MEAM 5100 Final Project- Team 5

Kevin Paulose, Tejendra Patel, Saurav Agrawal

January 3, 2024

1 Introduction

1.1 Overall Functionality

The final project was a cohesive product of learnings from all labs and lectures of MEAM 5100. We aimed to develop a four-wheel mobile base robot that should be capable of wall-following the circuit, tracking the IR LED beacon and pushing the police car amongst other auxiliary tasks.

Speaking of the the design choice made by our team, we made a mobile base capable of holonomic motion providing omnidirectional motion. We used mecanum wheels for this innovation. The brain of the robot was the ESP32 S2 microcontroller which handled all logical inputs and outputs of the robot. We used two ATMega32u4 (ItsyBitsy) microcontrollers in the beacon circuit coupled with phototransistors which helped us track the IR LED Beacons. Other major components in play were ultrasonic sensors, TB6612FNG motor drivers, yellow TT motors, PD70 photodiodes, a 2200 mah 14.8V 4S LiPo battery and a Miady 5000 mah power bank.

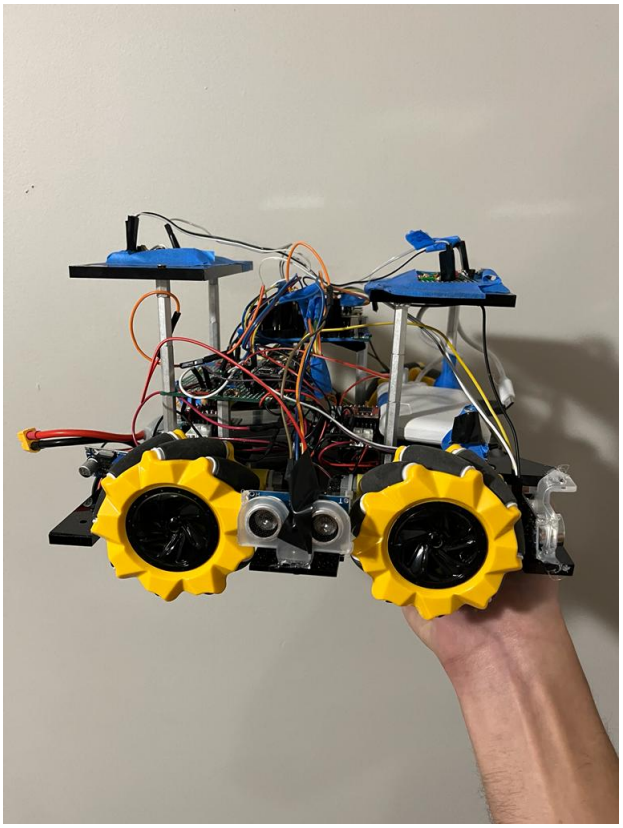


Figure 1: Final project robot- four-wheeled mobile based using mecanum wheels

1.2 Detailed Approach

1.2.1 Robot control

On the high level, we used an HTML webpage to commandeer the vehicle and communication was made possible using WiFi, i.e., via UDP and ESP-NOW protocols. The ESPS2 microcontroller and our laptop were connected to GM Lab's router and the communication was done via UDP over STA mode by hostingthe webpage on a local IP address (http://192.168.1.170/). Clicking the buttons on the webpage relayed information to the ESP32 S2 and the required PWM to the motors and other logical high-low signals were handled and the desired motion was output.

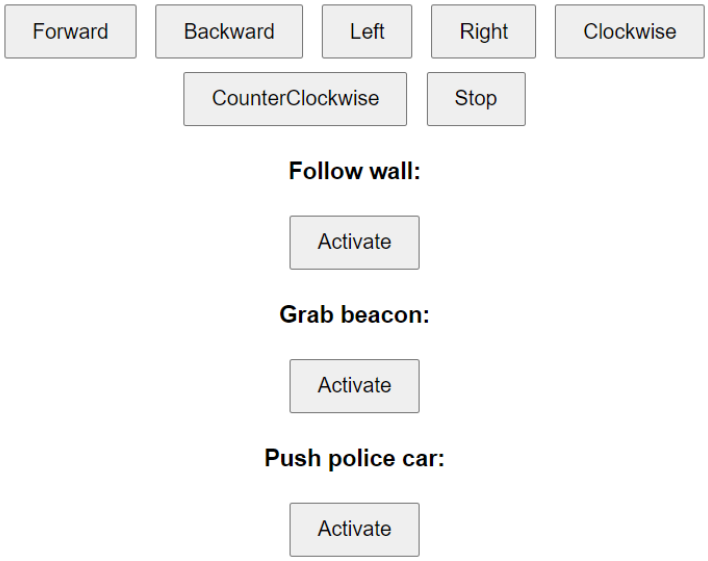


Figure 2: Website used to commandeer robot

1.2.2 Holonomic Motion

Contrary to the conventional wheels which offer motion in only 1 DOF on the horizontal plane- four cardinal directions (forward, backward, left, right), mecanum wheels novelize omnidirectional motion, i.e., provide 2-DOF motion capability to a mobile base. This is made possible through the many rollers each wheel has that are aligned at 45° to the wheel's axis. Using these wheels is advantageous as the motion is optimized by unlocking a new set of directional motions which saves time and offers robustness.

Figure 3: Mecanum wheels

Mecanum wheels can essentially make the robot move in diagonal directions, lateral left/right directions along with the usual forward backwards right left directions. More forms of motions can be unlocked by

using a combination of directions which can for example make the robot spin 360° about an axis without any translation.

By giving different motor speeds, i.e., different PWM values to each wheel, we can have our desired motion. Some of the popular combinations of motor speeds and corresponding directions are as follows-

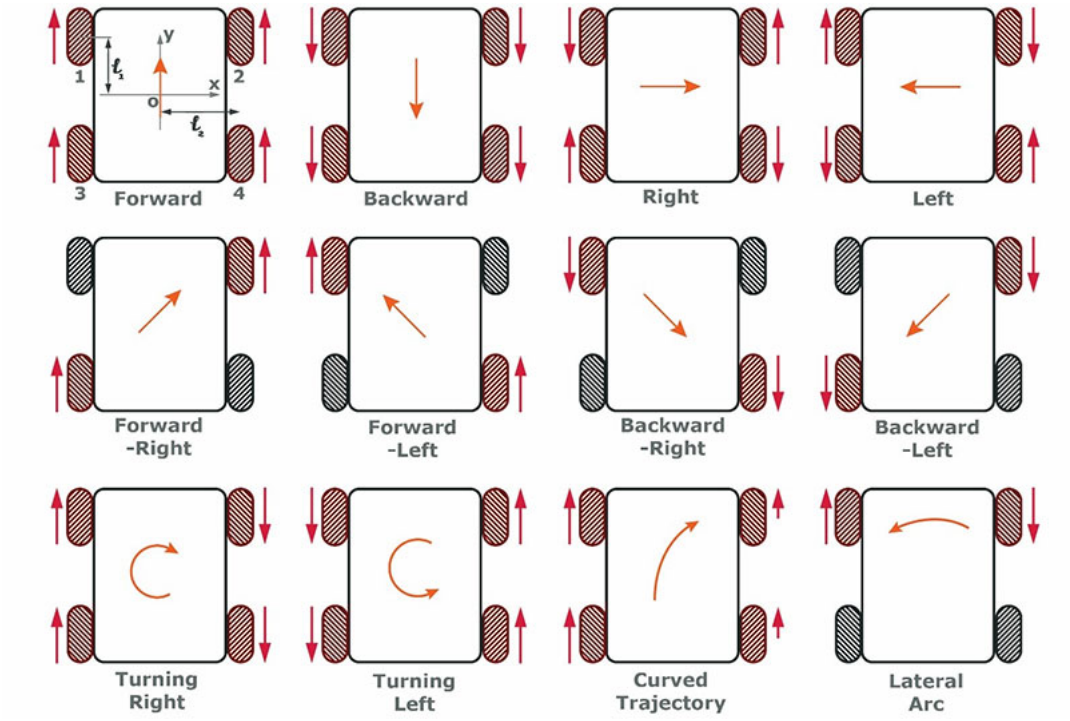
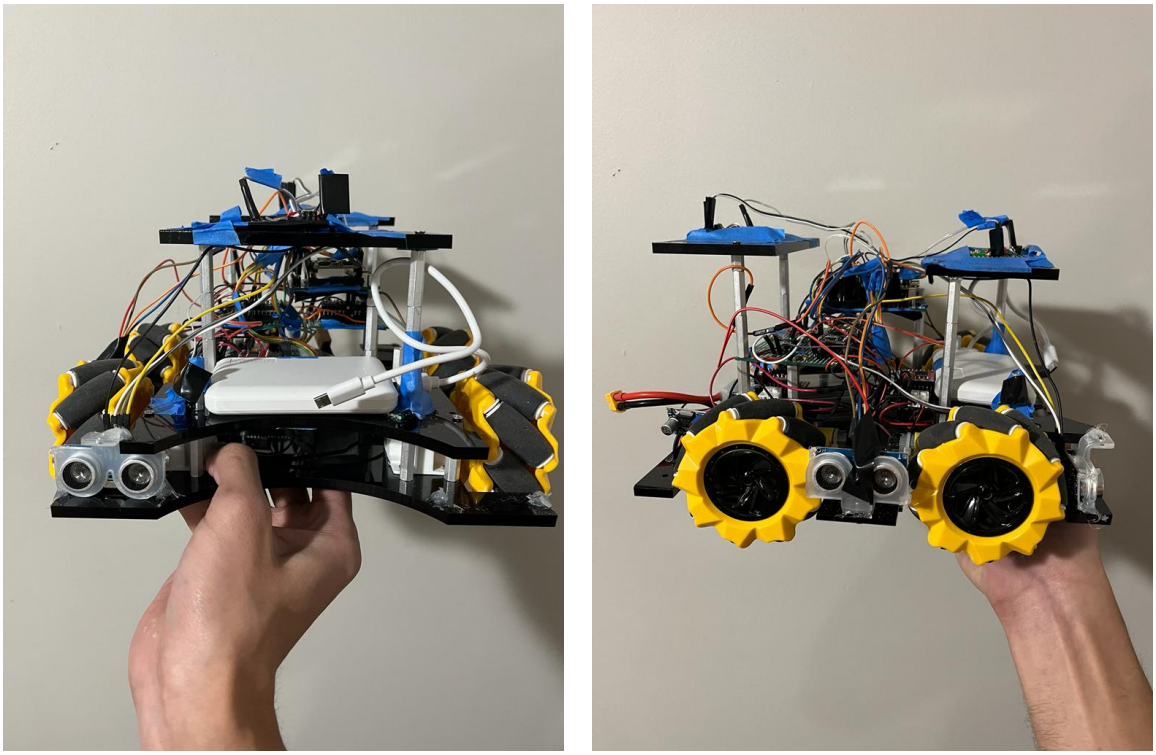


Figure 4: Mecanum wheels directions map

A note to using mecanum wheels is to mount the wheels in the right orientations. The top left and the bottom right have the same orientation while the top right and the bottom left have the same orientation but mirror to the other two.

1.2.3 Wall Following

Keeping graded evaluation priority, we planned making a minimalistic yet robust robot. So, for this task we used range sensors on the front and right side only and motioned the robot to repeat a "go straight take left" motion strategy to wall follow the GM lab circuit. We initially used the VL53L0X Time of Flight sensors. We loved the compact design of the sensor and its range-sensing accuracy, but they failed soon after using them for some time and we suspect it was due to its cheap quality. We immediately shifted to using ultrasonic sensors and surprisingly we received very steady readings (as opposed to our belief of receiving noisy readings). Two ultrasonic sensors were mounted- one at the front and one at the right side of the robot. This way we could successfully do wall following for three full circuits.



(a) Front side ultrasonic sensor

(b) Right side ultrasonic sensor

Figure 5: Ultrasonic sensors on our robot

1.2.4 IR Beacon Tracking

We used our beacon tracking circuit from lab 2 here. The IR LED was detected using an LTR-4206 phototransistor amplified by TLV272 opamp, whose output was fed into an ATMega32u4. The ATMega32u4 had beacon detection code flashed onto it and using GPIO pins PB5, PB6 of the ATMega32u4 logic high and logic low were sent to the ESP32 S2 for further use. We used two IR beacon trackers to detect the 23Hz IR beacon and 550Hz IR beacon, and this was inspired by the bang-bang beacon tracking explained in class by Professor Yim. Instead of using a servo, we made our robot spin 360° and whenever both beacon tracking circuits (i.e. both phototransistors) returned logic high, the bot would go forward a little bit and scan again. And this would go on until it reaches the beacon.

To tackle the exceptional case of the phototransistors being too far from the beacon, we initialised a sweeping code where the robot would have a lateral shift (move right or left away from the wall in a horizontal direction) and forward motion to move closer to the beacon. Once it detects the beacon, the spinning loop is immediately started and beacon tracking is successfully done.

1.2.5 Vive localization and UDP

We soldered the Vive circuit on a perfboard following Professor Yim's implementation to use the HTC Vive for robot localization. Going one step ahead, we used two identical circuits using the two PD70 photodiodes given to us, one at the front of the robot and one at the back of the robot. The mean of the two photodiodes' readings would be the centre of the robot and we could more importantly calculate the orientation of the robot concerning the Vive emitter. Our Vive location was continuously broadcasted to the GM lab router via WiFI using UDP protocol.

1.2.6 Police car push

Since the police car was sending its vive location via UDP, we received the vive location via UDP itself. We only used the first value received from the vive and locked onto it. Then, we swept along the Y-direction until when the robot was head-on with the police car and then the robot was programmed to go straight at $\tilde{1}50$ PWM

(highest being 255 PWM). We were amazed by how powerful (in terms of torque and RPM) the yellow TT motors were as we managed to push the police car from the centre to the end of the circuit wall.

1.3 Performance

1.3.1 Motor control

We achieved a very stable and accurate motion of the robot with our chassis design, motor mounts and payload balance. We did not need to implement a PID controller for motor velocity. However, to ensure sustained manoeuvrability, we implement a position PID using vive readings. Using this position PID, we adjusted the PWM values of the motor for our robot's desired trajectory. We were able to achieve excellent speed and torque from the yellow TT motors from our setup.

Video demonstration of police car push- Police car push task

1.3.2 Wall following

As mentioned before, we used a "go straight take left" motion strategy. The ultrasonic sensors on the front and right have very stable values and we defined an error within which the robot should move away or turn from the wall. For example, if both sensors are within the error range it is obvious for the robot to turn left and go straight until only the right sensor is near the error range. We could easily do three circuits of the track's wall.

Video demonstration of wall following- Wall following task

1.3.3 IR detection range and field of vision

We had a very powerful beacon tracking circuit which could detect an IR LED from 2 feet away and the phototransistors also had a wide peripheral vision for IR LED detection. This was advantageous to us as we were spinning our robot about its axis (like the servo in bang-bang beacon tracking) to detect the beacon.

Video demonstration of beacon tracking task- IR LED Beacon tracking task

1.3.4 Communications via ESP-NOW

We were able to receive vive locations on both our vive circuits and we were also able to send our vive location via UDP. We were also able to transmit communications packet data from our website-WiFi setup after integrating the robot to make it autonomous. However, there was lag when commanding the robot using the website using UDP while simultaneously transmitting data using ESP-NOW. We could not fully solve this issue but were able to reduce lag by simply keeping the robot connected to the WiFi network for a longer duration.

2 Mechanical Design

2.1 Frame design

The chassis was laser cut from a single sheet of acrylic material (0.25 inches thickness), providing a sturdy yet lightweight foundation for the robot. Holes were made in the CAD model which serves the purpose of attaching support structures, such as brackets and standoffs, to add more layers to the base for subsequent mounting of perfboard circuits and microcontrollers. Incorporating this support structure ensured the robot gains increased resistance to vibrations and shocks, ensuring the reliable performance of internal components during movement and operation. This approach not only strengthens the chassis but also provides a modular and versatile platform for mounting perf boards and other electronic elements. The entire chassis has the dimensions of a compact box measuring 25 cm x 22 cm x 15 cm, ensuring it can navigate efficiently in confined spaces.

2.2 Wheels

Using mecanum wheels consists of a trade-off between ease of integration and mobility. By manipulating the speed and direction of each wheel independently, the bot can execute complex movements, making it highly maneuverable. For our ease we coupled the PWM signals of top right-bottom left wheels and top left-bottom rights wheels.

2.3 Motor mounts

During lab 4 we realized the need to use motor mounts as the yellow TT motors after being mounted with the mecanum wheels were very frail and shaky while using the robot. We had 3D print some motor mounts from RPL during lab 4 but they were quite fragile and didn't help us much. So we used a new motor mount design for our updated chassis and we 3D printed the motor mounts from Education Commons. One aspect we improved on was to request the prints to use maximum infill so that they are rigid and offer great support and stability. Motor mount's CAD model is given in sub-section 8.3.4 (Appendix).

Figure 6: White colored motor mounts used for greater stability to the motor and mecanum wheels

2.4 Design considerations

In the meticulous design of a compact and efficient robot, every detail has been carefully considered to optimize functionality within the confines of a 25 cm x 22 cm x 15 cm chassis. Employing a vertical stacking approach, circuits are strategically mounted on spacers, maximizing spatial utilization without compromising essential functions. This intentional placement not only ensures efficient use of limited volume but also contributes to a sleek and streamlined appearance.

A distinctive aspect of the wiring system is its meticulous colour-coding. Each wire, selected with a purpose, carries specific meanings, enhancing clarity during both assembly and troubleshooting. This systematic approach streamlines the identification of circuits, facilitating a more efficient design and maintenance process. Cable management techniques, including bundling and securing with precision-engineered Molex connectors, add to the tidy appearance and reduce clutter within the chassis.

The decision to employ soldered connections adds a layer of stability to the circuits, minimizing the risks associated with loose connections during vibrations or sudden movements. This comprehensive integration of vertical stacking, logical component placement, colour-coded wiring, and soldered connections defines the core principles of compact robotic design. The result is a robot that maximizes spatial efficiency and exhibits a high level of reliability.

3 Electrical Design

3.1 Overall circuit design

- One ESP32 S2 was the central and sole micro-controller commanding the robot
- Two TB6612FNG motor drivers were used to drive the four yellow TT motors
- Two ATMega32u4 (with beacon code flashed) were used in the two LTR-4206 phototransistor circuits respectively; with logic levels high and low relayed to the ESP32 S2 via PB5, PB6 GPIO pins of both ATMega32u4.
- Two ultrasonic sensors mounted on front and right side of the robot
- Two vive circuits with logic output relayed to ESP32 S2
- 3.3V and ground ports of ESP32 S2 used by the vive circuits, and 5V and ground of ESP32 S2 used by all other circuits
- 2200mah 14.8V 4S LiPo was used with a buck regulator to step down voltage to $\tilde{9}$.1V, for the motor driver circuit and also the beacon tracking circuit

3.2 Motor driver circuit

The motor driver circuit's connections were made as follows-

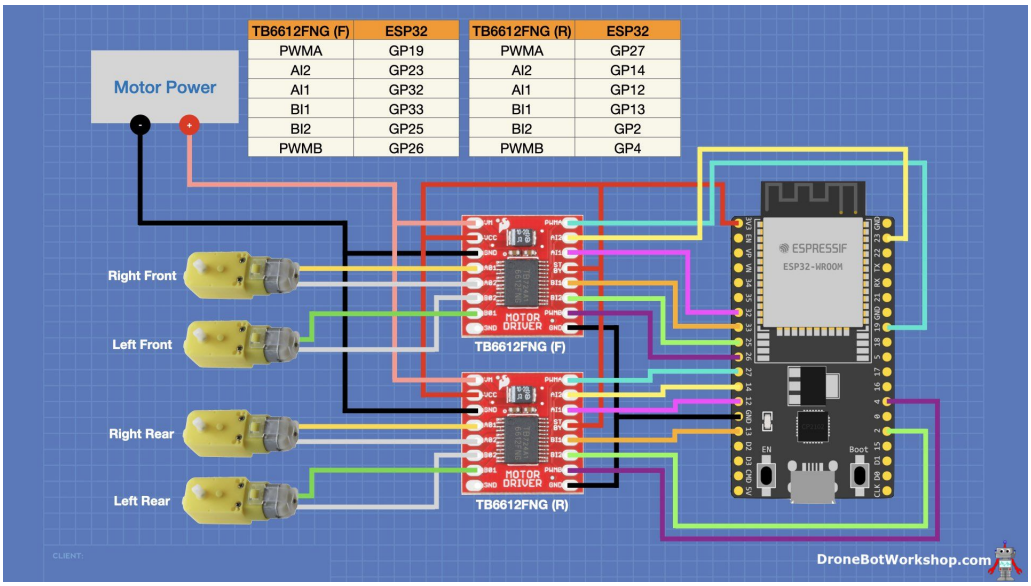
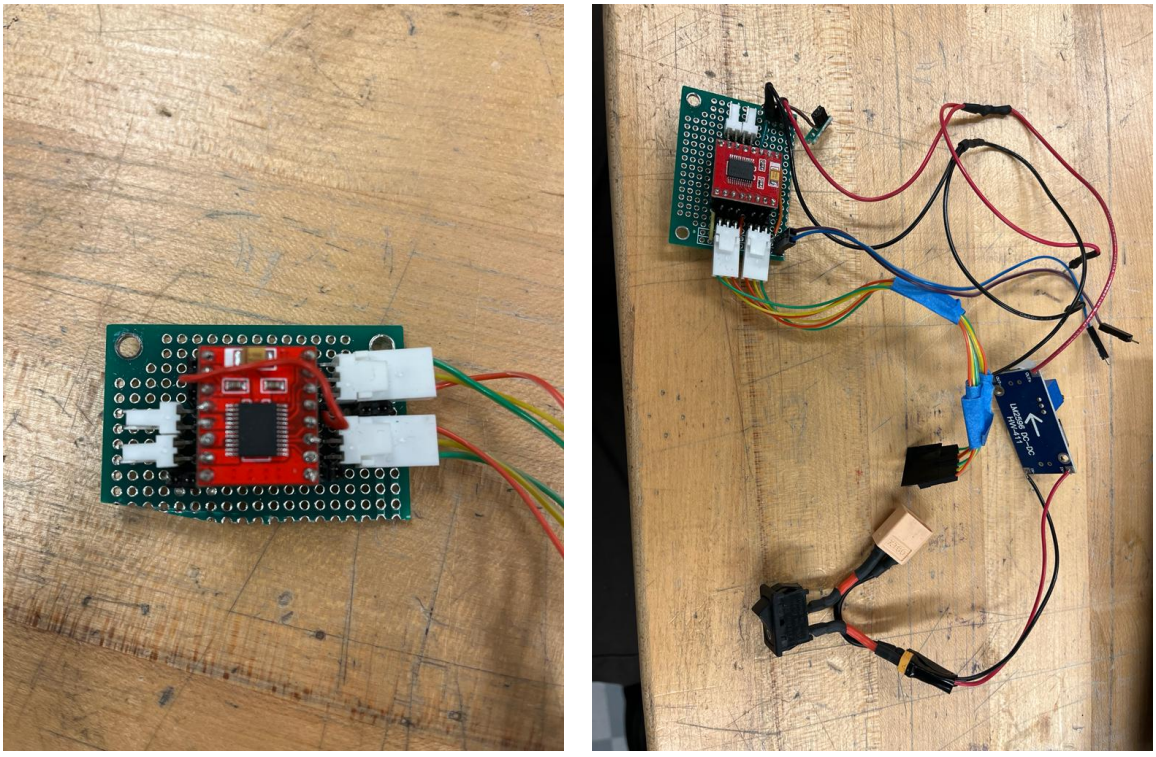


Figure 7: TB6612FNG motor driver-ESP32-motor wiring circuit

9.1V power voltage and ground from the LiPo battery (stepped down from 14.8V using a buck regulator), was supplied to the two motor drivers. Other 5V and ground connections were given from the ESP32 S2. The motors were connected as shown in the figure and PWM connections to the ESP32 S2 were also made.



(a) Motor driver-1 circuit

(b) Motor driver-2 connected to buck converter

Figure 8: Motor driver circuits

3.3 IR detection circuit

The ATMega32u4 worked independently from the rest of the circuit as it was powered from the LiPo itself via the VBAT port on the ATMega32u4 which can take up to 15V voltage. The beacon tracking code was flashed onto it and that gave the robot a higher degree of autonomy. A TLV272 opamp was used to amplify the LTR-4206's signal and finally, PB5, and PB6 GPIO ports were used to send the logic levels low and high respectively to the ESP32 S2 for further usage. We used two identical beacon tracking circuits for the two phototransistors.

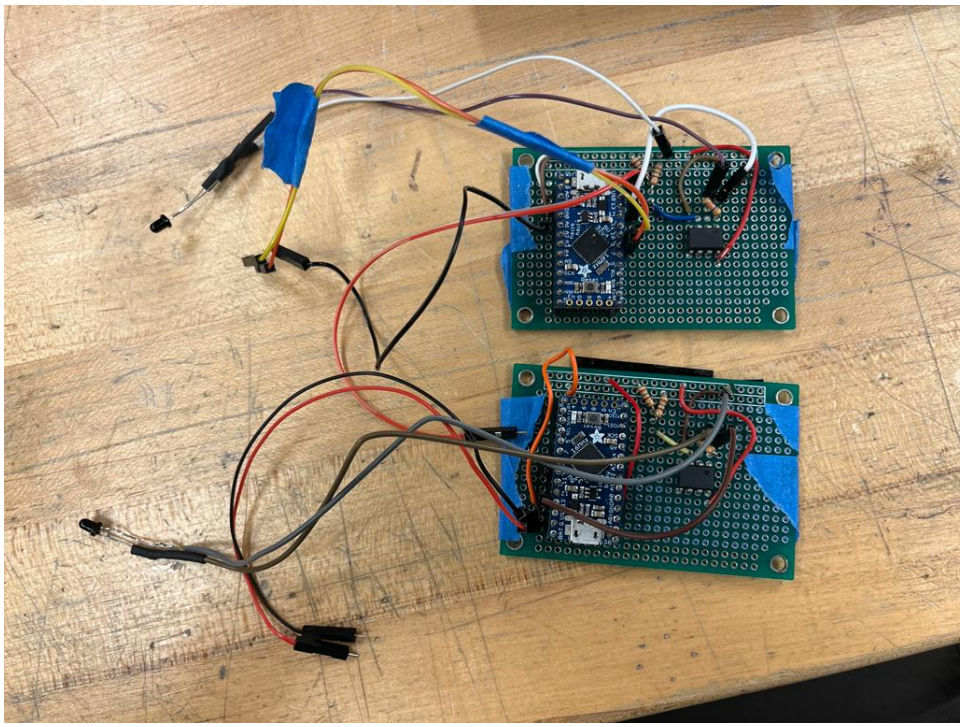


Figure 9: Beacon tracking circuits made using ATMega32u4, TLV272 opamp, LTR-4206 phototransistor and resistors

3.4 Vive circuit

We used two identical vive circuits as mentioned before. A single TLV272 opamp was used on each circuit to use the PD70 photodiode as per the implementation given in lecture slides. 3.3V was the main power supply for this circuit which was given from the ESP32 S2 directly. Two output, one from each vive circuit was fed into the ESP32 S2 for further vive data processing.

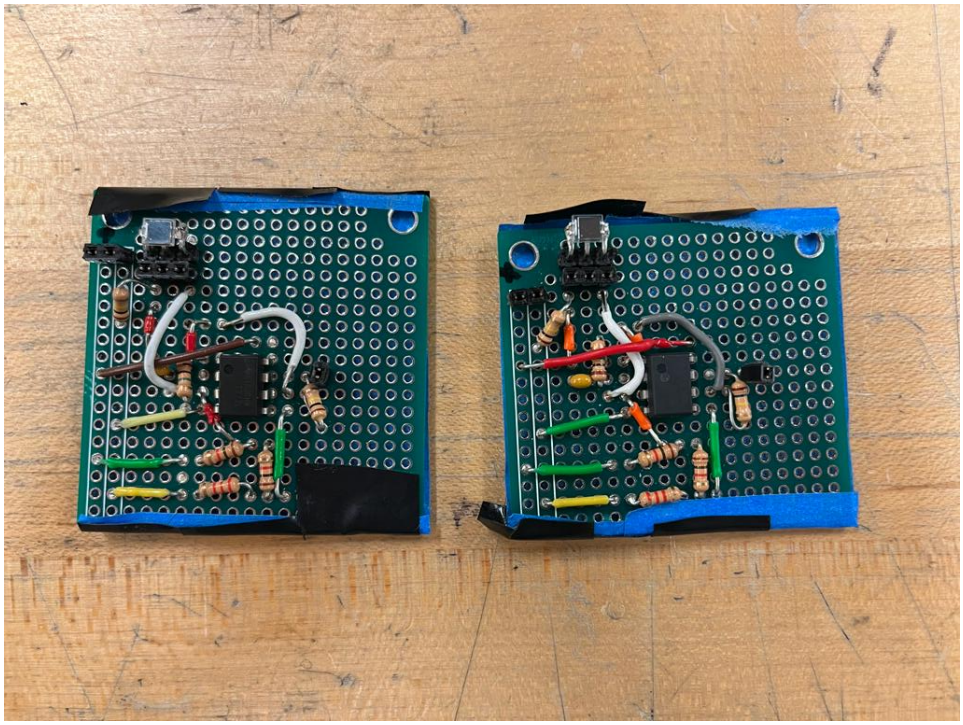


Figure 10: HTC Vive tracking circuits made using PD70 photodiode, TLV272 opamp and resistors

3.5 Sensor circuits

We used two ultrasonic sensors for the robot's navigation in the wall following task. Each ultrasonic sensor needed a 5V Vcc supply and ground both given from the ESP32 S2 and each sensor has two pins- trigger and echo, each connected to 2x2=4 GPIO pins of the ESP32 S2.

Detailed schematics of all circuits are mentioned in section 8.2 (Appendix).

4 Processor architecture and code architecture

4.1 Circuits and microcontrollers placements

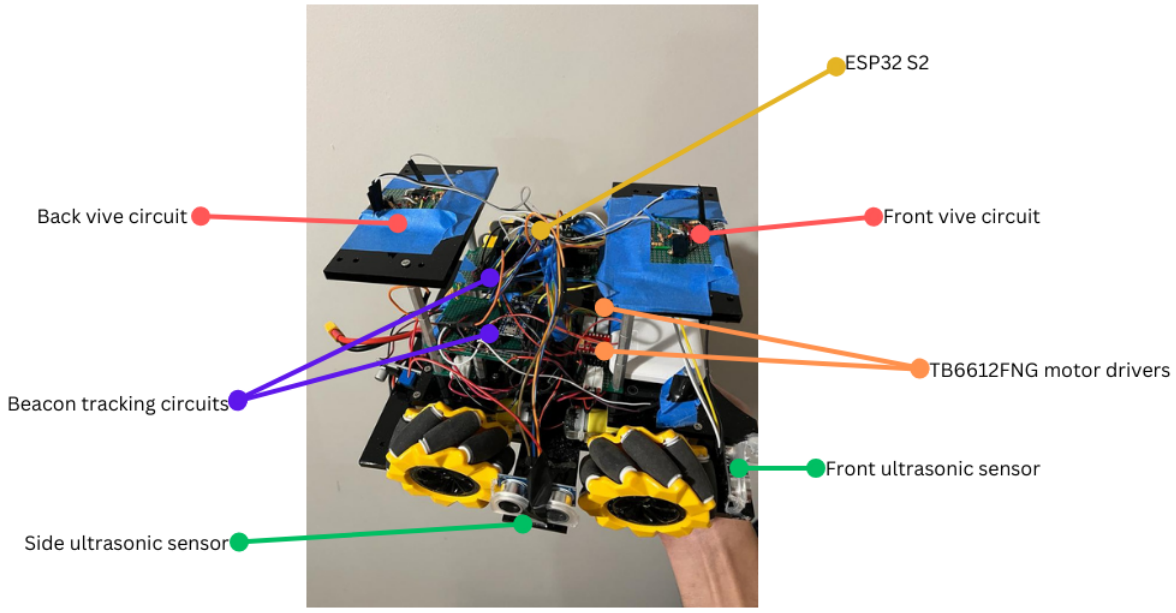


Figure 11: Robot with major component labels-1

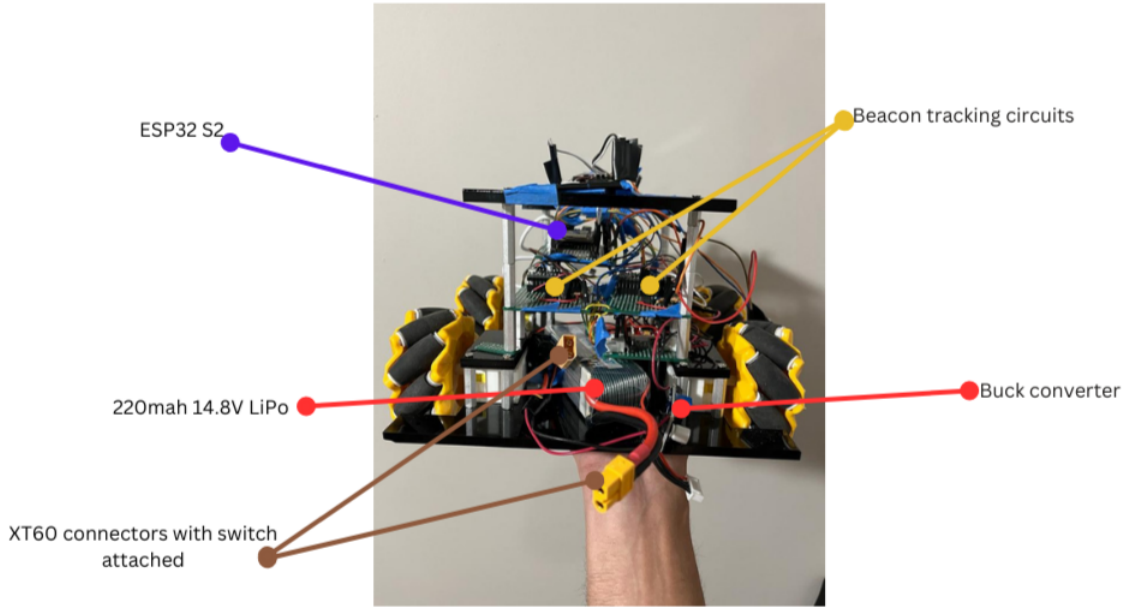


Figure 12: Robot with major component labels-2

4.2 Microcontroller architecture

All controls of the robot was handled by a single ESP32 S2. We soldered the perf board with extensions to mount the ESP32 S2 and using this we could get two whole rows of 5V and ground supply.

The ports of ESP32 S2 used by each circuit-

- Vive circuit-
  - 3.3V- 3v3 port of ESP32 S2
  - Ground- GND of ESP32 S2
  - Logic output- GPIO pin 40 and 41
- Ultrasonic sensors-
  - Vcc- 5V of ESP32 S2
  - GND- GND of ESP32 S2
  - Trigger & Echo- GPIO 21,20 and GPIO 34,33 respectively
- Beacon tracking-
  - VBAT of ATMega32u4- LiPo
  - GND of ATMega32u4- LiPo
  - PB5 of ATMega32u4- pin 1 and pin 38 of ESP32 S2
  - PB6 of ATMega32u4 - pin 2 and pin 39 of ESP32 S2
- Motor driver (TB6612FNG)-
  - Motor power (9V)- LiPo
  - Motor ground- LiPo
  - Motor 1 PWMA-PWMB pins- 4 and 7
  - Motor 2 PWMA-PWMB pins- 9 and 12
  - Vcc, StandBy pin of motor driver- 5V pins of ESP32 S2
  - AI1, AI2(Right front and right rear motors)- pin 10, 11 and 5,6 respectively for each motor
  - BI1, BI2(left front and left rear motors)- pin 14, 13 and 15,16 respectively for each motor

4.3 Software approach

A brief about the logical implementation of software has been explained for the four main tasks required in the final project. Extended pseudo-code has been mentioned in section 8.6 (Appendix).

4.3.1 Wall Following

The code for the wall-following with two ultrasonic sensors follows a structured approach. In *setup()*, pins, PWM channels, and serial communication are initialized. The *loop()* function continuously reads distances from the two ultrasonic sensors Sensor 1 and Sensor 2. The ***followWall()*** function calculates the wall-following error using PID control based on the target distances and sensor readings. The estimated error is then used in the ***calculateMecanumWheelSpeeds()*** function to determine individual wheel speeds for the mecanum wheels. Subsequently, the ***moveMotors()*** function controls the motor directions and PWM values based on the calculated wheel speeds. The ***stopMotors()*** function halts all motors and motor controllers. This process is iterated in the loop, ensuring continuous wall-following behaviour while dynamically adjusting motor speeds to maintain the desired distance from the wall. A conditional check for a specific distance from Sensor 1 also triggers a rotational movement using the ***moveMotors_dircontrol()*** function.

4.3.2 Beacon Tracking

The code for beacon detection begins with the setup and initialization phase, configuring the microcontroller's pins for motor connections and defining constants for motor control. The motors (Right Front, Left Front, Right Rear, Left Rear) are linked to specific pins of ESP32 S2, and the phototransistor's signal from ATmega32u4 pins (*Bleft_H, Bleft_L, Bright_L, Bright_H*) are integrated to detect the presence of a beacon. Motor control functions, namely ***moveMotors()*** and ***stopMotors()***, are established to control the movement and halt of the robot. The core algorithm in the loop section executes a beacon detection loop. The robot sweeps a particular area for beacon detection by moving in various directions (right, forward, left), periodically checking the states of phototransistors to identify the beacon's location. Once both phototransistors indicate a high signal, signifying the beacon's detection, the loop concludes, and the robot moves forward. If the beacon is not detected, the robot rotates counterclockwise. This code flow ensures that the robot autonomously follows the beacon's signal using the phototransistors and adjusts its movement accordingly.

4.3.3 HTC Vive

The code controls the robot with mecanum wheels using Vive trackers for positioning. It includes PID control for movement and rotation. The robot's position is adjusted based on target coordinates obtained from Vive trackers. Additionally, the code establishes a WiFi connection, sends UDP packets with positional information to a specific IP address, and handles synchronization and signal loss for the Vive trackers. The loop continuously updates and moves the robot while managing network communication and tracker status.

4.3.4 Police Car Push

The code is an extension of our position control code using HTC vive. The *setup()* defines hardware connections, motor configurations, PID constants, and movement modes. The main loop constantly tracks and adjusts the robot's position based on Vive tracker data. The robot's movement is controlled using PID-controlled mecanum wheel speeds. We take the first value of police car's vive location via UDP receive code (read on the ESP32 S2) and we make the robot sweep in the Y-direction until it reaches an error threshold near the center of the police car. Then the robot move forward only and keeps pushing the police car from it's center.

4.4 Performance issues

4.4.1 Sensor noise

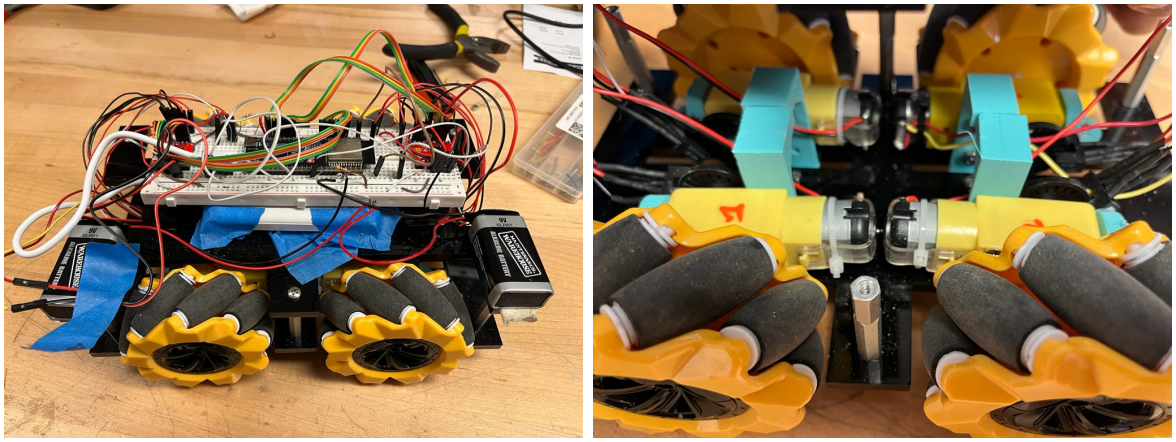
One of our greatest strengths was our circuits- vive circuits, beacon tracking circuits and motor driver circuits. We had done several iterations of soldering circuits on the perf boards in lab 4 which helped us perfect our circuits early on during the final project.

Another aspect of having reduced noise and loss of communications was the use of a single ESP32 S2 and no lost data over various data transfer protocols between different microcontrollers.

However, the robot still had little noise from the many dupont wires used for connections. Another source of noise was from the quality of the Time of Flight sensors and the ultrasonic sensors. We however managed to suppress the noise through more thorough soldering, insulation and proper wire connections through molex pins.

4.4.2 Vive timing

To address the issue of jittery Vive readings when the motors were turned on, a solution was implemented in the code. The strategy involved applying a moving average filter to the Vive input values. By smoothing out the fluctuations in the tracker readings, this filter helped to mitigate the adverse effects induced by motor operation. Additionally, fine-tuning the delay time proved to be an effective measure in enhancing the overall stability of the Vive tracker readings during robot operation.



(a) Lab 4 robot

(b) Lab 4 robot's base chassis

Figure 13: Lab 4 robot

5 Robot development timeline

5.1 First prototype- Lab 4

Our four-wheeled mobile base robot design was envisioned by us way back during lab 4 itself. We had bought the mecanum wheels and TB6612FNG motor drivers to build our robot in lab 4. We took this risk since we wanted to work on these components early on giving us a headstart in the final project.

Also, since we used encoders-photo interrupters in this design, we needed a lot of pins on the ESP32 board and ESP32 S2 was the obvious choice over ESP32 C3 and ESP32 WROOM. We used two 9V alkaline batteries in series (for more current at 9V voltage) to power this robot. Although we could implement a PD controller for yellow TT motors' velocity control, we could not get a straight line motion for over 1m distance. Also, some of the circuit was still made using breadboards and this resulted in noise in the circuit. We could successfully fulfil all tasks of Lab 4, but this robot was not capable of the final project's tasks.

5.2 Switch to 3-wheel robot

Having had a bad experience with the yellow TT motors in Lab 4, we ordered four new 12V 600rpm DC motors with hall encoders. However, due to the Black Friday sale, the purchasing department could only order two motors. This was a big bump to our timeline and the quick fix we planned on was to switch to a 3-wheeled robot, two standard wheels and 1/2 castor wheels type of chassis. Following was the design we planned to move with-

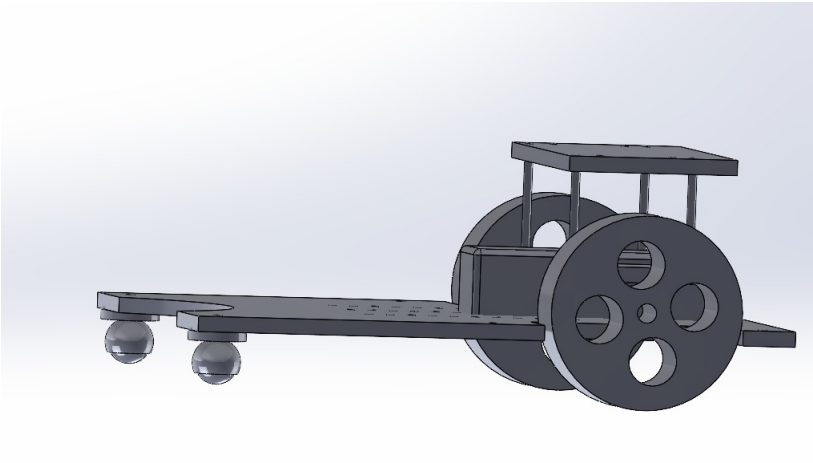


Figure 14: Three-wheeled mobile base design

5.3 New chassis, same old robot

Dissatisfied with the design change and realizing a high RPM motor wouldn't provide sufficient torque for the police car push task, we decided to go with lab 4's robot design, i.e., a four-wheeled base using the yellow TT motors. However, this time we made a wider, longer chassis for increased stability and used better quality 3D printed motor mounts. Another improvement we made was to use a 14.8V 2200 mah 4S LiPo battery to power the motors. The design of our final robot is as follows-

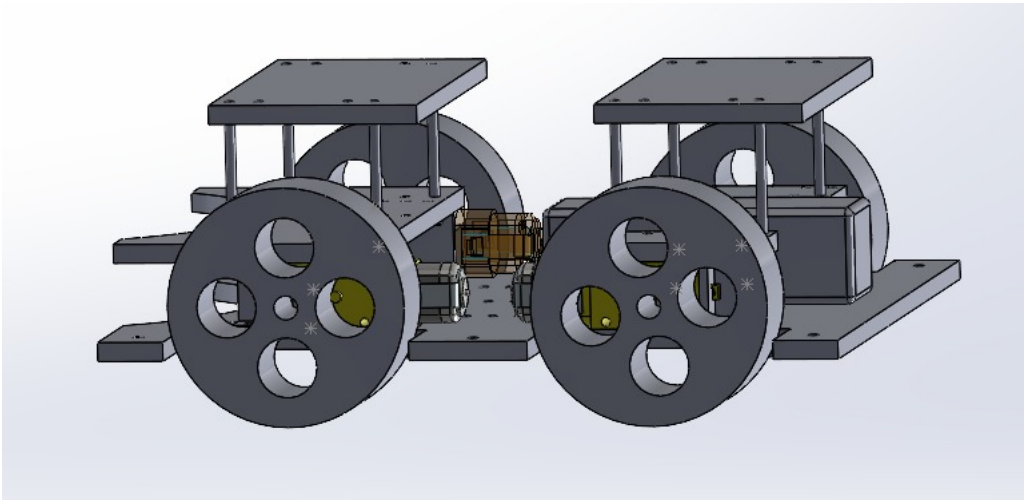


Figure 15: Final four-wheeled mobile base design

To our delight, the robot moved perfectly straight for longer distances and we got higher RPM and speed for just half PWM values. We accept that we underestimated the potential of the yellow TT motors and hence we did not need to use motor PID control. The semi-circular design at the front of the robot was to grab the beacon. All CAD models from lab 4 till the final project design can be found in section 8.3 (Appendix).

5.4 Parts damaged and replaced

The first big component we lost was the TB6612FNG motor driver. We suspect there was some back current that shorted the driver or maybe the metal surface touched with some other conducting surface on the robot. We would like to thank Team 8 who gave us a new motor driver and we could proceed without any issues.

The next big component that burnt was the ESP32 S2. Luckily TA Bruke was available late that night and he helped us out with a new one. Again, we suspect the board came in contact with some conducting surface or it was electrostatic residue from our fingers that shorted the board.

After these incidents, we insulated the bottom and sides of all boards using tape so that the solders at the bottom of the perfboards didn't come in contact with any conducting surface.

6 Grand Theft Autonomous Fall 2023

We are proud and honored to mention that we were the first team to check off the graded evaluation of the final project. Our team actively participated on both days of the competition. We stood at rank 2 at the end of day 1 (group stage). We had planned on implementing a gripper but since we had a strong police car push, we strategized pushing the police car first and then pushing the fake trophy to the other side. We crashed out of "semi-finals" in the bracket stage as we lost points since the team we seeded with wasn't using ESP-NOW which reduced our overall points.

The silver lining of the competition for us was that we could demonstrate a strong police car push and accurate beacon tracking. Given more time on our end, we would have integrated a gripper and done better beacon tracking and positioning. We thoroughly enjoyed both days of the competitions and thanks to Professor Yim and all TAs for setting up the amazing labs, lectures and GTA Fall 2023.

A compilation of our matches at GTA 2023- YouTube link

7 Retrospective

- What you feel was most important that you learned: Debugging. This course has taught us a lot of ways we can debug different problems, may it be software, hardware or others. Other key things learnt were obviously electronics and their usages, how to read datasheets and some good practices for operating electronic components/circuits.
- What was the best parts of the class: The final competition, GTA 2023 was the best part of the class. The feeling of able to test our robot to its fullest stress-free after weeks of hard-work was the best part of the class.
- What you had the most trouble with: Making circuits on breadboards as the connections are perfect always and the same circuit works differently during different times. Other main trouble aws using oscilloscopes and function generators as many a times the equipment at GM lab is faulty, and hence debugging becomes troublesome.
- What you wish was improved: TA office hours and their more one-on-one doubt solving can be improved. Equipment at GM lab can also be improved.

8 Appendix

8.1 Bill of materials

Item name	Quantity	Item Procured from	Total price
TB6612FNG Motor driver	2	Amazon	$9.99
LM2596 Buck converter	2	Amazon	$5.49
HC-SR04 Ultrasonic sensors	5	Amazon	$8.99
Mecanum wheels (80mm diameter)	4	Amazon	$ 24.30
5000mAh Mady Power Bank	1	GM lab	$ 7.00
ESP32 S2	2	GM lab	$ 16.00
ATMega32u4	2	GM lab	-
PD-70 photodiodes	2	GM lab	-
VL53L0X Time of Flight sensors	3	Amazon	$ 11.69
VL53L0X Time of Flight sensors	2	Amazon	$ 9.99
2200mah 14.8V 4S LiPo battery	1	Amazon	$ 23.59
Caster wheels**	2	GM lab	$ 4.00
SparkFun RedBot Sensor - Line Follower**	1	Amazon	$ 6.94
Base Plate (acrylic sheet)	1	Rapid Prototyping Lab	-
Motor mounts	4	Education Commons	-
TT yellow stock motors	4	GM lab	-
perfboards	6	GM lab	-
TLV272 opamp	4	GM lab	-
12V 600rpm DC motors with hall encoders*	2	Amazon	$ 25.58
25mm DC gearbox motor brackets*	6	Amazon	$ 11.89
4mm motor coupler*	1	Amazon	$ 6.94

*highlighted parts were returned to the GM lab (handed over to Professor Yim) to settle our $150 allocation
**parts were ordered but not used finally due to design modifications

8.2 Circuit schematics

The whole robot architecture's circuit schematic, with detailed pinouts of ESP32 S2 and all peripheral circuits- circuit link

8.2.1 IR detection circuit

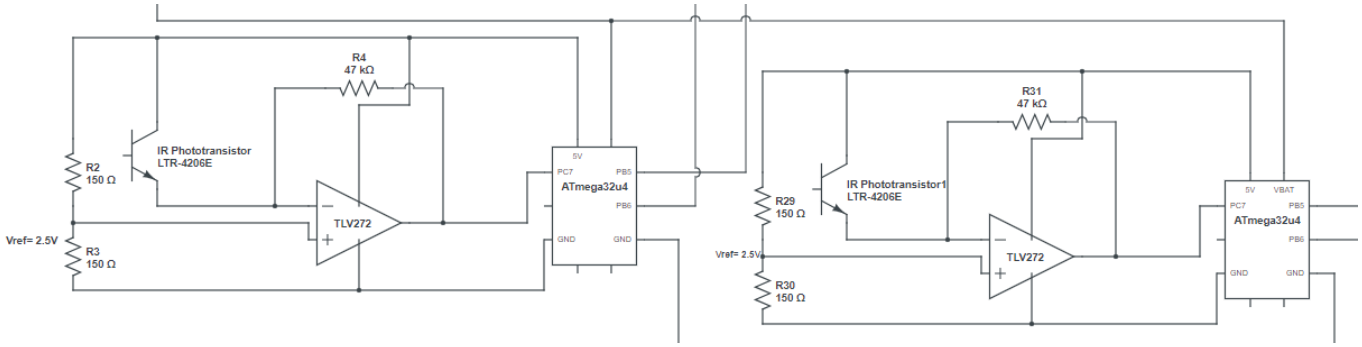


Figure 16: Beacon tracking circuit diagram (two circuits used)

8.2.2 HTC Vive circuit

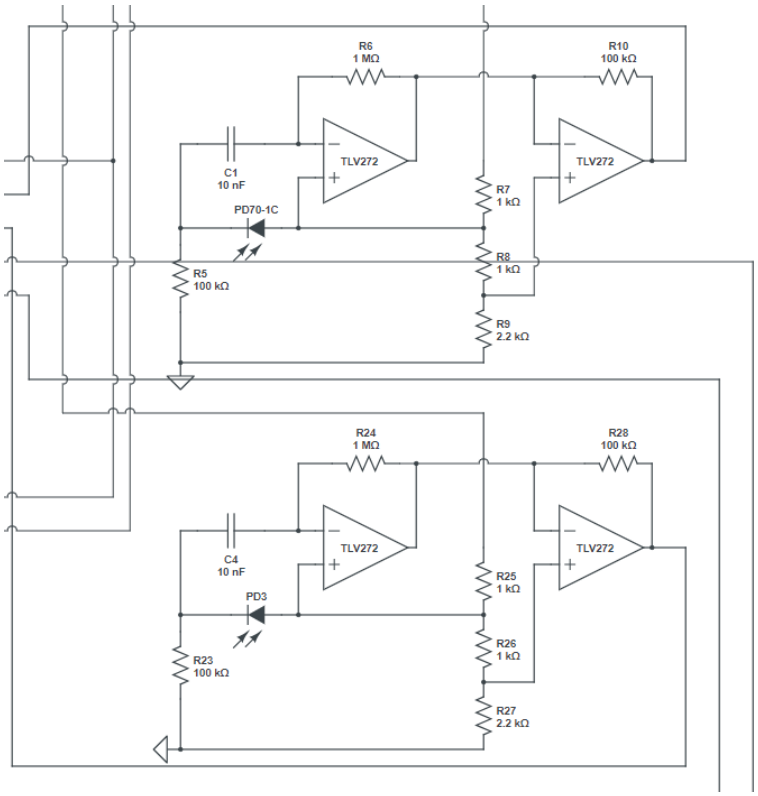


Figure 17: HTC Vive detection and tracking circuit diagram (two circuits used)

8.2.3 Motor driver circuit

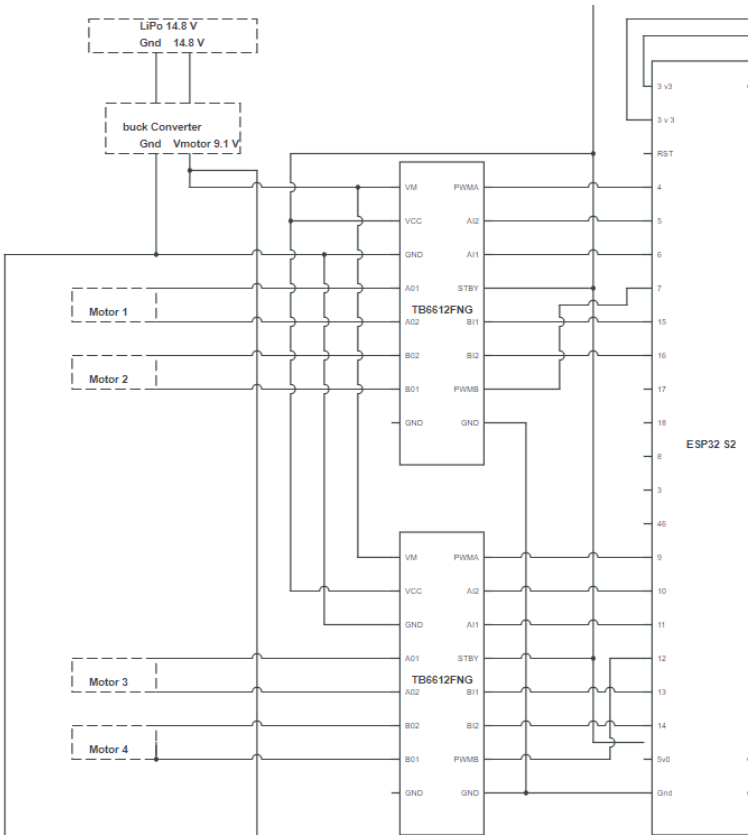


Figure 18: TB6612FNG motor driver circuit diagram (two circuits used)

8.3 CAD models

8.3.1 Lab 4 robot design

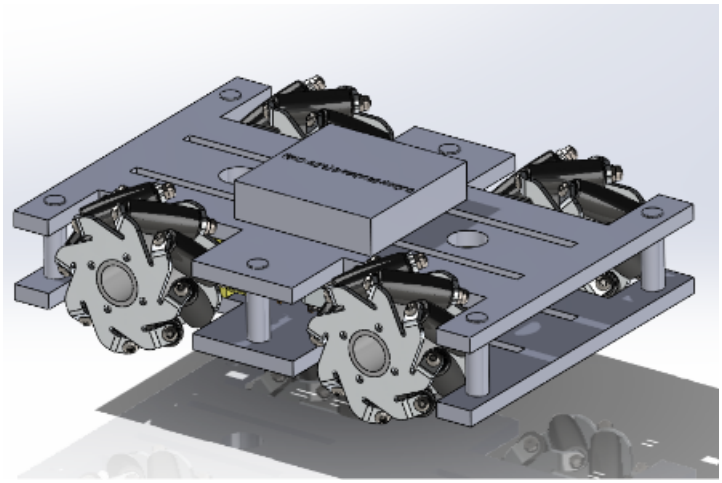


Figure 19: Lab 4 robot design

8.3.2 Chassis design

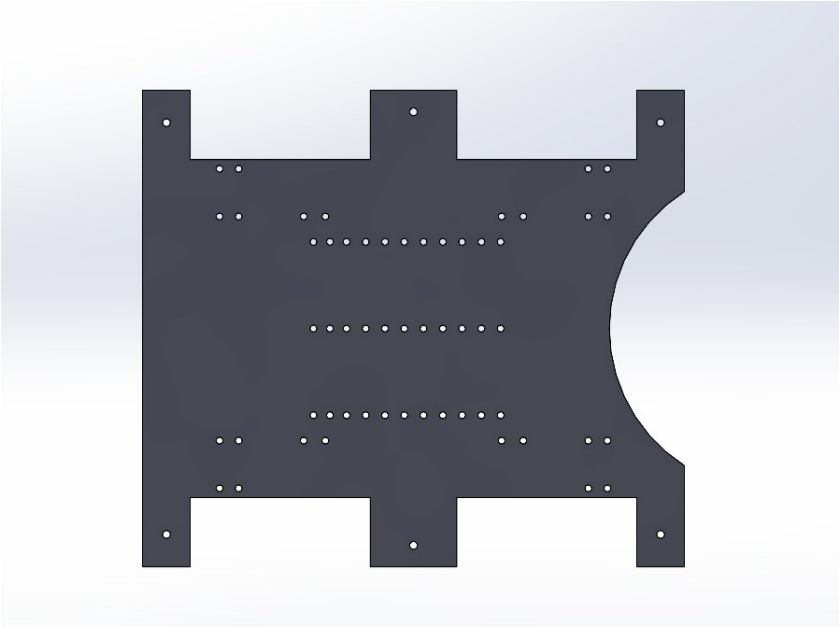


Figure 20: Robot's base/chassis cad model with holes defined for accurate mounting

8.3.3 Chassis' dimensions

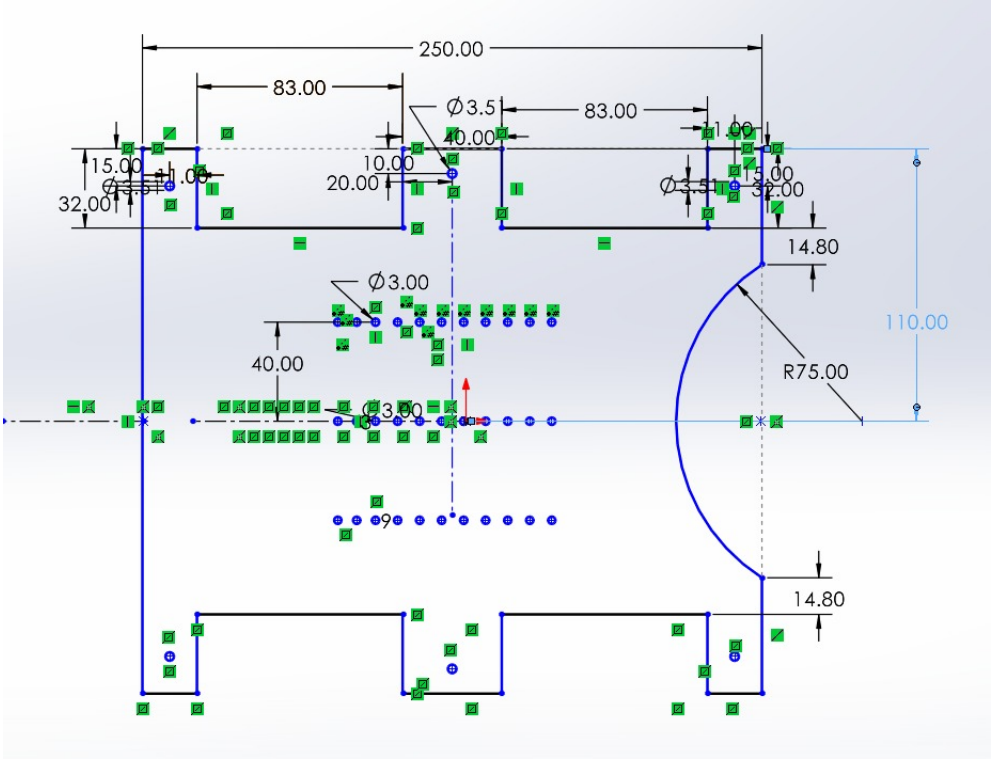


Figure 21: Chassis dimensions

8.3.4 Motor mount

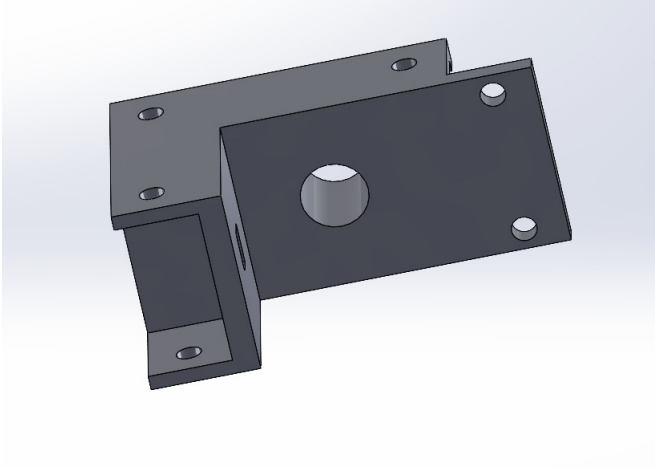


Figure 22: Chassis dimensions

8.3.5 Final project robot

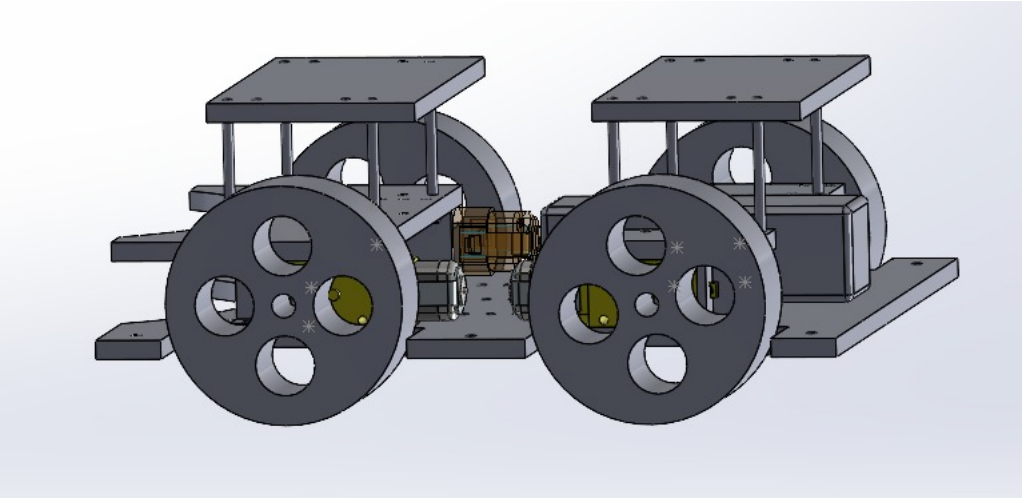


Figure 23: Chassis dimensions

8.4 Videos

8.4.1 Wall following

YouTube link- Wall following task

8.4.2 Beacon tracking

YouTube link- IR LED Beacon tracking task

8.4.3 Police car push

YouTube link- Police car push task

8.5 Datasheets

- TB6612FNG motor driver datasheet: https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf
- Ultrasonic sensor HC-SR04 datasheet: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- LM2596 buck converter datasheet- https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1703116820318

8.6 Software approach algorithms

Algorithm 1 Wall Following Robot

```
Set up ultrasonic sensor pins (trigPin1, echoPin1, trigPin2, echoPin2)
Set constants for wall following (wallDistance, wallFollowSpeed, wallKP, wallKD)
Initialize variables (prevWallError, integralWallError)
function FOLLOWWALL(minDistance, maxDistance, sensorDistance)
    if sensorDistance is within minDistance and maxDistance then
        Calculate mecanum wheel speeds for forward motion
        Move the robot forward
    else
        Calculate wall error based on desired and measured distances
        Apply PID control to compute wall-following effort
        Calculate mecanum wheel speeds based on wall-following effort and forward speed
        Move the robot with the calculated wheel speeds
        Update previous wall error for the next iteration
    end if
end function
function WALLFOLLOW
    Measure distance from the first ultrasonic sensor (trigPin1, echoPin1)
    Measure distance from the second ultrasonic sensor (trigPin2, echoPin2)
    Print distances to the serial monitor
    Call followWall function with appropriate parameters based on the second sensor reading
    if distance from the first sensor is below a threshold then
        Rotate the robot counterclockwise for a specified duration
    end if
    Add delay for stabilization (adjust as needed)
end function
```

Algorithm 2 Beacon Tracking

```
Define beacon sensor pins (Bleft_H, Bleft_L, Bright_H, Bright_L)
Initialize variables (highDetected = false)
function BEACONTRACKING
    Print "Beacon" to the serial monitor
    while not highDetected do
        Move the robot right (MEC_RIGHT) at full speed
        if Bleft_L is HIGH and Bright_L is HIGH then
            Set highDetected to true
            Break the loop
        end if
        Delay for 2500 milliseconds
        Move the robot forward (MEC_FORWARD) at full speed
        if Bleft_H is HIGH and Bright_H is HIGH then
            Set highDetected to true
            Break the loop
        end if
        Delay for 1000 milliseconds
        Move the robot left (MEC_LEFT) at full speed
        if Bleft_H is HIGH and Bright_H is HIGH then
            Set highDetected to true
            Break the loop
        end if
        Delay for 2500 milliseconds
        Move the robot forward (MEC_FORWARD) at full speed
        if Bleft_H is HIGH and Bright_H is HIGH then
            Set highDetected to true
            Break the loop
        end if
        Delay for 1000 milliseconds
    end while
    if Bleft_H is HIGH and Bright_H is HIGH then
        Set highDetected to true
        Move the robot forward at full speed (MEC_FORWARD)
    else
        if highDetected then
            Stop the motors and rotate counterclockwise (MEC_ROTATE_COUNTERCLOCKWISE) at half
speed
        end if
    end if
    Delay for 100 milliseconds to avoid rapid checking
end function
```

Algorithm 3 Vive Tracker Control

Include necessary libraries and define hardware connections
Set up server and create Vive tracker instances
Define motor connections, Mecanum wheel movement modes, and PID control constants
Initialize target and current positions, and previous errors
function CALCULATEPDEFFORT(target, current, prevError, kp, kd)
    Calculate and return control effort based on proportional and derivative terms
end function
function CALCULATEPDEFFORT_ROTATION(target, current, prevError, kp, kd, fr_pwm, fl_pwm, rr_pwm, rl_pwm)
    Calculate and return rotation control effort with motor direction adjustment
end function
function CALCULATEMECANUMWHEELSPEEDS(xEffort, yEffort, fl_pwm, fr_pwm, rl_pwm, rr_pwm)
    Calculate and normalize individual wheel speeds based on control efforts
end function
function MOVETOTARGETCOORDINATES(targetX, targetY, currentX, currentY)
    Calculate PD efforts for X and Y positions
    Calculate Mecanum wheel speeds and move the robot
end function
function MOVEMOTORS_DIRCONTROL(speedRF, speedLF, speedRR, speedLR, dircontrol)
    Move all 4 motors using specified speeds and direction control byte
end function
function MOVEMOTORS(speedRF, speedLF, speedRR, speedLR, dircontrol)
    Move all 4 motors using specified speeds and direction control byte
end function
function STOPMOTORS
    Stop all motors and motor controllers
end function
Set up Vive trackers
while true **do**
    Acquire position information from Vive trackers
    if tracking status is OK for both trackers **then**
        Calculate target and current positions, and move robot
    else
        Handle tracking synchronization or signal loss for each tracker
    end if
    Adjust delay as needed
end while

Algorithm 4 Police Car Movement

```
Define constant values and pins
Initialize variables: flag, error, x1, y1, x2, y2, x_center, y_center, x_target, y_target, car_y
while true do
    if WiFi is connected then
        Set RGB LED color to full white
        Send UDP packet with the current coordinates (x_center, y_center)
    end if
    if Both Vive sensors are receiving signals then
        Read coordinates from both Vive sensors (x1, y1, x2, y2)
        Calculate the center coordinates (x_center, y_center)
        Set RGB LED color based on Vive sensor data
    else
        Set Vive sensor coordinates and center coordinates to zero
        Check synchronization status for Vive sensors and set RGB LED color accordingly
    end if
    if Vive signals are weak or missing then
        Handle synchronization and signal status for Vive sensors
    end if
    if car_y is not initialized then
        Read car_y from UDP server
        Set error to a large value (1000)
    end if
    if flag is 0 then
        Set target coordinates (x_target, y_target) to the center with a slight offset
        Calculate error using the PD controller and move the robot to the target
        Check if the error is small, and if true, set flag to 1 and move forward
    end if
    if error is small and flag is 1 then
        Move the robot forward
    end if
    Add a delay to control loop execution (adjust the delay as needed)
end while
```

References

[1] Mecanum Wheel Robot Car with ESP-NOW Remote Control *https://dronebotworkshop.com/mecanum/*, Unknown, Last accessed 17 December, 2023