MEAM 5100 - Grand Theft Autonomous 2022

Group - 22
Dhruv Parikh
Parth Sanghavi
Yug Ajmera

January 5, 2023

Contents

1 Functionality

We successfully completed **ALL** functionalities and checked-off with **Extra Credit** :

1. Control the robot
2. Wall Following autonomously
3. Autonomously identify, go to and push either trophy or fake (using 23Hz or 700Hz beacon tracking)
4. Use the Vive system to transmit your X-Y location
5. Autonomously move to the police car box and push it 12 inches.

Approach to win the game

We built a differential drive robot with two wheels at the back and two castors in the front. Two ultrasonic sensors, one at the front and one on the right are used to follow the wall autonomously using a PD controller. Two IR photo transistors are attached on the front (with casing on them to reject noisy signals) to detect beacon of certain frequency and move towards it. To autonomously reach the beacons on the other side of the arena, we first follow the wall for about 10 seconds and then start our tracking process. Using our current XY location obtained via Vive system we employ a hybrid strategy where we first follow the wall to reach the high fidelity zone and then use the police car coordinates to approach it and push it towards the opponent area. Our priority was to complete the robot for graded evaluation with all necessary functionalities. The following are the design choices made in order to achieve the same.

1.1 Controlling the Car

We used two DC Gearbox Motor at the back and 2 caster wheels at the front of the car. The setup of the car was differential drive. Due to time constraint, we intended to avoid complications with the holonomoic drive. With this setup, if we control the left wheel and right wheel using PWM, we can achieve turning.

This task was similar to Lab 4. Our code followed a Finite State Machine model. To switch states, we had a HTML5 web interface with Soft AP mode, that can switch different modes using buttons, and a display header that displays the current mode. As the ESP had to serve the entire web page, the lesser the code length, lesser is the latency and better is performance. Hence we made sure our entire code works on only 1 javascript function to handle all the buttons and a GET request to indicate mode. This also makes sure that the page doesn't have to be refreshed on each call.

1.2 Wall Following

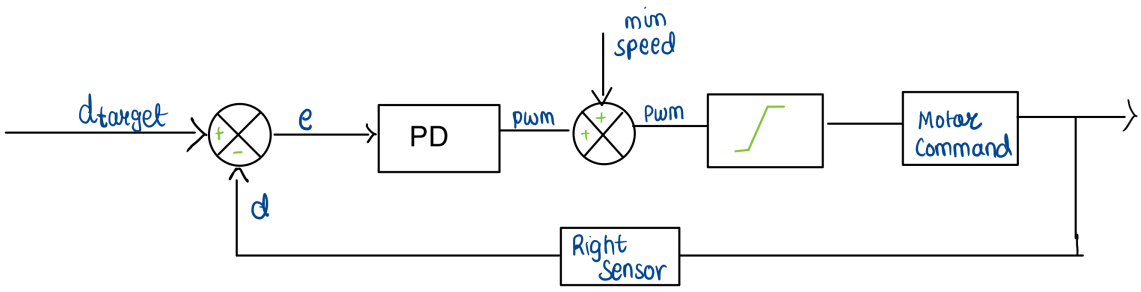


Figure 1: Controller Block Diagram

We used 2 ultrasonic sensors, one in the front and one on the right hand side of the robot. Figure 2 shows the placement of the sensors and two situations that the robot encounters while following the wall.

In situation ①, i.e. when the forward reading is greater than 15cm, the robot follows a PD control algorithm shown in Fig. 1. This is to make sure the robot follows the wall till it reached the end. The right sensor reading is used as the current distance and the target is set to 15 centimeters. The error is computed as the difference between the current and target, and the PD controller is designed to give PWM outputs to the respective left and right motors. For example, if the robot is too close to the wall, the right wheel is given greater PWM for the robot to turn left and move away from the wall, and vice versa.

Tuning K_p and K_d in PD controller was loosely based on Ziegler–Nichols method. K_p is used to tune the response to the error, while K_d controls the damping of the oscillations. Our robot followed the wall perfectly with $K_p = 30$ and $K_d = 5$ values.

In situation ②, when the robot reached the end of right wall, the robot reverses for 100 ms, takes a left turn by 90 degrees (this open loop and based on delay) and then switches to ① mode again. This approach enabled our robot to follow the wall smoothly.

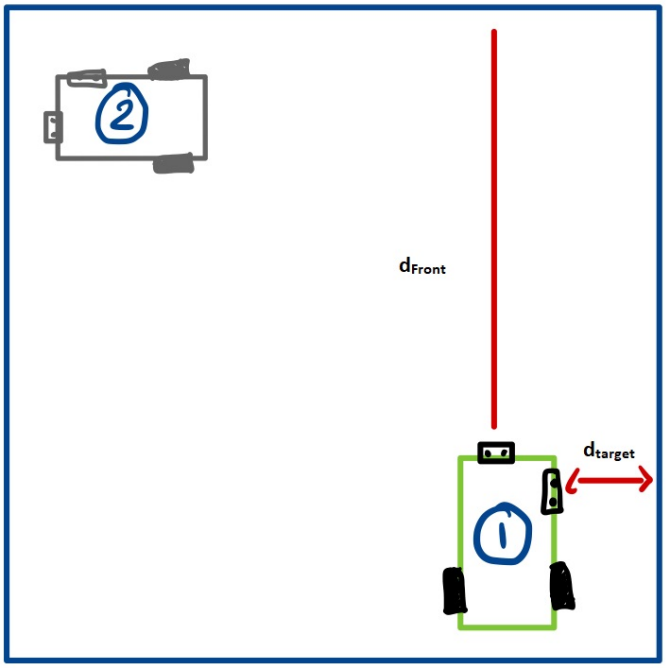


Figure 2: Wall Following Modes

Wall following is the workhouse of our entire code as it is used as an assistive mode for each of the other modes (to reach the other side of the arena if beacon is far, or to reach the high fidelity zone to push to police car). Therefore it was crucial to get it working perfectly.

Previous Approach: Initially, we tried a bang-bang control approach with delay to perform wall following. We defined a tolerance zone of 15 cm ± 5cm from the wall such that if the car goes out of this zone, it should turn left and vice-verse. Because of the latency, the correction was not instantaneous and required a lot of tuning. PD controller on the other hand was robust to changes in the system (such as weight/speed), provided smoother control and had lesser parameters for tuning.

1.3 Beacon Tracking

For this functionality, we used two LTR4206 IR diodes and mounted them at the two ends of the front side of our robot. We also 3D printed a hollow cylindrical cap and attached them on the diodes to make sure they only receive signal from the front and all the other noisy signals are obstructed.

To search for the beacon, we rotate our car until we get a signal (desired frequency) from both the diodes, and then move forward, towards the beacon. Whereas if the signal is detected on only diode, the car rotates in that direction until we get a signal on both of them. Figure 3 shows the beacon tracking behaviour.

If due to external noises we loose signal from either or both of the diodes, we rotate our robot in the direction where the frequency was last detected. For example, rotate right if the right diode detected the signal before the

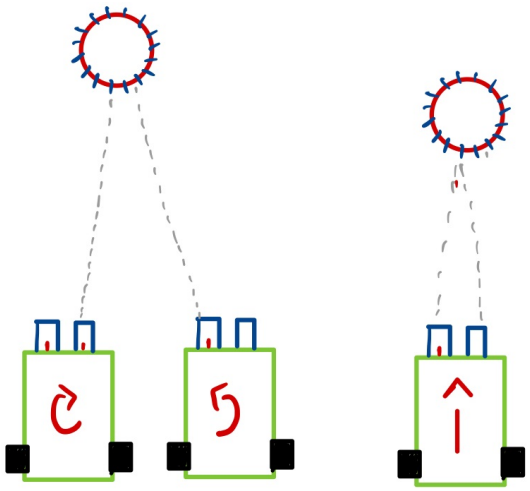


Figure 3: Beacon Tracking

signal got lost in both of them. We even tried rotating the robot in any one arbitrary direction for searching, but our approach was much faster and reduced the search time.

Our beacon circuit was able to sense the frequency upto 180 cm (about half of the arena). In case the robot is not able to detect the beacon on any one of the diode, i.e. the beacon is on the other side of the arena or if we want to reach the beacon that is on the other side, we perform wall following for 1000 iteration ($\approx$ 10s) to reach the other side and then shift to the beacon tracking mode.

1.4 Pushing the Police Car

To complete this task, we used the UDP protocol to read the vive location of the police car and filter those values (described in detail in Section 4). We assumed that the police car is stationary between the time we start and reach its location. When this mode is initialized, multiple samples of police car location is read and then stored in a variable. A low pass filter with very low cutoff frequency is them employed.

For localization of our car, we used a vive sensor (PD70-01C diodes) and the circuit and code provided to us that gives us the real-time XY location. With extensive testing, we realised that these readings are robust when the robot is close to the vive system (which is at the centre of the arena) and error increases as it moves farther away. We define this region as high fidelity region shown in Figure 4.

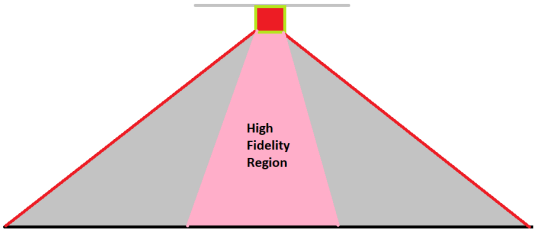


Figure 4: Vive Senor Trust Zone

If the robot is away from the centre of the arena, we first use wall following to reach this high fidelity zone. Then the vive readings are used to reach the police car location and push it forward. This approach allowed us to overcome limitations arriving from unreliable vive location.

With our hybrid approach, this task worked out in the first test itself! Since our wall following was very robust, and the vive location was very reasonable in close range, our tests had 80-90% success rates usually converging under 10 seconds.

2 Mechanical Design

2.1 Design Methodology

Given the restrictions for the vehicle to be smaller than 30cm by 30 cm by 30 cm, we designed the vehicle to be 24 cm by 27 cm by 7 cm. The mobile base has a differential drive setup, enabling the vehicle body to rotate with a negligible turning radius with the help of two yellow 12V DC motors and two caster wheels mounted at the back and front of the chassis. We chose the differential drive since it is easy to implement and to avoid issues with the steering mechanism. We used 72mm diameter wheels mounted on the shafts of 2 DC motors.

The primary requirement for the chassis was to have mass at the bottom part of the robot to prevent tilting if we collided with another robot and wide enough so that all the components could be readily put on the chassis. The robot chassis was wide so that the police car can be hit easily. Moreover, in lab 4 we found that the design was not big enough to mount all the circuits, hence this solidified our design decision to go towards bigger design.

The power bank and LiPo battery were placed in the bottom deck of the robot while all the light components were on the higher deck to meet the second criterion: the robot needed a lower center of gravity to be stable. We also added two castor wheels in the front to provide additional stability to the robot.

With all the weight considerations, TT motors provide sufficient torque. A few factors, including battery capacity, were considered when choosing a DC motor because high torque motors may quickly deplete a battery due to their high current demands. We also considered perfboards, batteries, and circuitry placement beforehand. Hence, we decided to add adjustment slits on the chassis so that the circuit can be supported by zip ties, and wire management can be done properly.

The motor mountings are 3D printed PLA, and the chassis is built of 3mm acrylic. The electronics components are mounted on standoffs to separate any soldering below the soldering boards. We used the same mount that came with the ultrasonic sensors we ordered online. HTC vive was also mounted on top of our chassis with the help of standoffs. The decision was to keep it at the top, so nothing could interfere with the signal that is being received and transmitted by the sensor.

We also additionally 3D printed the casing for the two IR. The IR was mounted using the 3D-printed mountings, that were made to hold them, and the design was made in such a way that, all the surrounding lights wouldn't hamper the signal received by the micro-controller.

We went through roughly 3-4 design iterations (including Lab 4.2 car design) to get to the final design. Wire routing and lengths were also considered during the mechanical design to eliminate long cables, complicated wiring, and troubleshooting. In the final design, we achieved modularity, lightweight, simplicity, and a lower center of gravity.

2.2 Failures and Learning

Three items did not work:

- Our beacon mount was 3D printed according to the first iteration of the beacon provided to us. But because the beacon's design changed, reducing its height in a later edition, we were required to alter our beacon mount.
- We also constructed mounts for the ultrasonic sensor; however the initial run of these mounts was damaged due to the collision of our chassis with the wall during our initial testing for wall following. We decided to buy the mounts online because of the timing restrictions.
- In the earlier design iteration, we incorporated the idea for a gripper at the front of the robot to grab the beacon and return it to its box in the final competition. But we did not go further with it as we did not participate in the competition.

  Although we had the grippers designed and manufactured, the actual task of grabbing the beacon and bringing it back required a lot of effort because of our time constraints. So, we decided to skip that step and instead proceed to the beacon and push it.
- We created a 3D-printed platform for the HTC Vive circuit to be on top of everything so that the sensor could send and receive information from the device. However, in a subsequent design, it was decided not to do this and instead utilize standoffs to provide the circuit with the necessary height in the interest of time.

3 Electrical Design

3.1 Methodology

This section outlines the methodology of designing circuits for each subsystem. Although each of our systems had noise, we used filters in software as they can be tuned.

Beacon Circuit

The appendix contains a diagram of the beacon circuit. The phototransistor's sensing range was planned to be 3 and a half meters when the circuit was designed, but its actual performance was only about 2 meters. We employed a variety of electronic components in this circuit, including resistors, transistors, capacitors, opamps, and comparators. We used two opamps and significantly raised the gain to extend the sensing circuit's range. Even after doing this, the oscilloscope only picked up a voltage of less than 2 volts, and the ESP detects logic high and low at a difference of 3.3 volts.

We utilized a comparator to generate logic high and low for the ESP since the voltage difference needed for ESP to detect the signal should be more than 3 V. We gave the opamp an offset voltage of 1.7V by greatly raising the gain in order to create a voltage difference across the phototransistor.

Component Selection for Beacon Circuit:

- TLV727 - Opamp
- TLV272 - Opamp used as Comparator
- LTR4206 - Phototransistor

HTC Vive

Vive circuit is shown in the appendix. Initial vive circuit that was shown in class was chosen for the final implementation. At first, the circuit was missing on some pulses. Hence the gain of first opamp was increased to 2 MΩ and the first resistor was decreased to 47 KΩ. By doing this we were able to detect all the pulses.

The second circuit shown didn't worked for us. We believe it was because of the high pass filter not filtering all the values since there were many spikes detected. However as the circuit was revised towards the end of the semester, we didn't had time to debug it further.

Component Selection for HTC Vive:

- TLV727 - Opamp
- PD70-01C - Phototransistor

Ultrasonic

We had two choices of distance measuring sensors - ultrasonic and ToF. We decided to go with Ultrasonic as it was cheaper. Moreover, a resolution of mm was not required for wall following as the arena was fairly simple.

Component Selection for Sonar:

- HC-SR04 - Ultrasonic sensor

Motors and Battery

Our motor driver circuitry was same as in lab4. We used 1 SN754410 motor driver for controller two motors. We anticipated a voltage drop of 2 Volts, and hence we used a 2 Cell 7.4 V LiPo battery with 1300 MAH capacity. The main reason for not using other motor drivers was that we knew that the ampere draw of the motors is low and hence the battery won't drain that quickly.

For powering ESP32 we used a power bank of 5V. We intended to have separate power line for the MCU and Motors and avoid daisy chaining in general. Hence the battery is only used for motors and all other sensors/MCU

are powered through the battery bank.

3.2 Failures and learnings

- Initially while designing the beacon circuit, all the components were assembled on a breadboard. The circuit was working perfectly fine, and even sensing the 700 Hz frequency up to a distance of 3-4 meters, but when all the components were soldered together on the solder board, the circuit gave noise on its output. So in the final iteration we had to switch to the Bread board circuit.
- The initial circuit of Vive that was provided needed to give correct values. The oscilloscope showed valid peaks. However, we found out that the small peaks (X/Y) were not reaching logic high for the ESP to detect. We increased the gains so that it reaches a valid output (described in the section above) and it worked out.
- We planned to use vive circuit throughout the arena hoping for reliable values. However, we found that the mounting of the vive sensor plays a very important role. If mounted too high, the sensor starts throwing garbage values at the edge of arena. This had to be handled; thus, we used a hybrid approach to tackle this issue.

4 Processor architecture and code architecture

4.1 MCU Block Diagram

We used 2 ultrasonic sensors, 2 IR diodes and 1 vive circuit to fulfill all the functionalities. We used the ESP32 Pico D4 and allocated the pins in such a way that we only required one micro-controller to execute all the tasks. A LiPo battery is used to power the motors, whereas the sensor are powered using the ESP which is powered through a power bank. Such a power division is made so that LiPo gets drained relatively slowly and our robot lasts in the game longer. The block diagram is shown in Figure 5.

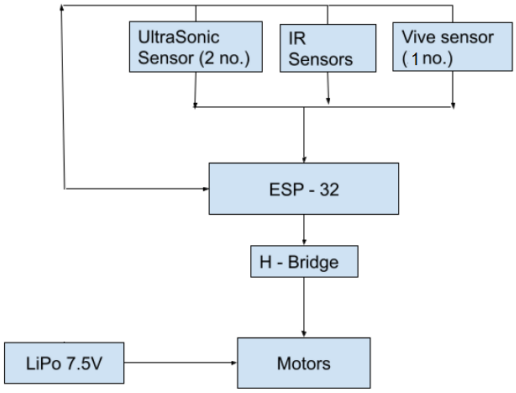


Figure 5: MCU Block diagram

4.2 Software Approach

Our design philosophy was to achieve the functionality with minimum complexity. We realised that the interrupt based routine calls are prone to bugs as a clean exit from a function is not guaranteed. Hence, our Software architecture is based on Finite State Machine Model.

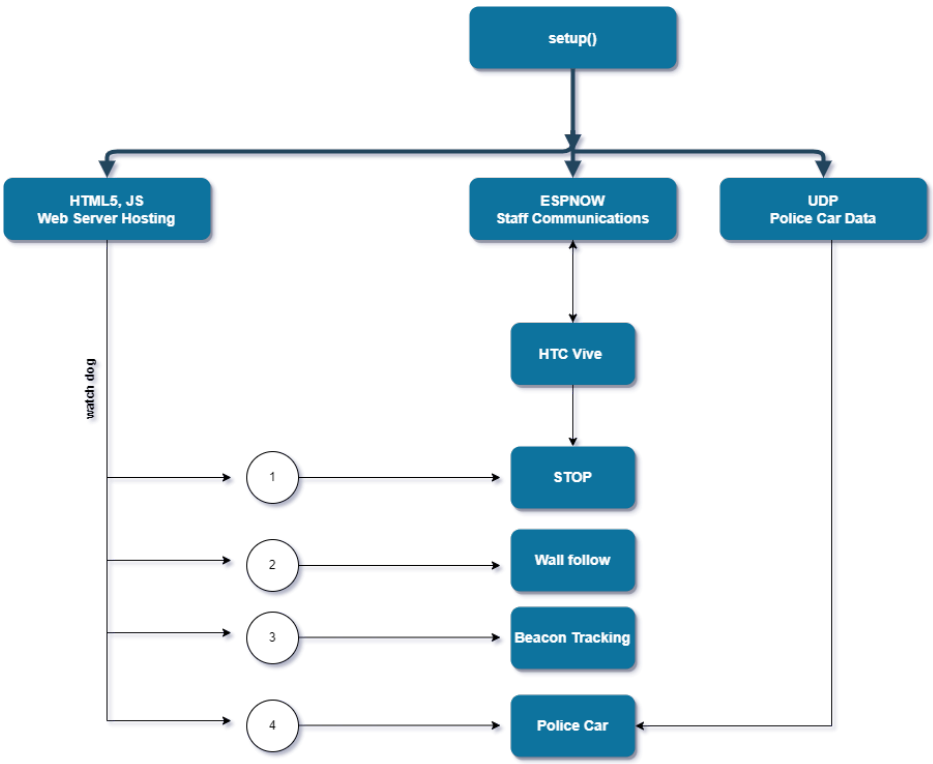


Figure 6: Code Architecture

To that end, we used a HTML5 webpage to switch between behaviours using buttons. We have separated functions for each functionality and the GET request buttons switches between the functionalities smoothly. This approach led to minimum delay in communication and gave modularity to our code allowing fast and efficient changes whenever required. Figure 6 shows the code architecture.

We start with initializing all the variables, making objects and establishing communications in the setup. ESPnow communication runs in the void loop regardless of the state. We have also attached a callback for handling police car data. With this setup, we can control all the states using the web interface. Upon each state request, we call the respective function which are described in detail in the following sections.

4.2.1 Web interface and Staff Communications

Figure 7: Web Page

Figure 7 shows the web page that was hosted from the ESP32. In the main loop, h.serve() is called every time so as to poll the commands sent from the website. Based on the state, the code simply calls their respective function. The HTML5 contains 4 buttons and a header which indicates the current state of the robot. There is 1 JS function that handles all the button press and updates the header. Upon the execution of the JS function, GET request is called and ESP32 reacts on that. Such a light module of HTML helps us to get low latency and therefore good performance of the robot.

The staff communications was done using ESP-Now. We had a master time that kept track of time. When the desired frequency matches, we send the data to the staff by packing the message into 13 length character array.

4.2.2 Filters

Throughout the software architecture, we have used 3 window median and a low pass filter with variable cutoff frequency in the software. This applies for each of the sensors (Vive, Ultrasonic and Beacon). Adding a low pass filter added delay to the system but also avoided noisy readings which damped out the car performance.

4.2.3 Wall Follow

Wall following functionality was done using the a PD control architecture described in figure 1. Each ultrasonic sensor reading is passed through a low pass filter of 5 windows (weight = 0.7). Figure 8 describes in the detail the logic of the wall following code.

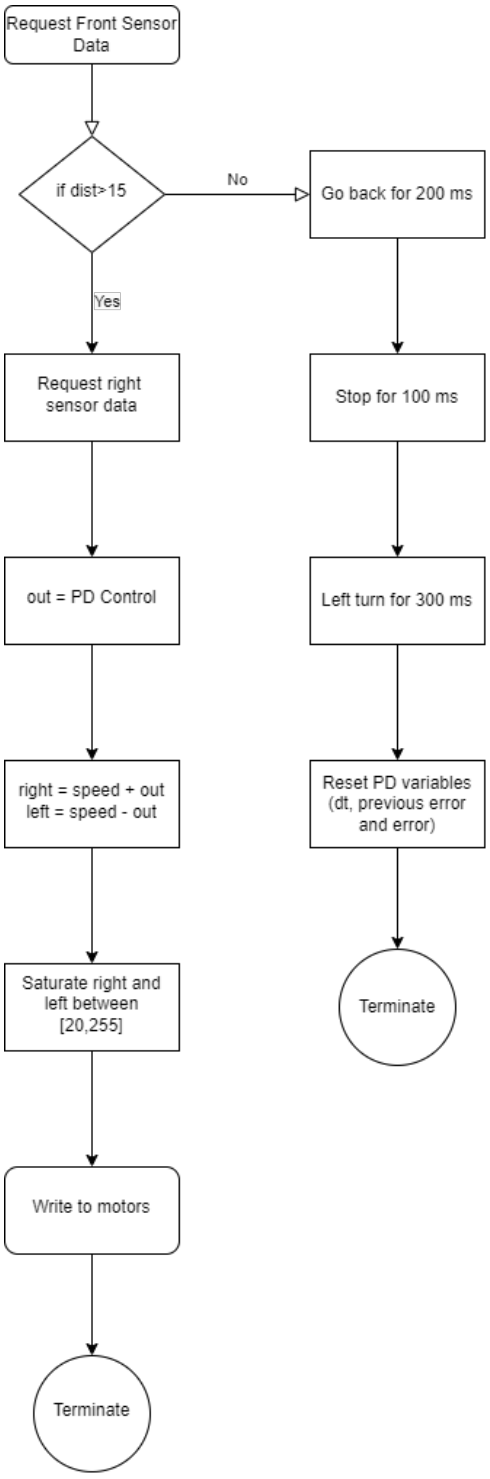


Figure 8: Wall following Logic Diagram

4.2.4 Beacon Tracking

Figure 9 describes the beacon tracking flow in detail. Upon first call, we enter in wall following mode for a 1000 iterations (9-10 seconds). This allows us to reach to the center of the arena. Now, we switch to the beacon tracking mode where we can now see the trophy in both halves of the arena.

To detect the frequency, we calculate time of high pulse and low pulse. Inverting this will give us frequency. This frequency is then low passed with 5 windows that outputs a very smooth signal.

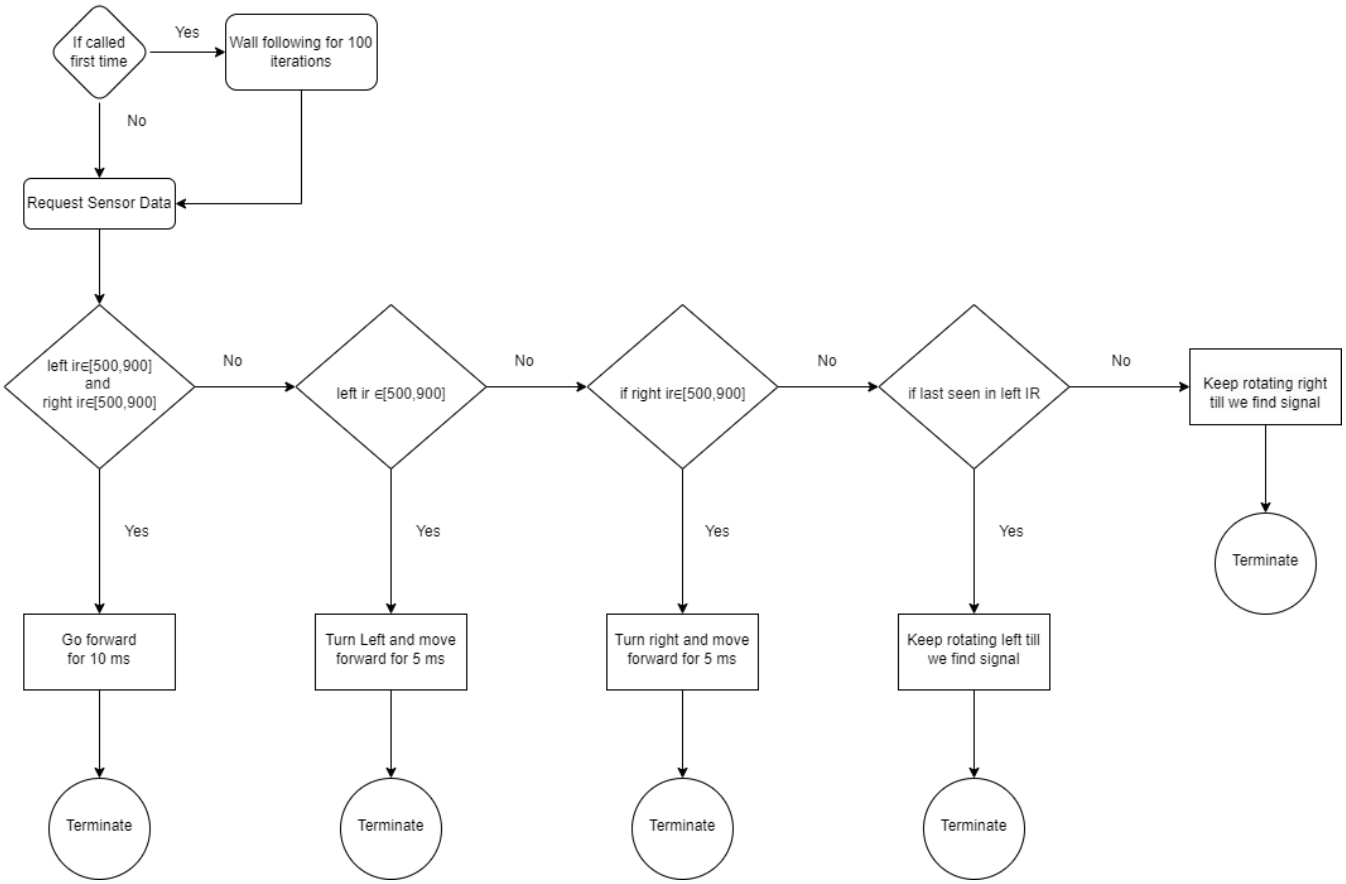


Figure 9: Beacon Tracking Logic Diagram

4.2.5 Push the Police Car

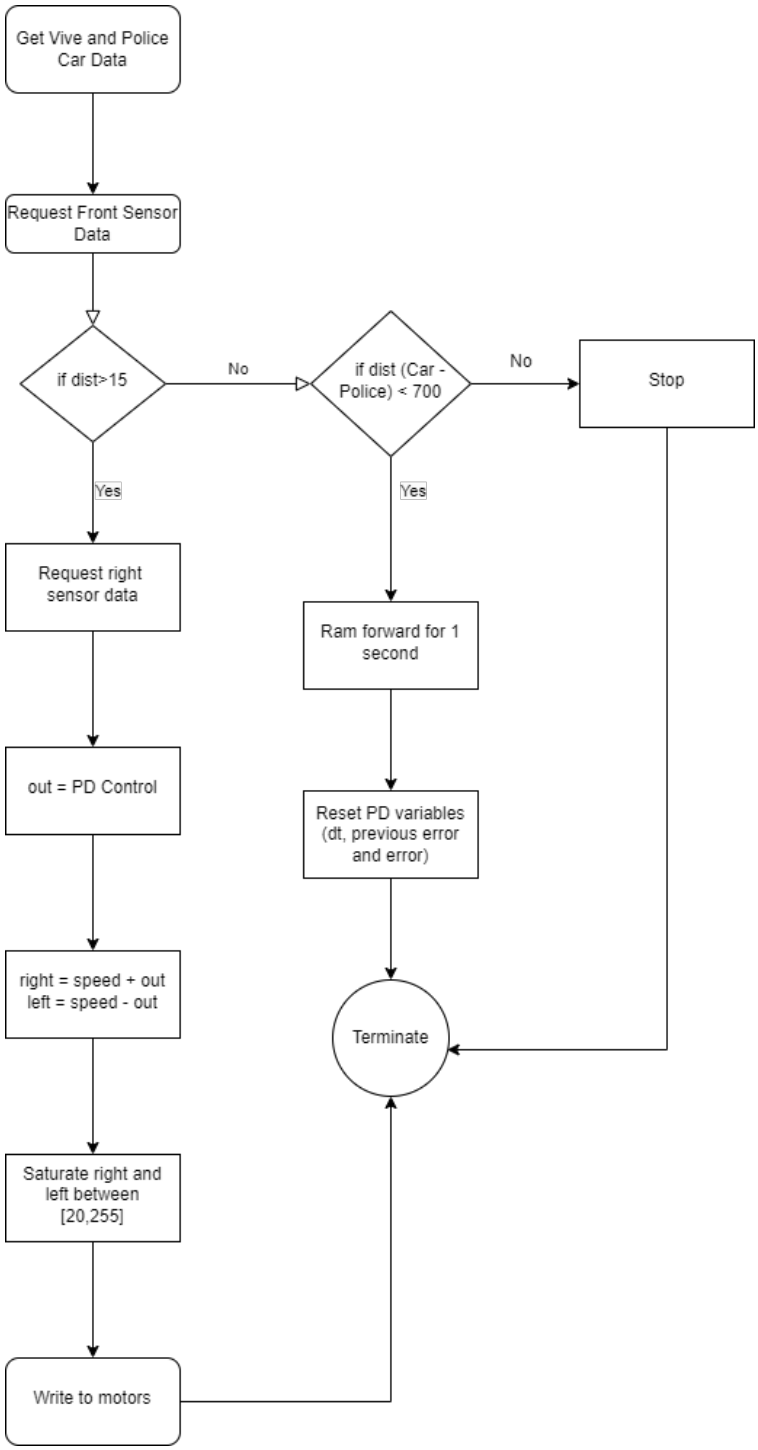


Figure 10: Police Car Mode Logic Diagram

Police Car pushing was one of the most challenging task of the game. We got smooth readings from vive when we used a 3 window median filter coupled with a low pass filter of 20 windows. The police car location is assumed to not change by a large margin. Hence when the function is called, we first take the police car data around 20 times, use a low pass filter of 20 windows and get a very robust data.

With this, we went in the high fidelity region of vive described in figure 4 using wall following and then once we are inside the region, we use the vive to ram the police car. However, if the vive data doesn't agree, i.e if the range of the data exceeds our thresholds, then we stop the car and wait till we are getting reliable data.

This architecture proved to be very robust and reliable where we obtained success rate of 100% even when starting with bad orientation.

4.3 Failures and Learnings

- **Global Variables** - Scope of variables is extremely important in C++. At first when we were using the functions for modularizing the code, we realized we couldn't return multiple values. Hence we were forced to use global variables (we didn't wanted to use pointers). However, the global variables are very difficult to keep track of and they are prone to being used in all the functions. For example, dt was calculated using t1 which is a global variable of previous recorded time. t1 is used in sending vive data and wall following. However, both usage have different meaning. For sending vive data, we record the overall time of the loop and for wall following, we only wanted loop time of the wall following and then reset the time. This lead to a lot of errors and the vive data was not being sent at all.

- **Initializing the filters and D controller** - Initializing filters is of utmost importance. If you start with any value for low pass filter, it would take a lot of time to converge to the correct value. If you initialize the low pass value incorrectly everytime its called then the filter will never provide a good estimate. Hence, initializing the filters are very nescessary and should be done in first function call before anything else.

- **Print Statements delay** - Print statements although useful, induce a lot of delay in the program. This delay being in microseconds don't mean much to us; however, in the controller, it can impact the performance. We were having multiple print statements in the loop and hence the loop speed was slow. This meant that the controller corrections were delayed resulting in overshoots. Once we remove the print statements, the corrections became a lot smoother.

- **Code integration** - When we integrate code, the loop overhead increases. There are a lot more variables to handle and a lot more functionalities that run in parallel. This impacts the loop speed of our code. We saw a decrease in performance when we integrated the entire code. For example, at first only wall following code was there in the loop. However in the final integration, vive sensor reading and sending module was present in the loop. This function takes 20-25 ms to compelete. Hence for 25 ms, we can't perform wall following! We had to retune our PD gains due to this.

- **Differences in control architecture** - A control architecture that reacts according to the error works much better than bang bang type control. For wall following we were trying to using bang bang control with some deadband in middle, however this didn't worked as expected and hence we used a PD controller which works more directly on error. Moreover, the system was tunable using standard conventions and hence we didn't had to hard code correction parameters.

5 Retrospective

- What you feel was most important that you learned

  1. Time management
  2. Patience

- What was the best parts of the class

  1. The TAs and the professor were very responsive on Ed discussion which really helped us a lot.
  2. Application of the theory in the labs allowed us to appreciate the application aspect

- What you had the most trouble with

  1. Vive sensor was unreliable for us. We had plans to implement more complex and robust algorithms, we had a very good reading. This proved to be a major bottleneck in our project.
  2. Soldering was troublesome since the circuit that worked on breadboard didn't work when soldered. Every circuit was soldered for our car, but we ended by using the breadboards as the circuits just stopped working.

- What you wish was improved

  1. We had major plans to participate in the competition, but couldn't due to the vive issues. It was difficult to debug as we didn't understand the circuit completely.
  2. Communication using i2c and spi could had been made compulsory in the project because those protocols are industry standard and would had provided us with a good platform to implement them.

- Anything else about class.

  1. Labs were the most exciting component in the class as we got to interact with a lot of people and see their interesting designs.

6 Appendix

6.1 Bill of Materials

Part	*Material*	*Quantity*	*Cost*	*Total Cost*
Chassis	Acrylic	1		
TT 5V DC Motors	OEM	2	$2	$4
Wheels	OEM Adafruit	2	$ 3	$ 6
Castor Wheels	Amazon	2	$ 4	$ 8
ESP 32 PICO D4	Had previously	1	$ 10	$ 10
LiPo Battery 7.4V	Given by TA	1	$ 10	$10
Aluminium Standoffs	Mini Store	25		
5Ah Power Bank	Given By TA	1	$ 8	$8
Jumper Cables Wires	Mini Store	20		
HC-SR04 Ultrasonic Sensor	Amazon	2	$3	$6
Capacitors & Resistors	Mini Store	50		
TLV 727 Opamp	Mini Store	15		
IC SN754410 Motor Driver	Mini Store	1		
Miscellaneous Hardware	Mini Store			
Total				$52

6.2 Photos, renderings of the robot and CAD

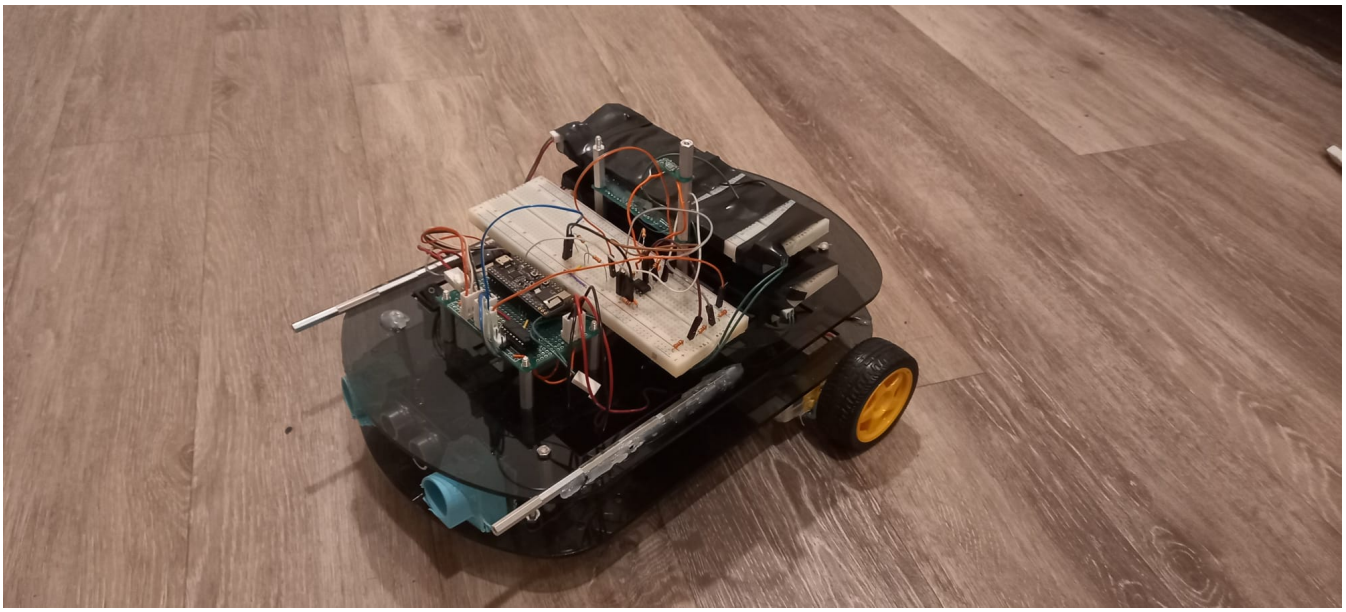
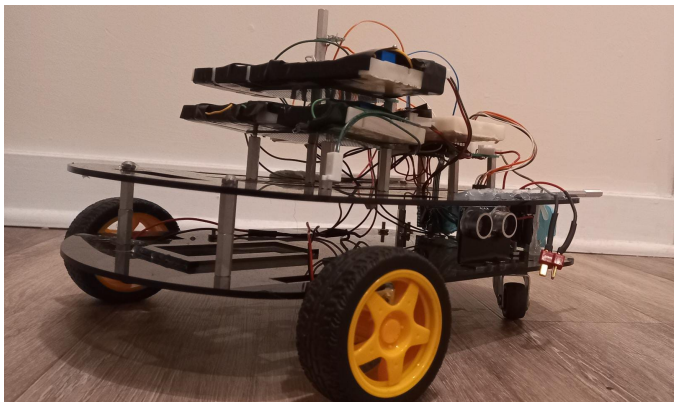
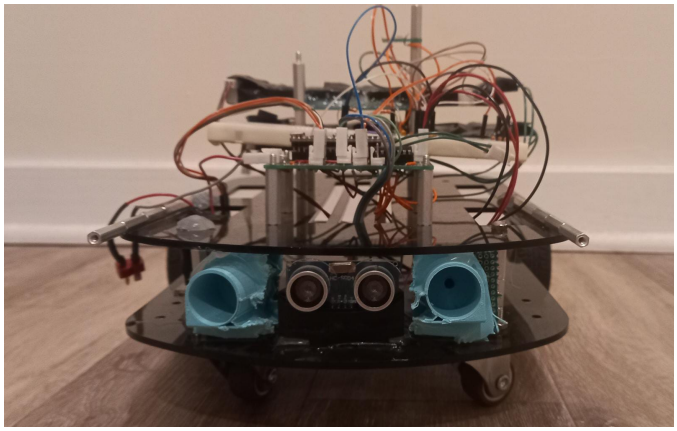


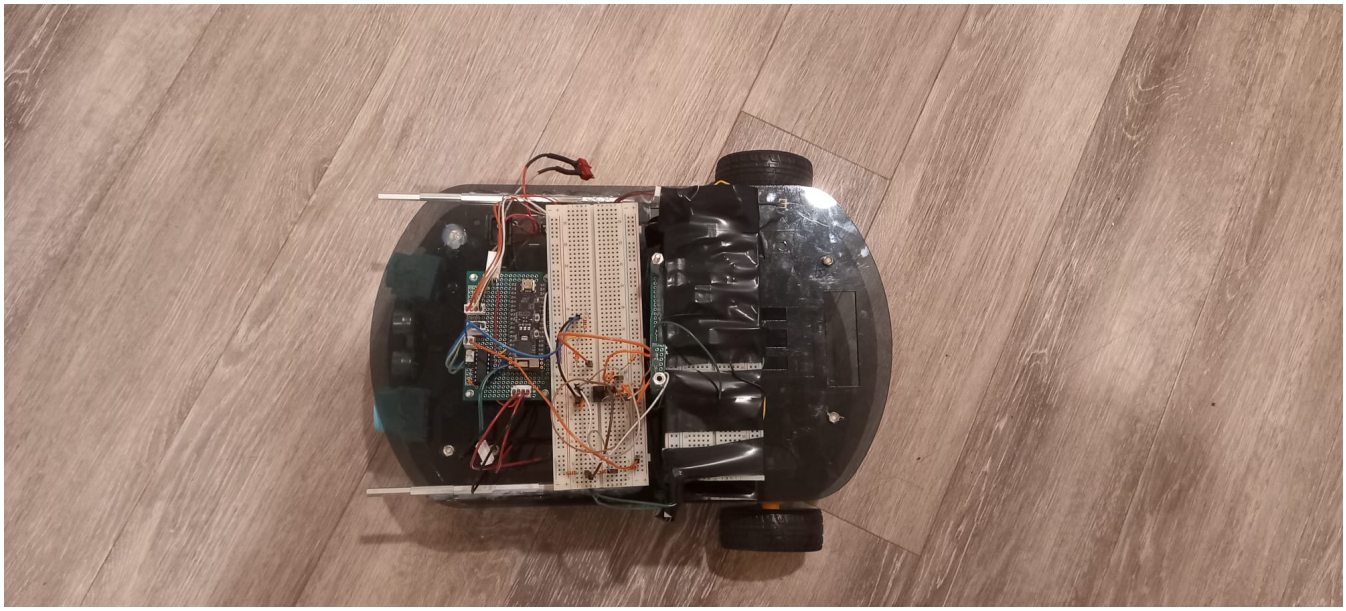
Figure 11: The Terminator



(a) View 1



(b) View 2



(c) View 3

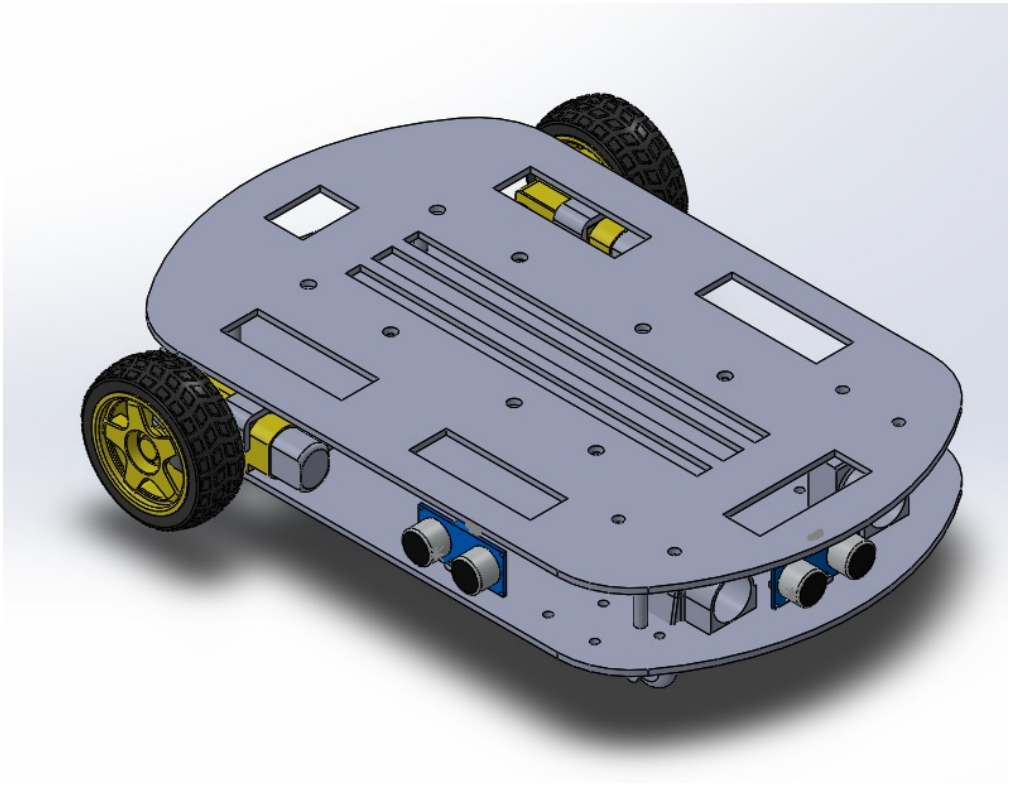


Figure 13: 3D Rendering

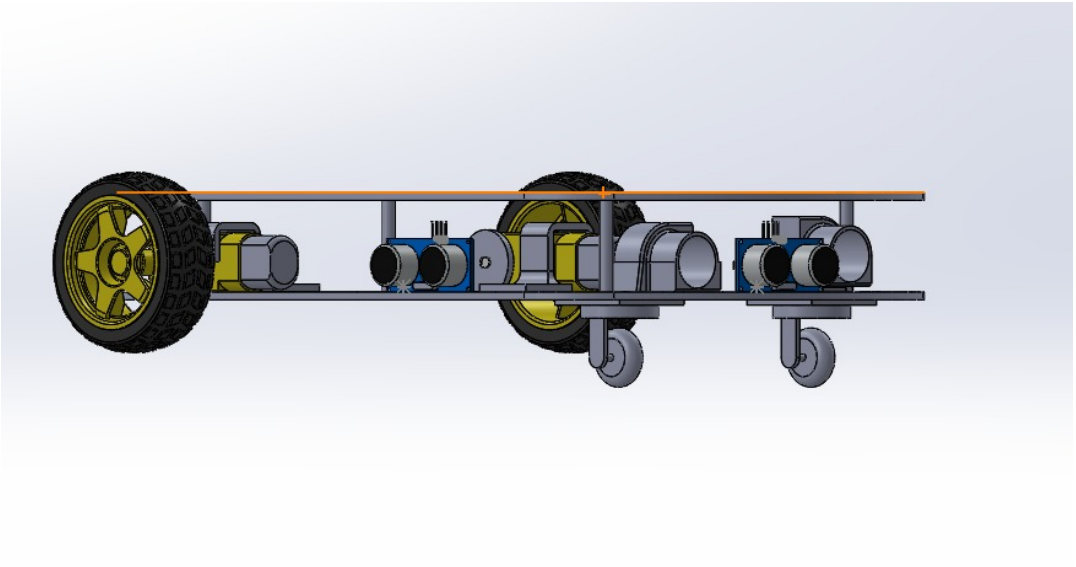
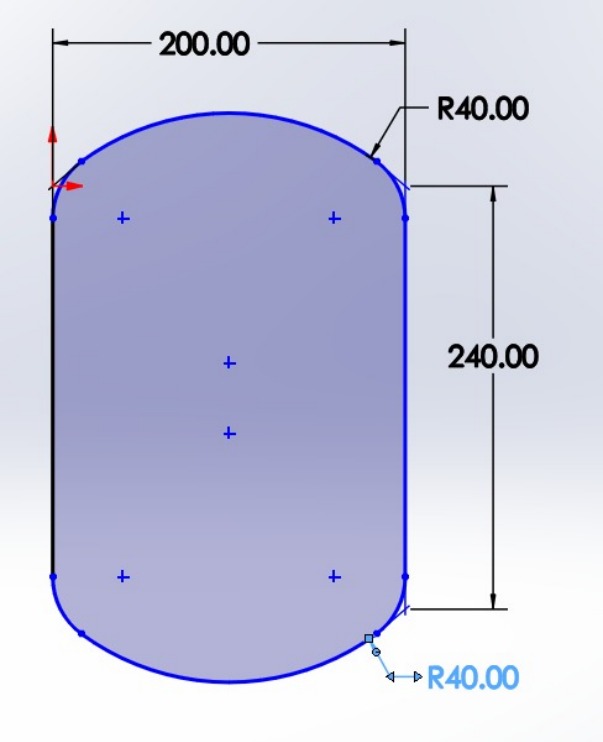
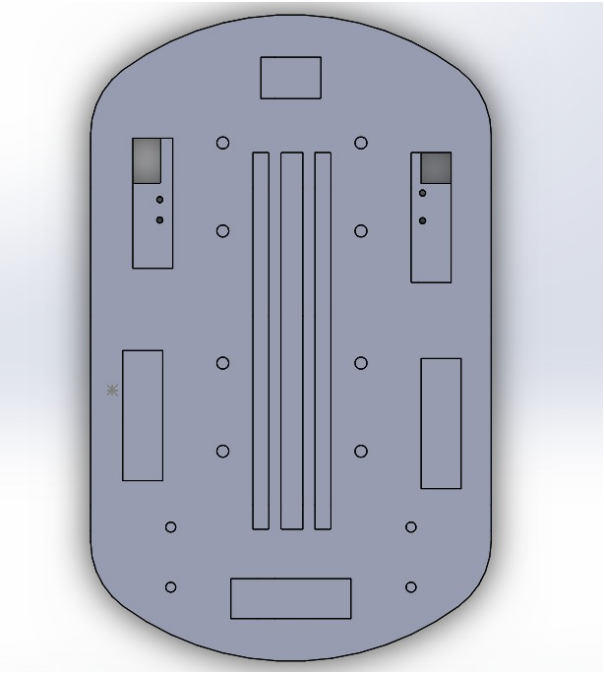


Figure 14: 3D redering Bottom View

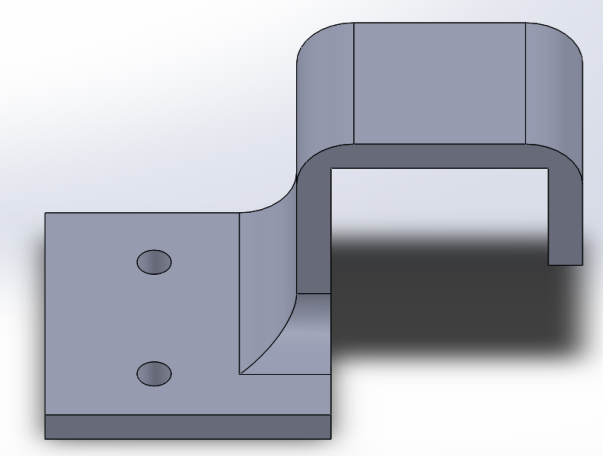


(a) Base with measurements

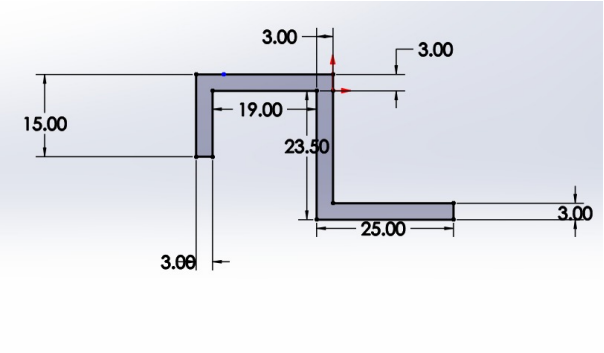


(b) Top view of the base

Figure 15: Base of the Robot

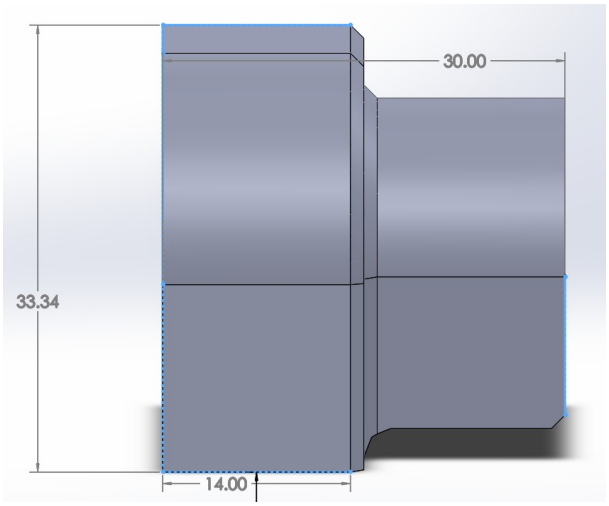


(a) Motor Mounting

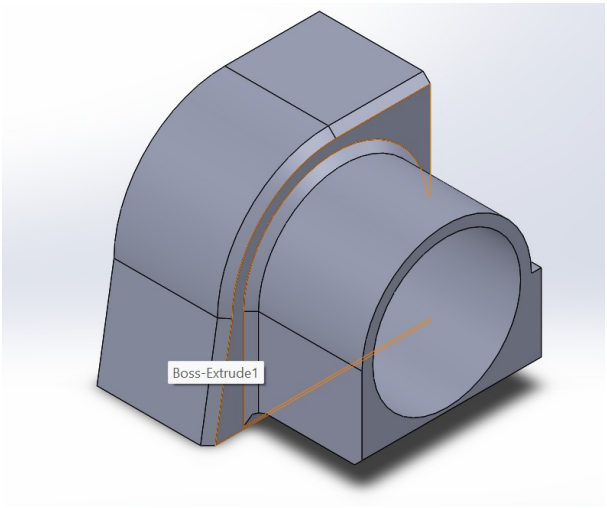


(b) Motor Mounting Dimensions

Figure 16: Motor Mounts



(a) Beacon Mounting Dimensions



(b) Beacon Mounting

Figure 17: Beacon Case

6.3 Circuit Diagrams

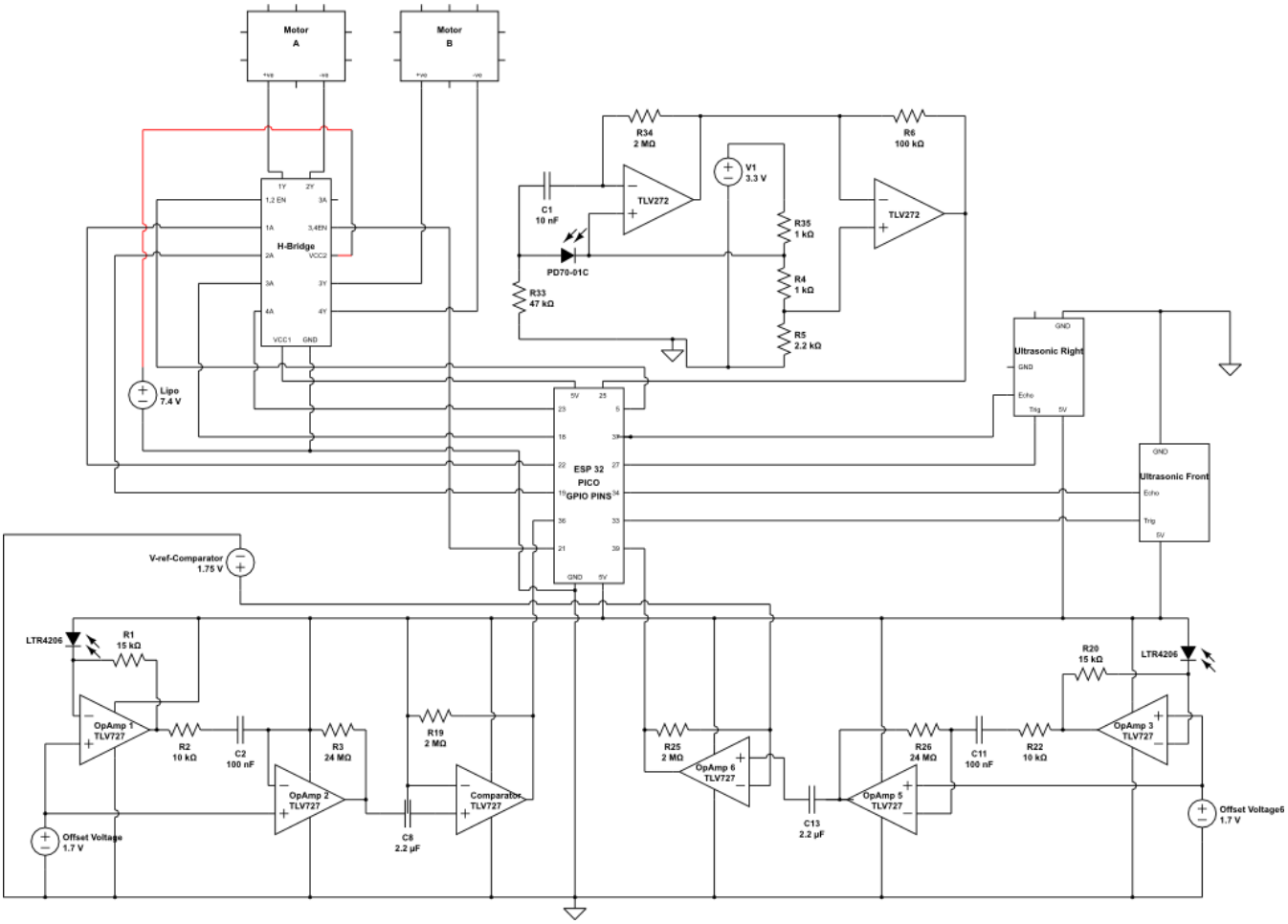


Figure 18: Full Integrated Final Circuit

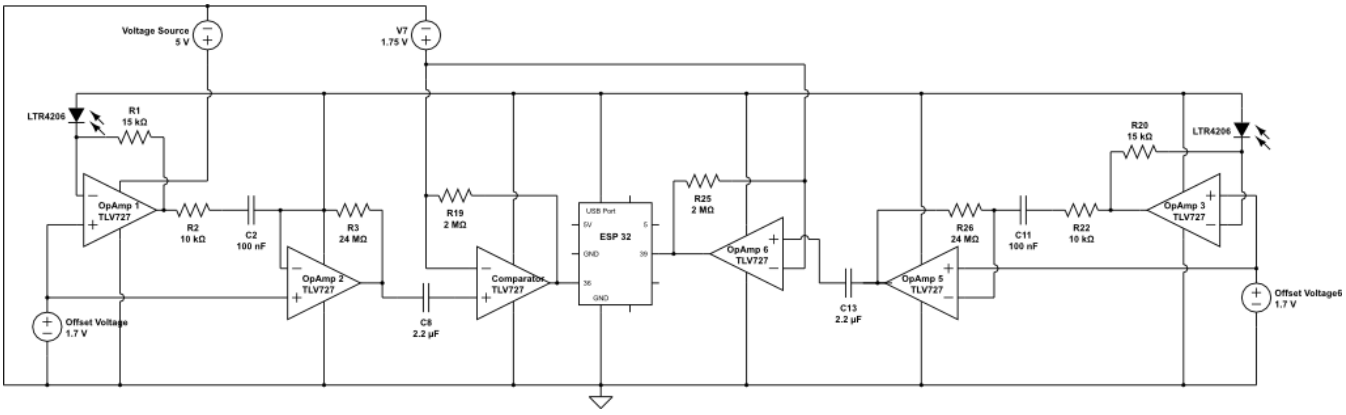


Figure 19: Circuit of Beacon Sensing

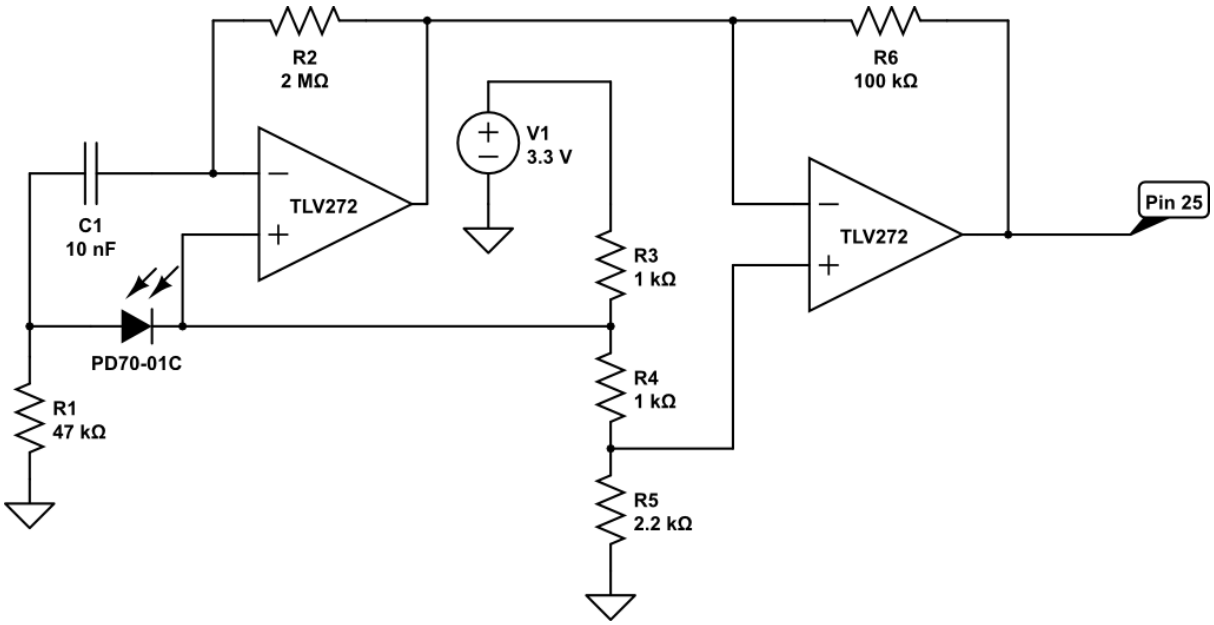


Figure 20: Vive sensing circuit

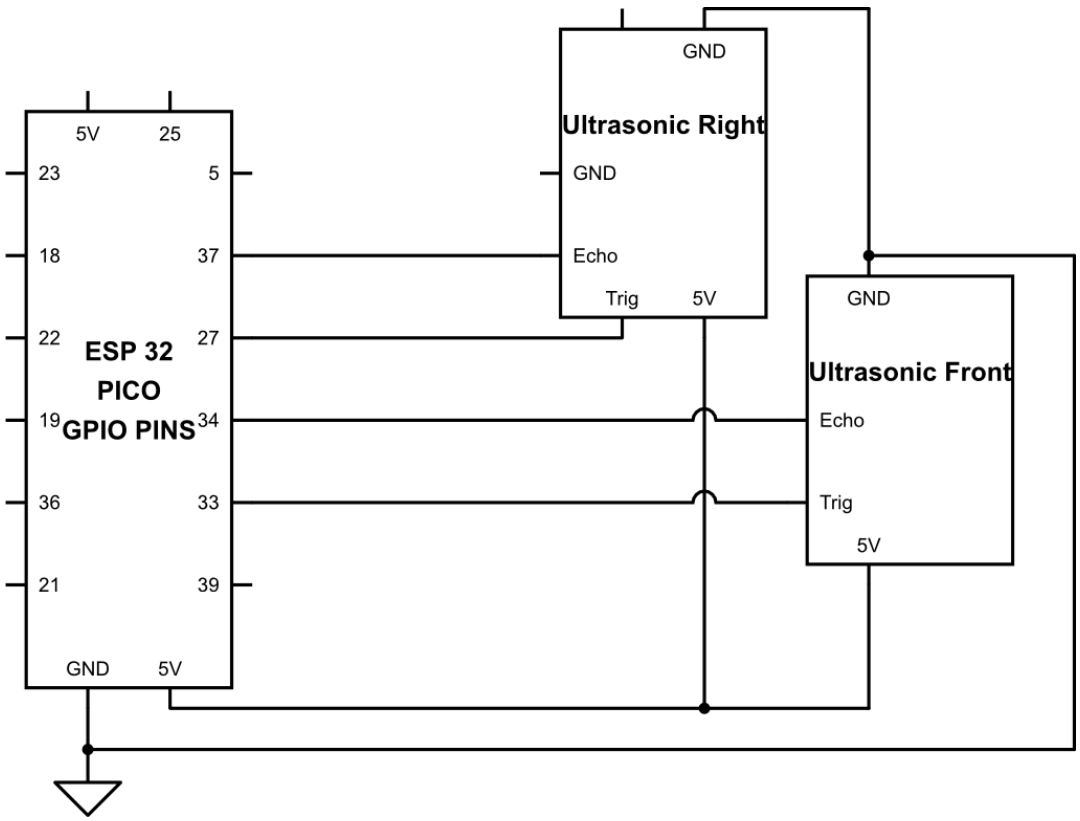


Figure 21: Ultrasonic circuit

6.4 Datasheets

- Ultrasonic Ranging Module HC - SR04 Data sheet
- CNHL 2200mAh 7.4V 2S 30C Lipo Battery (Datasheet unavailable)

6.5 Videos

Videos of the working functionalities can be viewed here:

- Wall Following
- Beacon Tracking 1, Beacon Tracking 2.
- Pushing the Police Car