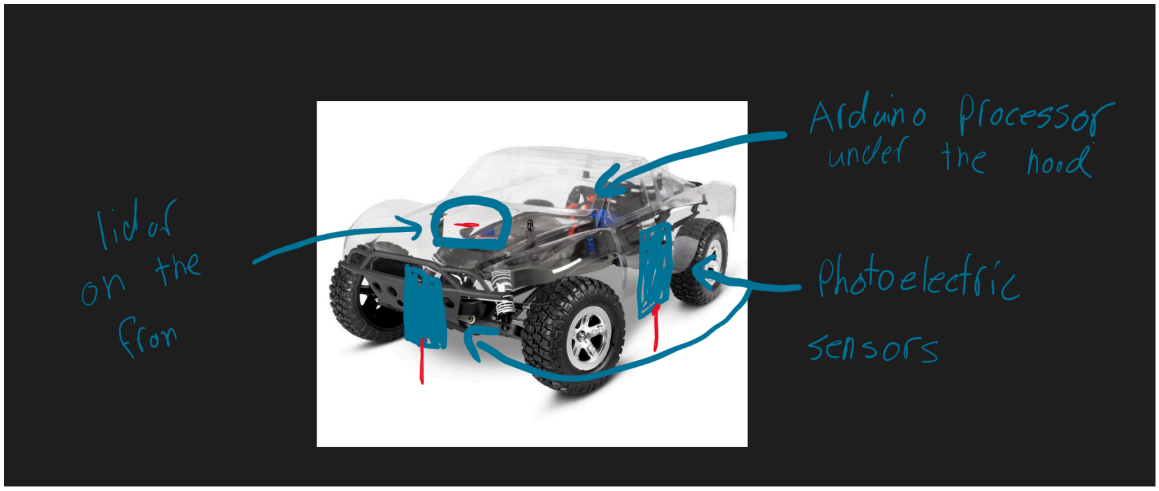
4. SSH into the Pi on the laptop
    a. This will require the IP address of the Raspberry Pi, which will change from day to day. If you have not yet gathered the IP address, briefly connect the Pi to a monitor, keyboard, and mouse, open the terminal, and run the ip addr command to find the current IP
5. Open a terminal and navigate to the directory `~/Race-Of-Doom-Git-Folder/sdmay24-43/rpLiDAR_sdk-master/output/Linux/Release/`
6. Run the command "`sudo ./ultra_simple --channel --serial /dev/ttyUSB0 115200`"
7. The RC car's autonomous steering is now online
8. Flip the power switch on the bottom of the car to on, and turn on the RC car remote control. The car is now ready to drive

9. The car can be accelerated by pulling the trigger of the remote towards the handle, and reversed by pushing the trigger away from the handle
    a. A higher pressure on the trigger will result in a higher speed. A slower speed will allow for better wall detection and automated steering

10. If the car is going to hit a wall, try reversing the car to give the autonomous steering another chance to detect and adjust to the wall

Appendix 2 - Alternative/Initial version of the design

The initial design of the car:

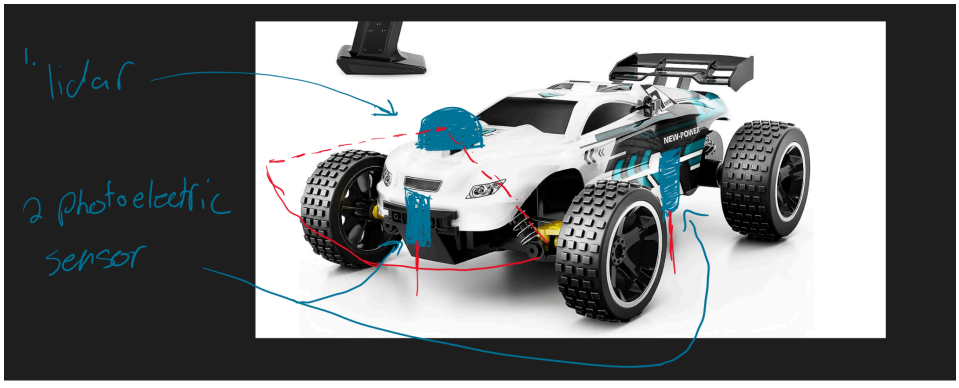


First specifications:

- LiDAR on the front of the bot for object detection
    - This will allow the bot to detect obstacles on the track
- Photoelectric sensors on both sides of the bot as well as the front of the car
    - detects the borders of the track or holes
- Our processor will be hidden away underneath the hood of the car to protect it from damages
    - There are thoughts of increasing the processor with some sort of protection against crashing
- The possibility for other sensors are still being considered like for example, an IR sensor

The first iteration of the RC car before further specifications included a much larger RC car with the LiDAR sensor on the front of the hood, as shown in the image above. Additionally, the initial design included photoelectric sensors on the front and sides of the car that were going to detect black lines on the ground. However, the specifications later changed that the track would use raised cardboard walls, effectively making the photo-electric sensors useless. The initial design also utilizes an Arduino under the back hood, but was later switched to a Raspberry Pi for more processing power and to operate more complex tasks.

Second Initial Design:



1. LiDAR - DFR0315 from DFRobot. This LiDAR spins on its own, so we can detect objects in the track with this component. We won't need to build a motor and IR sensor combo
2. PhotoElectric Sensors - SEN0239 from DFRobot. This simple photoelectric sensor will let us detect the track's borders by detecting the change in color
3. MicroProcessor - Raspberry Pi 4 - We've decided to use a Raspberry Pi for our processor because it is more versatile than an Arduino. This processor will also be hidden underneath the hood of the vehicle in order to protect it from the elements.
4. RC car - Tecnock RC Racing Car. This car is a simple and cheap RC car which allows us to spend more money on our components

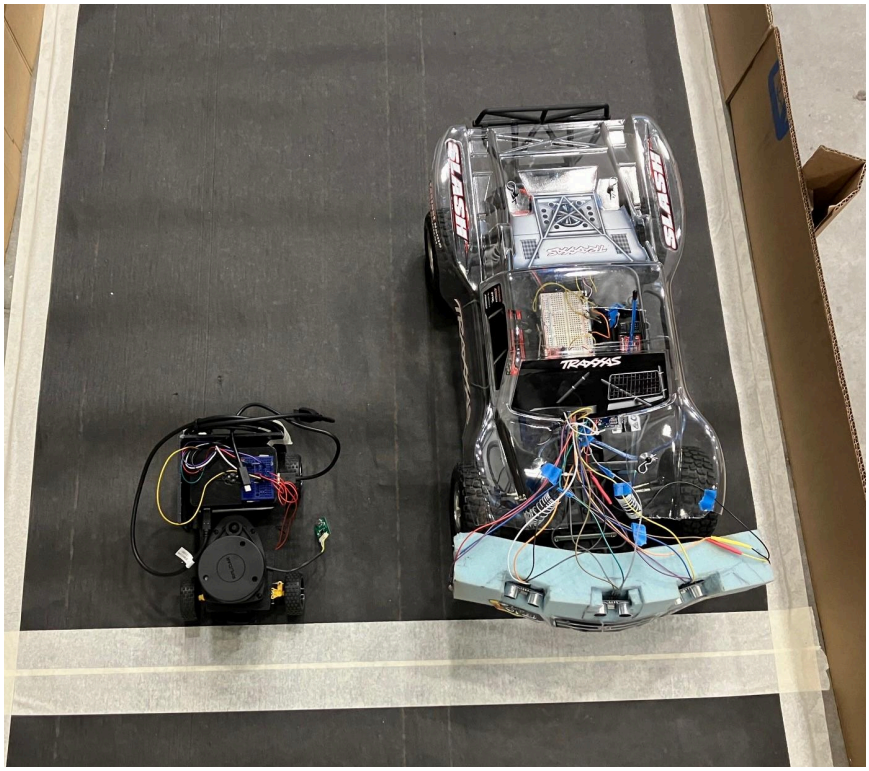
The second initial design shows the exact RC car utilized in our project with a LiDAR sensor on the hood and two photoelectric sensors on the side. As stated before, the photoelectric sensors were removed from the final design as the LiDAR sensor proved efficient at detecting walls for autonomous functionality, so the photoelectric sensors were removed from the final design.

The RC car shown above had much smaller dimensions than initially expected, making using the LiDAR on the hood not feasible. Furthermore, the size of the Raspberry Pi and the battery used in the final design were much larger than anticipated, making our final design much different from either initial design.

The initial choice for software design involved utilizing Python on the Raspberry Pi. However, the decision was later revised to employ C++ because the LiDAR sensor's Software Development Kit (SDK) is exclusively available in C++. This change was necessary to ensure seamless integration with the LiDAR system and to leverage its full functionality within the project.

Appendix 3 - Other Considerations

One aspect of this project that quickly became evident when the RC car body first arrived at the end of the previous semester was the almost comically small size of the RC car body when compared to the components attached to it. This comedic contrast is even more notable when comparing the car to the vehicle designed by the other car team. Sitting the two next to each other, the difference in size is distinct and hilarious.



Appendix 4 - Code

```
/*****************************************************************
 * Known Constraints
 *
 * (nodes[pos].angle_z_q14 * 90.f) / 16384.f shows the current angle.
 * Around 1180 scans are done per 360 degrees.
 *
 * nodes[pos].dist_mm_q2/4.0f is limited to values of 100-2000 (10cm to 2m)
 *
 * nodes[pos].quality is has only been observed to have values of 0 and 47.
 * 0 is for invalid reads (distance is 0), 47 is valid.
 *
 *
 * Placeholder Values for Distance:
 *
 * Ground < 1000 away.
 * Wall < 500 away.
 * Angle: 300 - 360, 0 - 60
 * Critical Wall Angle: 340 - 360, 0 - 20
 *
 *
 * Common Function Comparisons:
 *
 * (nodes[pos].quality != 0) - indicates if a scan is valid (greater than
0).
 * ((nodes[pos].dist_mm_q2/4.0f) < 1000) - indicates if we are looking at
the ground (or closer).
 * ((nodes[pos].dist_mm_q2/4.0f) < 500) - indicates if we are looking at a
wall (or closer).
 *
 *
 * Call to print out all values:
 * printf("%s theta: %03.2f Dist: %08.2f Q: %d \n",
                 (nodes[pos].flag & SL_LiDAR_RESP_HQ_FLAG_SYNCBIT) ?"S ":"
",          // Observe if this is a sync bit, which will make sure there
are no clocking issues.
                 (nodes[pos].angle_z_q14 * 90.f) / 16384.f,
           // Observe what angle we are currently looking at.
                 nodes[pos].dist_mm_q2/4.0f,
     // Observe what distance away we are from the object currently being
```

```
looked at.
                    nodes[pos].quality >>
SL_LiDAR_RESP_MEASUREMENT_QUALITY_SHIFT);       // Determine if this is a
quality measurement (non-zero)
 *
 *
 *****************************************************************/



        usleep(200000);
        if (SL_IS_OK(op_result)) {
            drv->ascendScanData(nodes, count);
          int rightWeight = 0;
          int leftWeight = 0;
          int rampWeight = 0;
            for (int pos = 0; pos < (int)count ; ++pos) {
            if (((nodes[pos].dist_mm_q2/4.0f) < 500) && (nodes[pos].quality
!= 0) && (((nodes[pos].angle_z_q14 * 90.f) / 16384.f < 60) ||
((nodes[pos].angle_z_q14 * 90.f) / 16384.f > 300))) {

                // Printing scans where the result is valid, less than 0.5m
away, and in the 40 degree range in front of the car.

                if ((((nodes[pos].angle_z_q14 * 90.f) / 16384.f) > 340)) {
                  // Critical wall angle to the right.

                  if (((nodes[pos].dist_mm_q2/4.0f) <= 200)) {
              // Check if we are very near a wall to turn into.
                      leftWeight--;
                  } else {
                      leftWeight++;
                  }

                } else if (((nodes[pos].angle_z_q14 * 90.f) / 16384.f) <
20) {       // Critical wall angle to the left.

                  if (((nodes[pos].dist_mm_q2/4.0f) <= 200)) {
              // Check if we are very near a wall to turn into.
                      rightWeight--;
                  } else {
                      rightWeight++;
                  }
```

```
                }

            } else if (((nodes[pos].dist_mm_q2/4.0f) == 0) &&
((((nodes[pos].angle_z_q14 * 90.f) / 16384.f > 70) &&
((nodes[pos].angle_z_q14 * 90.f) / 16384.f < 90)) ||
(((nodes[pos].angle_z_q14 * 90.f) / 16384.f < 290) &&
((nodes[pos].angle_z_q14 * 90.f) / 16384.f > 270)))) {

                // If we are getting invalid on the edges, continue
straight as we are on the ramp.

                 rampWeight++;
                 if (rampWeight >= 69) {
                  // continue straight
                  gpioWrite(17, 0);
                  gpioWrite(27, 0);
                  gpioWrite(13, 0);
                  gpioWrite(19, 0);
                  leftWeight = 1;
                  rightWeight = 1;
                  printf("Go straight on the ramp\n");
                }
            }
          }

          if ((abs(leftWeight - rightWeight) <= 10) && (rightWeight != 0)
&& (leftWeight != 0)) {
            // If there are walls on both sides...
            // continue straight
            gpioWrite(17, 0);
            gpioWrite(27, 0);
            gpioWrite(13, 0);
            gpioWrite(19, 0);
            printf("Walls on both sides... stay straight\n");

          } else if (rightWeight > leftWeight) {
            // If there is much more wall presence on the right...
            // turn left
            gpioWrite(17, 0);
            gpioWrite(27, 1);
            gpioWrite(13, 0);
            gpioWrite(19, 1);
            printf("Turning Left - right: %d - left: %d \n", rightWeight,
```

```
leftWeight);

          } else if (leftWeight > rightWeight) {
            // If there is much more wall presence on the left...
            // turn right
            gpioWrite(17, 1);
            gpioWrite(27, 0);
            gpioWrite(13, 1);
            gpioWrite(19, 0);
            printf("Turning Right - right: %d - left: %d \n", rightWeight,
leftWeight);

          } else {
            // If there is no immediate danger, center the car
            float leftDist = ((nodes[983].dist_mm_q2/4.0f));
            float rightDist = ((nodes[197].dist_mm_q2/4.0f));
            if ((abs(leftDist - rightDist) <= 50)) {
                  // continue straight
                  gpioWrite(17, 0);
                  gpioWrite(27, 0);
                  gpioWrite(13, 0);
                  gpioWrite(19, 0);
                  printf("Straight into the wall!\n");

            } else if (rightDist > leftDist) {
                // turn right
                gpioWrite(17, 1);
                gpioWrite(27, 0);
                gpioWrite(13, 1);
                gpioWrite(19, 0);
                printf("Turning Right - Centering...\n");

            } else if (rightDist < leftDist) {
                // turn left
                gpioWrite(17, 0);
                gpioWrite(27, 1);
                gpioWrite(13, 0);
                gpioWrite(19, 1);
                printf("Turning Left - Centering\n");

            } else {
                  // continue straight
                  gpioWrite(17, 0);
```

```
                    gpioWrite(27, 0);
                    gpioWrite(13, 0);
                    gpioWrite(19, 0);
                    printf("Straight into the wall!\n");
                }
            }

            if (ctrl_c_pressed){
              break;
            }
        }
    }

    drv->stop();
      delay(200);
      if(opt_channel_type == CHANNEL_TYPE_SERIALPORT)
        drv->setMotorSpeed(0);
    // done!
on_finished:
    if(drv) {
        delete drv;
        drv = NULL;
    }
    //reset, clear, etc
    gpioTerminate();
    return 0;
}
```