MEAM 5100 Final Project Report

Authors: Gia Dcosta, Saayuj Deshpande, Samhitha Vedire

Aim

The goal of this project was to build a reliable autonomous robot capable of attacking the nexus and tower, attacking static and dynamic enemy targets, attacking the TA Bot and driving up the ramp. This project also aimed at transmitting WiFi usage to the Top Hat of the robot at 2 Hz.

Functionality

Gameplay

We developed the following gameplay strategies:

Autonomous

ToF sensors should be used to detect obstacles and other robots in the robot's path. When an opposing robot is encountered, the robot should slow down or stop based on proximity. If the distance reaches a minimum threshold, a servo-driven weapon should extend to strike the opponent's whisker switch. In scenarios involving the nexus or towers, the robot should slow down to try and avoid the swinging arms. Upon identifying a safe opportunity, the robot should approach the button and apply a slow forward motion to keep it pressed. A similar strategy should be employed for tower button interactions.

Manual

Our main strategy for this part of the game was to focus on attacking the buttons on the nexus and towers. If any other robot got in our way the next task would be to approach another button, and keep rotating amongst the targets.

Design Choices for Gameplay

In order to make sure button pressing and fending off other robots was efficient, we built our bot to be heavy by using a 1/4" acrylic sheet as the base plate. We kept the spacing between each layer of the bot as small as possible in order to make the motion more stable, especially up the ramp. To increase the stability further, we decided to use four wheels. The design and structure of our robot is explained in detail in next section.

Specific Functionalities

This section describes the high-level approach to the specific functionalities required by the robot. The code is described in detail in the code architecture section.

Localization and Navigation

Infrared sensors were used for HTC Vive tracking to get the robot's approximate coordinates on the field. We soldered two Vive circuits to get position and orientation. The locations of each known nexus and tower, along with the buttons, were recorded to use in the autonomous behavior codes.

Wall Following

The robot utilized Time-of-Flight (ToF) sensors for continuous localization and navigation. Side-mounted ToF sensors measured the distance to the nearest wall and helped the robot to adjust its steering dynamically to maintain the desired safety distance. A front-facing sensor detected corners and obstacles, allowing the robot to adapt its path as needed. We were able to execute wall-following during our initial tests, but due to a last minute issue with the ToF sensors not working, full tuning on the field was not possible.

Target Attacking

The overall idea behind this part of the functionality was to sequentially go to each target on the field and press the button. If an obstacle encountered does not match any of the recorded target locations, then an attempt to hit the opponent's whisker switch should be made. Although we were able to get coordinates using the Vive circuits, we were not able to fully test autonomous target attacking on the field.

Communication and Manual Controls

On opening the webpage, three buttons would appear - targets, wall and manual. During the manual part of the gameplay, we focused on having a way to switch between driving the robot ourselves and letting it run code to attack other targets. In addition to directional control using buttons, We incorporated emergency commands such as a motor stop and a weapon control button that would stop the robot from moving before swinging the weapon.

During our tests and on the field, manual control performed very well. The directional movement of the bot was stable and we did not need a feedback control loop to adjust motor speeds. The bot also traversed the ramp during our testing. Button pushing was also successful given the weight of our robot.

Mechanical Design

Aim

The aim of the mechanical design was to build a stable and sturdy robot, capable of withstanding the weight of its components without causing mechanical failure.

Description of the Robot

Our robot utilized a differential drive system powered by 500 RPM motors. The mechanical configuration included a four-wheel drive setup with standard wheels to ensure stability and ease of control during operation. The mechanical design of the robot can be divided into three layers. Each of these layers was joined together with the help of 3 mm fasteners (dowels and screws).

- The overall height of the robot was measured to be 7 inches.
- The overall width of the robot (including wheels) was measured to be 8 inches.
- The overall length of the robot was measured to be 10 inches.

Material Used

The material used for building this robot was Acrylic (1/4" as well as 1/8" sheets).

Manufacturing Process

The process for manufacturing the robot was laser cutting, as it is precise and time-intensive.

Robot Layers

1. **Bottom Layer:** This layer, made of 1/4" acrylic sheet, served as the base plate to stabilize the robot by supporting the weight of all heavy components (e.g., batteries, motors). Components placed in this layer included LiPo batteries, motors, ToF sensors, and the whisker switch.

2. **Middle Layer:** This layer housed the main circuit of the robot, motor drivers, and a cutout for the whisker switch to protrude to the top layer.

3. **Top Layer:** This layer contained the Top Hat, photosensors, and a servo motor. Both the middle and top layers were made of 1/8" acrylic sheets.

Intended and Actual Mechanical Performance

Wheels and Steering

Initially, we explored using mecanum wheels to achieve omnidirectional movement. These wheels were intended to provide the robot with the ability to maneuver in tight spaces and execute complex movement patterns with greater precision. However, we faced significant integration challenges with the mecanum wheels. Specifically, it was difficult to connect them to the 500 RPM motors due to incompatible mounting mechanisms. This issue led to inefficiencies in transmitting power from the motors to the wheels, compromising performance.

As a result, we switched to standard wheels, which proved more straightforward to implement. The standard wheels provided sufficient traction and stability, allowing the robot to navigate the game environment effectively. Although this decision limited the robot's agility compared to using mecanum wheels, it significantly reduced complexity and increased reliability.

Steering with Differential Drive

The differential drive system, powered by two independently controlled motors, was intended to provide precise control over the robot's movement. In practice, we encountered challenges in steering accuracy. Small discrepancies in motor performance caused the robot to drift slightly from its intended path. This issue required fine-tuning of the motor control algorithms to minimize errors. Additionally, balancing the power distribution across the four wheels was critical to maintaining consistent movement.

Lessons Learned

- Switching from mecanum wheels to standard wheels simplified the mechanical design and avoided potential delays in fabrication and assembly. This experience highlighted the importance of considering component compatibility during the design phase.

- Challenges with steering underscored the need for careful calibration and testing to address mechanical inconsistencies in a differential drive system. Future designs may benefit from integrating feedback mechanisms, such as encoders, to enhance control accuracy.

Electrical Design

Intended and Actual Performance

Motor Drivers and Differential Drive Challenges

The intended goal for the motor drivers was to achieve smooth and precise control of the robot's differential drive system. The differential drive was chosen for its simplicity in design and control. While the system performed adequately in simpler maneuvers, it introduced significant challenges during wall-following and autonomous target attacking. Maintaining a consistent trajectory required careful tuning due to differences in motor performance and difficulties with fine-tuned control.

ESP32-C3 and ESP32-S2 Wi-Fi Communication

Initially, we used two ESP32-C3 modules for programming various motors and sensors. However, we encountered persistent issues with unstable Wi-Fi connections, particularly during high-load operations. After troubleshooting, we switched to the ESP32-S2, which proved more reliable for our specific use case, since we did not have to communicate between the ESPs using ESPNOW, thus providing stable Wi-Fi transmission.

Sensors

- **Time-of-Flight (ToF) Sensors:** The ToF sensors were intended for precise distance measurements during wall-following and target acquisition. These performed as expected, providing reliable distance data that allowed the robot to follow walls effectively.
- **Vive and Photosensor:** The Vive and photosensor combination was implemented for effective localization. While the circuit worked well on its own, integrating this with the entire bot circuitry and code proved challenging. Later, we found out that all the tasks could be performed quite easily without using Vive.

Lessons Learned from Failures

ESP32-C3 Wi-Fi Issues

The primary lesson learned was the importance of thoroughly testing hardware communication in realistic conditions early in the design phase. The switch to ESP32-S2 resolved the connectivity issues, but it taught us the need for trying out different things to figure out what works best.

Differential Drive Limitations

The difficulties encountered during wall-following and target attacking highlighted the importance of mechanical design in conjunction with electrical systems. Precision control of a differential drive system is too complex, and we underestimated the impact of these challenges during initial design planning.

Autonomous Attack

While the ToF and Vive sensors provided accurate data, integrating them with the entire circuit for autonomous attack proved difficult. Real-world testing revealed limitations in the robot's responsiveness, particularly when maneuvering toward multiple targets within the 45-second timeframe.

NOTE: Electrical schematics are included in the appendix to provide detailed diagrams of the circuit design and sensor integration.

Processor and Code Architecture

MCU Connections

We only had one MCU, the ESP32-S2. The connections are shown in Figure 1.

Software Approach

This subsection describes the logic for each functionality and how the integration was done.

Autonomous Behavior

1. **Target Attacking:** The movement strategy in this code is designed to navigate the robot toward a target position while avoiding obstacles.

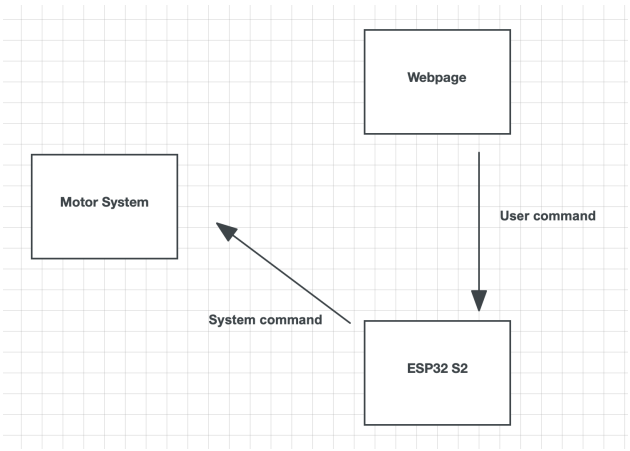

Figure 1: MCU Logic Connections

The robot continuously compares its current position, obtained through the Vive tracking system, to a target position. The differences in the x and y coordinates determine how the robot should adjust its movement to approach the target. ToF sensors measure distances to walls or obstacles on the left, right, and front of the robot. If the front distance is below a predefined threshold, the robot moves backward until the front is clear. If the left or right distance falls outside the safe range, the robot adjusts its path to correct its alignment (e.g., turns left if the right side is too close and vice versa).

The robot evaluates whether the difference in x or y coordinates is larger. If the x difference dominates, it prioritizes turning to align itself horizontally; otherwise, it moves forward to reduce the y difference. If both differences are within a small threshold, the robot assumes it is near the target and stops. Sensor readings and position updates occur in a loop. The robot continuously recalculates its strategy based on new data, ensuring it reacts to obstacles or positional changes in real time.

2. **Wall-Following:** The logic in this code relies on two VL53L0X distance sensors, one on the left side and one in the front. The robot moves forward by default and adjusts its motion based on the sensor readings to maintain a desired distance from the walls.

   The front sensor detects obstacles ahead. If the measured distance is less than the threshold, the robot moves backward until the obstacle is sufficiently far away. The left sensor ensures the robot stays within a specific distance range from the wall on its left side. If the distance is less than a minimum value, the robot adjusts to the right. If the distance exceeds a maximum value, it adjusts to the left.

   While adjusting left or right, the robot performs specific motor actions to turn in the desired direction. The sensor readings are continuously monitored, and adjustments stop once the distance is within the acceptable range. If no adjustments are required, the robot continues moving forward.

Manual Control

1. **Webpage:** The HTML interface includes buttons for directional control (Forward, Backward, Left, Right, Stop) and a Servo button. These buttons are designed to send specific commands to the ESP32 web server. The JavaScript logic handles user interactions by sending HTTP GET requests to the ESP32's web server when a button is clicked.

2. **Motion and Control:** For our bot, we did not use PID control because our directional motion was very stable. The corresponding endpoints are managed based on the user input. For better weapon accuracy, once the servo button is pressed, the motors come to a stop before the weapon is swung.

Localization

Using the sample Vive codes given for getting coordinate output, we obtained x and y values at various points on the field. We recorded values for each target on the field, giving them a buffer area in order to calculate the threshold at which the bot has to stop.

I2C

Using the sample master and slave codes given, we added logic to the master code to send packets for every 5 WiFi packets sent or received (as per the count given in the lecture slides). A global variable is used to keep track of the number of WiFi packets sent or received by the master ESP32. Inside the loop, the count is checked every time and at the required frequency of 2Hz.

Overall Integration and Issues

The integration involves combining the autonomous codes with the manual control and webpage. Separate handlers are given for each major functionality - target attacking, wall following and manual control. In the same code, I2C communication is initalized and every handler updates the WiFi packet sent/received count.

The combined I2C and WiFi codes, despite working together during testing, did not work together on the day of evaluation. The problem was likely with the WiFi channel that we were using. The issue could also have been with some pins on the ESP32-S2 not being compatible with what we were using them for (the datasheet was consulted). Although we got accurate coordinate values using the Vive tracking system, we could not solve the issue of our sensors malfunctioning in the end and were unable to integrate localization into the main code.

Retrospective

Gia Dcosta

Most Important Lessons Learned

One of the most important lessons I learned during this project was the value of iterative design. Encountering challenges such as integrating mecanum wheels emphasized the importance of flexibility in decision-making and adapting to practical constraints. I also gained a deeper understanding of mechanical systems and how to optimize them for reliability and functionality.

Best Parts of the Class

The best part of the class was the hands-on experience of designing and building a fully functional robot. The iterative nature of the project, combined with the collaborative environment, made the learning process both challenging and enjoyable. The excitement of testing our robot in gameplay scenarios was particularly rewarding.

Biggest Challenges

The most significant challenge was steering accuracy with the differential drive system. Small discrepancies in motor performance created difficulties in maintaining a straight path, which required extensive calibration and testing. Additionally, the initial struggles with mecanum wheels consumed valuable time and resources before transitioning to a more practical solution.

Suggestions for Improvement

One area for improvement would be providing more structured guidance or tutorials on integrating complex mechanical components such as mecanum wheels. Additional resources or access to alternative mounting solutions could save time and enhance the learning experience.

Thoughts on Gameplay

The gameplay aspect of the project added an exciting and competitive element to the class. It provided a clear goal and a context for applying the skills we learned. The rules and objectives were well-defined, but I believe incorporating more dynamic gameplay elements could make the experience even more engaging.

Additional Thoughts

Overall, this class was an incredibly valuable experience. It combined theoretical knowledge with practical application in a way that felt impactful and relevant. I appreciated the opportunity to work as part of a team, tackle real-world engineering challenges, and see our design come to life.

Saayuj Deshpande

Most Important Lessons Learned

I think the most important lessons taught to me by the final project and the course in general, is to expect the unexpected. We had our entire bot assembled and ready to test, and then when we connected the battery to the entire ESP32 S2 circuit, everything got fried. It was disheartening, but we still pulled through. it also taught me the importance of perseverance, determination and time management. On the technical side, I also headed the electrical part of the project and Lab 4, which was an enriching experience. I also learnt mechanical design throughout this course, which I had not explored before.

Best Parts of the Class

The best part of the class for me was hands-down the final project. To assimilate everything we had learnt throughout the course felt very satisfying. I particularly enjoyed making the circuits, soldering them, and also writing the logic for wall following and autonomous movement. Lab 3 Waldo was also a favorite of mine.

Biggest Challenges

Lab 2 was very tough/time intensive. The circuit just was not working, how much ever I tried. Somehow, I got it to work after a long time. In Lab 4 and the final project, debugging hardware issues was very tough. Even though the logic and code made sense, things never worked the first time on the hardware. It took several iterations to get it right. For me, mechanical design was also pretty rough. I did not know Solidworks, so I had to spend a lot of time on it.

Suggestions for Improvement

The course content and the labs are phenomenal in my opinion. But I have one complaint/suggestion - lab 2 was not used anywhere in the final project. Maybe that could be somehow utilized or another lab could be introduced? Also, I would have liked more transparency on the order placement system through PEFS, etc. as we had a lot of issues and delays with our orders.

Thoughts on Gameplay

I cannot think of anything better for the gameplay. It was challenging, but at the same time, interesting. Maybe some greater incentive to make the bots autonomous could be given though.

Additional Thoughts

I would love to be associated with this beautiful course in some way or the other, maybe in the form of a Teaching Assistant in the future.

Samhitha Vedire

Most Important Lessons Learned

The most important lessons learned for me were anticipating unexpected challenges and solving them with what is available in the moment. During our assembly process, we faced multiple small challenges that made us look at our inventory and decide what the best possible decision was. Similar situations during testing made me appreciate the importance of adaptive decision making.

Best Parts of the Class

The best part of this class was easily the fact that the project was a fully integrated mechatronic system. It solidified the design, assembly and coding process in my head and I have gained a deeper understanding of how intricate such systems are in the real world. I was able to step out of my pure software background and gain basics in electronic and mechanical sytems.

Biggest Challenges

The biggest challenge was debugging hardware issues after the bot assembly was done. It was a tedious task to take apart parts of the robot when we didn't know what part of the circuit failed, and finding the point of failure was extremely time-consuming.

Suggestions for Improvement

One area of improvement would be providing more live demos during office hours, with respect to every new electronic and mechanical component introduced in class. It would make the learning curve a little less steep and save time during the building process.

Thoughts on Gameplay

The gameplay was very exciting. Even though our team did not compete, the final competition added a much needed respite from regular academics and deadlines and I appreciated it a lot. The rules and constraints were well made and the game was an apt culmination to the entire class.

Additional Thoughts

I have really loved being a part of this course. It has given me the exact combination of knowledge I sought out at the beginning of the semester and aligns perfectly with my areas of interest. I have learned a lot of lessons, both academic and personal.

Appendix

Code

Algorithm 1 Robot Control and Wall Following Algorithm

```
Initialize system components: WiFi, I2C, sensors, motors, and servo
Set AP network credentials (ssid, password, local_IP, etc.)
Configure motor control pins and PWM properties
Set up web server endpoints for motor and servo control
Initialize distance sensors with unique addresses
Start the web server
while true do
    Check for client requests on the web server
    Perform periodic I2C communication with the slave device
    if response from slave indicates stop signal then
        Stop all motors
        Enter error state with LED indication
        while true do
            Blink red LED
        end while
    end if
    Handle web server requests:
    if '/update' endpoint is accessed then
        Extract direction argument and call setMotorSpeed(direction)
    else if '/servo' endpoint is accessed then
        Stop motors
        Sweep servo angle from 0 to 180 and back
    else if '/targets' endpoint is accessed then
        Log and update packet count for target handling
    else if '/wall' endpoint is accessed then
        Perform distance adjustments:
        If front distance < threshold, call moveBackward()
        If left distance < minimum, call adjustRight()
        If left distance > maximum, call adjustLeft()
    end if
end while
```

Algorithm 2 Autonomous Target Attacking

Initialize:
Set up sensors: LEFT_SENSOR, FRONT_SENSOR, RIGHT_SENSOR
Define thresholds: MIN_DISTANCE, MAX_DISTANCE, FRONT_WALL_THRESHOLD
Define PWM signals and movement directions
Set target coordinates $(x_{\text{target}}, y_{\text{target}})$
Initialize current coordinates $(x_{\text{current}}, y_{\text{current}})$
function MEDIANFILTER(a, b, c)
    Sort a, b, c and return the middle value
end function
procedure SETUPSYSTEM
    Initialize sensors and motor pins
    Start Vive tracking system
    Begin forward motion
end procedure
procedure MAINLOOP
    while True **do**
        Read sensor distances: LEFT, FRONT, RIGHT
        Update current position using Vive system
        if ISNEARTARGET **then**
            Stop motors
            Exit Loop
        end if
        RECALCULATEMOVEMENT(LEFT, FRONT, RIGHT)
        Wait for 100ms
    end while
end procedure
function RECALCULATEMOVEMENT(LEFT, FRONT, RIGHT)
    if FRONT < FRONT_WALL_THRESHOLD **then**
        Turn left or right based on distances
    else if LEFT < MIN_DISTANCE **then**
        Adjust robot to move right
    else if RIGHT < MIN_DISTANCE **then**
        Adjust robot to move left
    else
        Continue moving forward
    end if
end function
procedure UPDATEPOSITION
    if Vive system is receiving data **then**
        Apply median filter to smooth position
        Update $(x_{\text{current}}, y_{\text{current}})$
    else
        Reset position to default
        Attempt to re-sync Vive system
    end if
end procedure
function ISNEARTARGET
    Compute differences: $x_{\text{diff}}, y_{\text{diff}}$
    if $x_{\text{diff}} < \epsilon$ **and** $y_{\text{diff}} < \epsilon$ **then**
        return True
    else
        return False
    end if
end function

Bot Photos

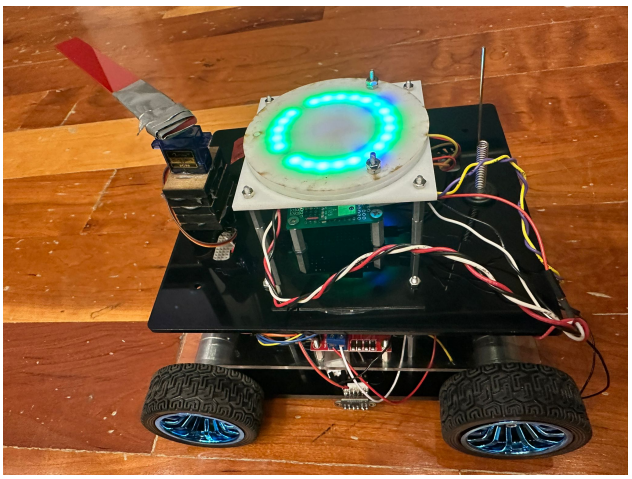

Figure 2: Bot View 1

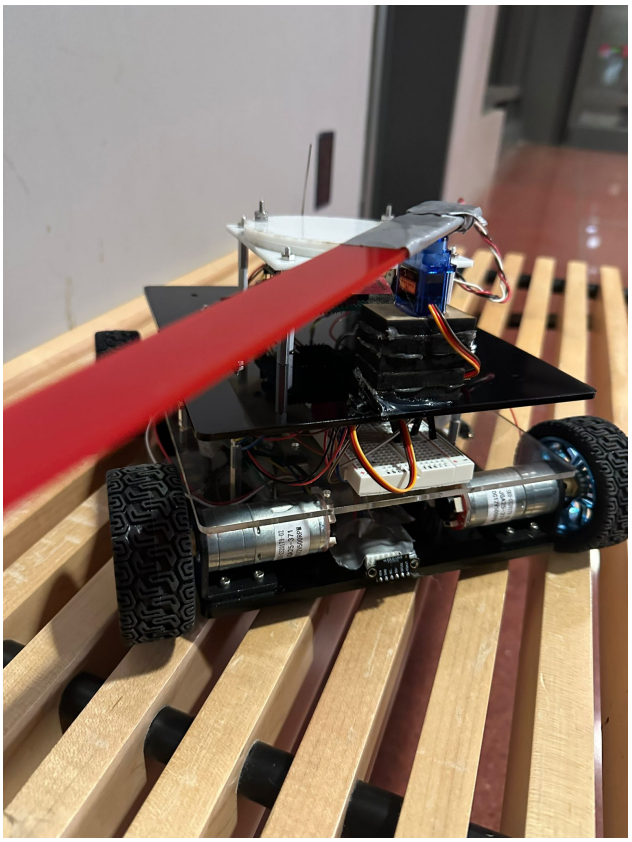

Figure 3: Bot View 2

Bill of Materials

Serial No.	Description	Material/Component	Quantity
1	Motors	Garosa 12V Encoder Gear Motor	4
2	Wheels	Plastic + Rubber	4
3	Chassis	Acrylic	1
4	Battery	Zeee 3S Lipo	2
5	ToF Sensors	VL53L0X	2
6	Photosensors	PD70-01C/TR7	2
8	Whisker Switch	ME-8169	1
9	Servo Motor	SG90	1
10	Microcontroller	ESP32-S2	1

Table 1: Bill of Materials

CAD Drawings

The three plates we laser cut to assemble our bot are given below. The links to the .dwg files are also given.

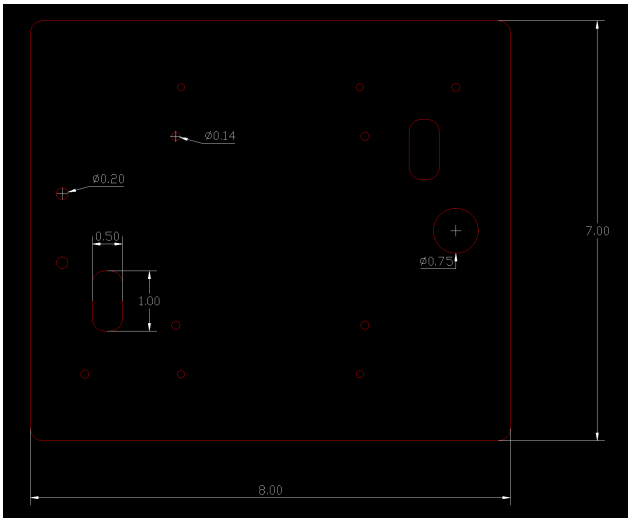

Figure 4: Top Plate

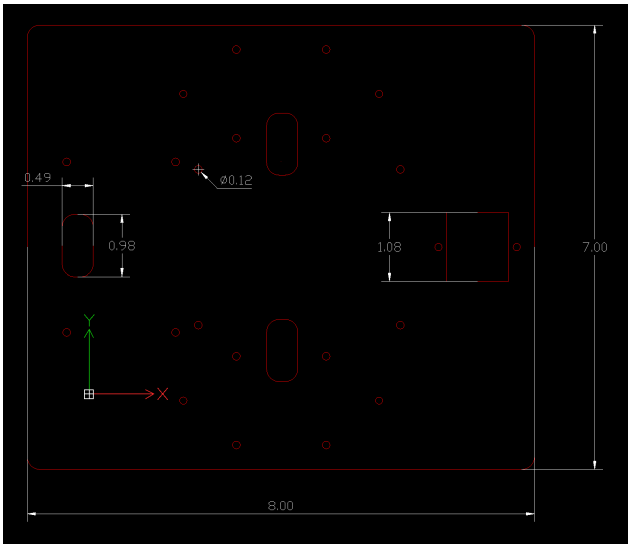

Figure 5: Middle Plate

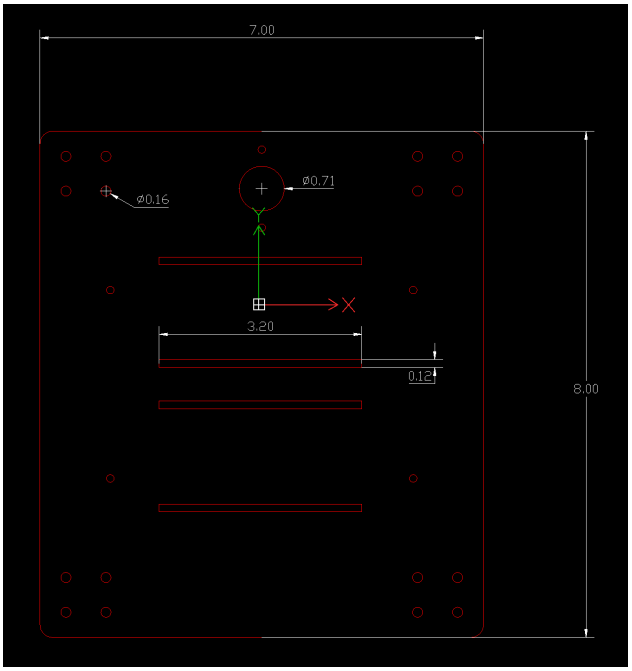

Figure 6: Base Plate

Circuit Diagrams

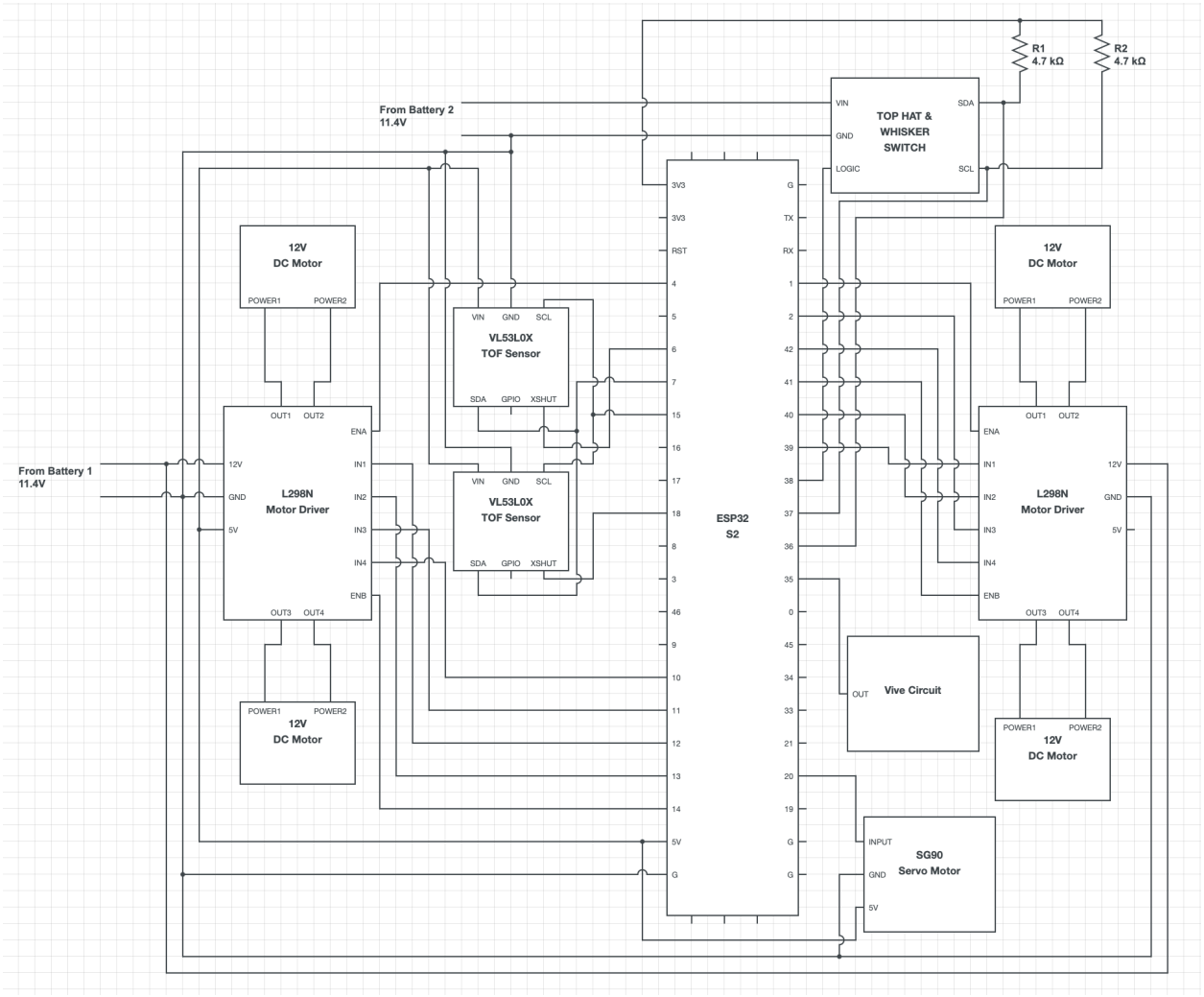

Figure 7: Circuit diagram of entire bot connections

NOTE: Vive circuit and Top Hat circuit as given in class.

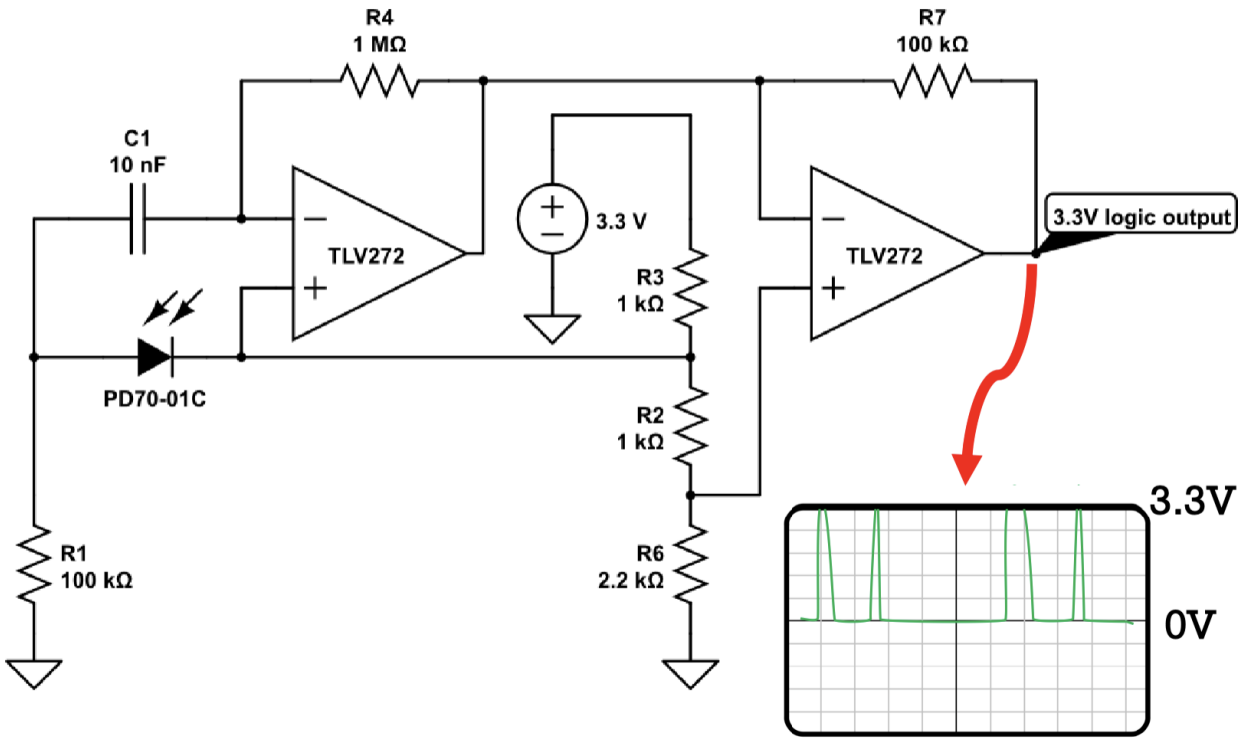

Figure 8: Vive circuit

Datasheets

The datasheets/purchase links for the components we used are given below:

1. Motors: Amazon link for motors
2. Battery: Amazon link for batteries
3. ToF Sensors: Amazon link for sensors, Datasheet
4. Photosensors: Datasheet
5. Whisker Switch: We used the specifications given on Canvas.
6. Servo Motor: Amazon link for servo motor, Datasheet
7. ESP32-S2: Datasheet

Video Links

https://youtu.be/kRGAJ2Ujurk