Track Bot

Senior Project by Angelika Canete and Miguel Saenz

California Polytechnic State University, San Luis Obispo

Computer Science & Software Engineering Department

Advised by Dr. John Seng

Table of Contents

Introduction

The purpose of this project was to create a robot that safely follows a person to keep track of and set their running pace. This project is designed for people who want to get into running and want to keep track of their desired pace. The idea is for the robot to run alongside a human to track their progress and deliver results at the end of the run. Results would include analytical data of their past runs so that they may see if they are progressing or not. This first portion of this project uses a robot to successfully track and follow a human according to their body width, drives towards them and safely stops when needed.

Budget and Bill of Materials

Part	Part Selected	Quantity	Link	Price (per item USD)	Price (no ship/tax USD)
Processor	Pi 5	1	Link	45	45
Webcam	Playstation Camera	2	Link	25	50
Battery	LiPo 5000 mah (2 pack)	1	Link	55	55
Motor	10:1 Metal Gearmotor with 64 CPR Encoder	2	Link	70	140
Motor Controller	HiLetGo 43 A	2	Link	11	22
Screws	Negligible				
Lidar	RPLidar A1M8	1	Link	100	100
Wheels	Pololu 90mm (2 pack)	1	Link	9.5	9.5
Buck Converter	DWEII 12V to 5V Converter	1	Link	15	15

Terminal Block	Joinfworld 35A Terminal Block	1	Link	9	9
Portable Charger	Anker GANPRIME Power Bank 2-in-1	1	Link	100	100
Filament	Polymaker Orange PLA	1	Link	25	25

We started by deciding on which compute unit we should use in order to control our system, and landed on the Raspberry Pi being the most capable and most flexible option early into the design process. From there we determined which motors made the most sense, selected appropriate motor controllers and wheels for said motors, and determined which camera we thought would work best. Later into the design process we figured that a Lidar would be a useful utility component and so added that to our list as well. This brought the cost of the overall robot including tax and shipping where applicable to a grand total of $624. Now, the real cost was fairly reduced in a number of ways, in that we were lucky enough to already have access to the Playstation cameras we intended to use, the motors we selected, and the portable charger that we ended up using. That made the real cost for our iteration of this project come out to instead approximately $310. This price was much more manageable for the scale and scope of our project, and so we pushed forward.

Mechanical

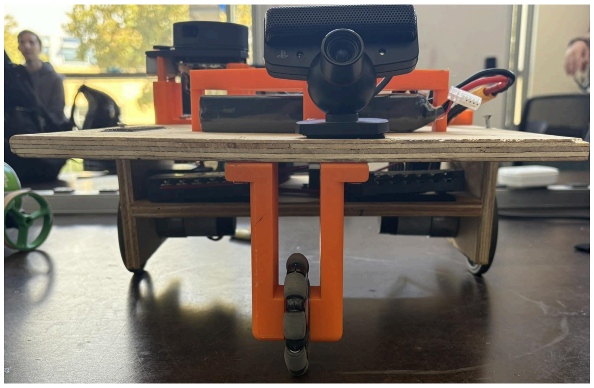


Fig 1: Front view

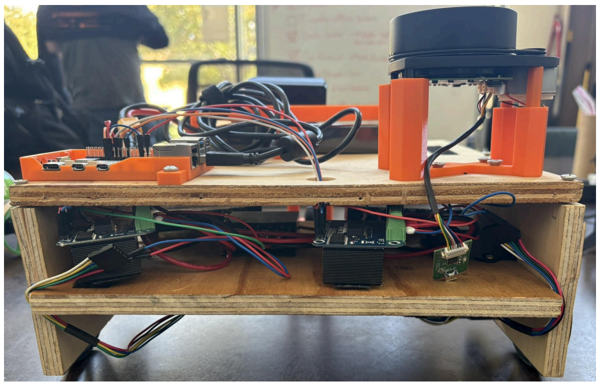


Fig 2: Back view

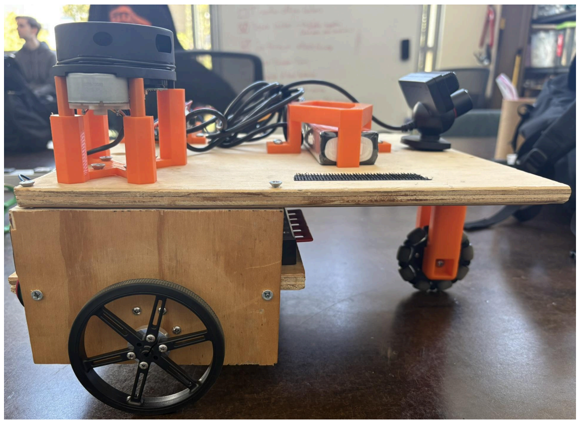


Fig 3: Right side view

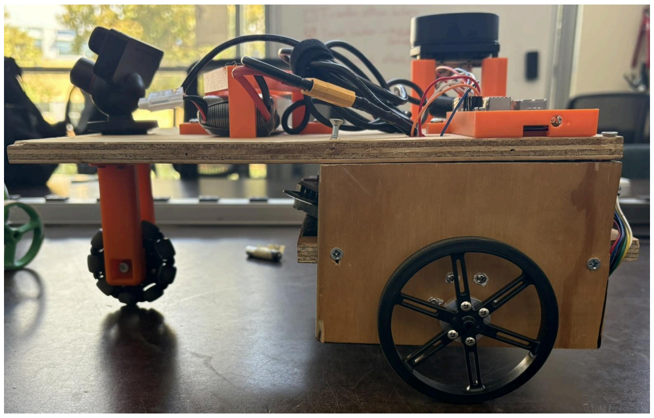


Fig 4: Left Side view

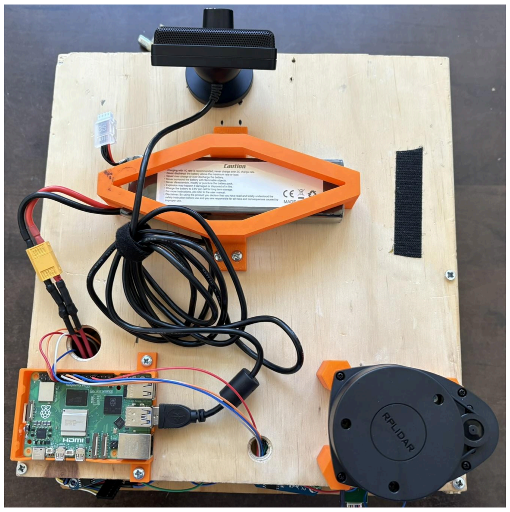


Fig 5: Top View

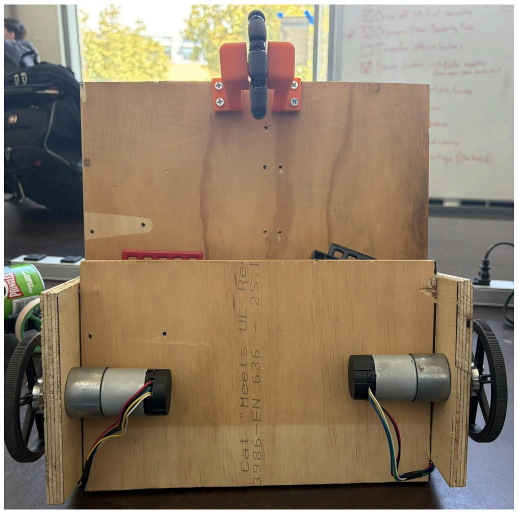


Fig 6: Bottom View

The above Figures (#s 1-6) show all the angles of the finished robot, with the minor exception of the portable power bank that we used to power our Raspberry Pi as we needed to return it to the owner before the conclusion of the quarter. The vast majority of the mechanical and structural components were made out of .25" plywood and all other structural components were supplemented with PLA 3D printed components. Below are renders of each of these (custom designed) 3D printed components.

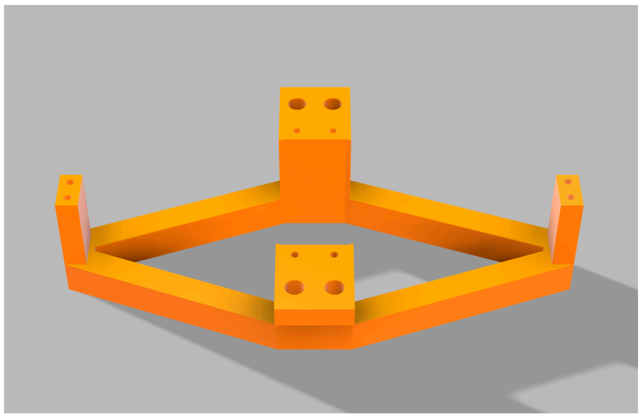


Fig 7: Battery Holder

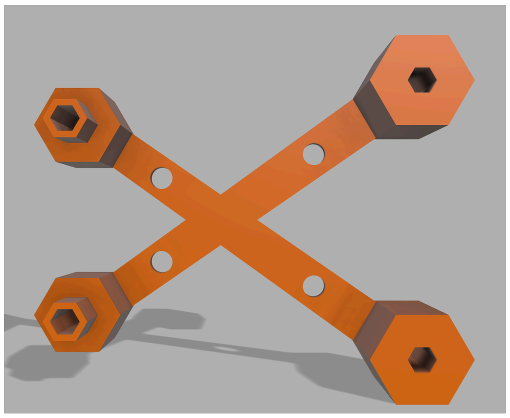


Fig 8: Lidar Holder (Friction Mount)

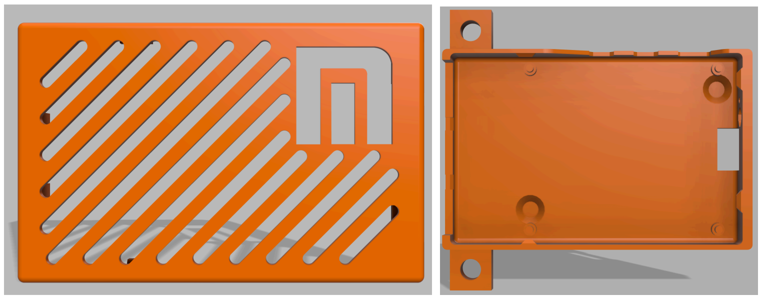


Fig 9-10: Top and Bottom of Raspbery Pi Case

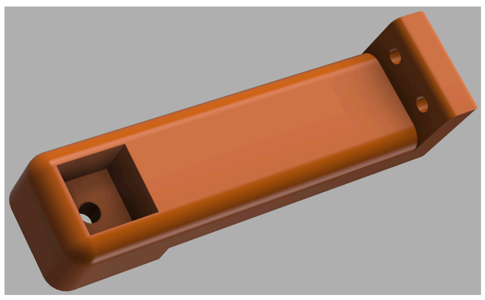


Fig 11: One (1) Omni Wheel Axle

Each of these 3D printed components was mounted to arbitrary positions on the Plywood plate, as a general layout was agreed upon, but drilling lead holes and using wood screws made it so that we could perform the layout and assembly all at once in person. The battery holder, Lidar holder, and Raspberry Pi case were all mounted on the top of the top-most wooden plate with #10-32 0.25” wood screws. The battery holder and Pi case are largely unremarkable and are simply purpose built receptacles for their respective components. The Lidar holder however is unique in its design as we needed to make it sturdy enough to keep the Lidar in a consistent fixed position, but also allow for the quick removal of the Lidar so that development could continue in parallel on drive and Lidar code when developers were in separate locales. To achieve this, the Lidar holder has specific offsets and orientation for the standoffs that come with our Lidar model, as listed in the Bill of Materials, that ensure a consistent position. To allow for easy removal, the Lidar holder is designed to just be a friction fit with the Lidar rather than any sort of explicit mechanical coupling or fastening. Similarly, two Omni Wheel Axles were affixed to the bottom of the topmost plate using those same wood screws and screwed together with a standard #10-32 .5” bolt that was hand tightened with a #10-32 nyloc nut to allow the wheel to free spin while maintaining structural integrity across the two axles.

Hardware

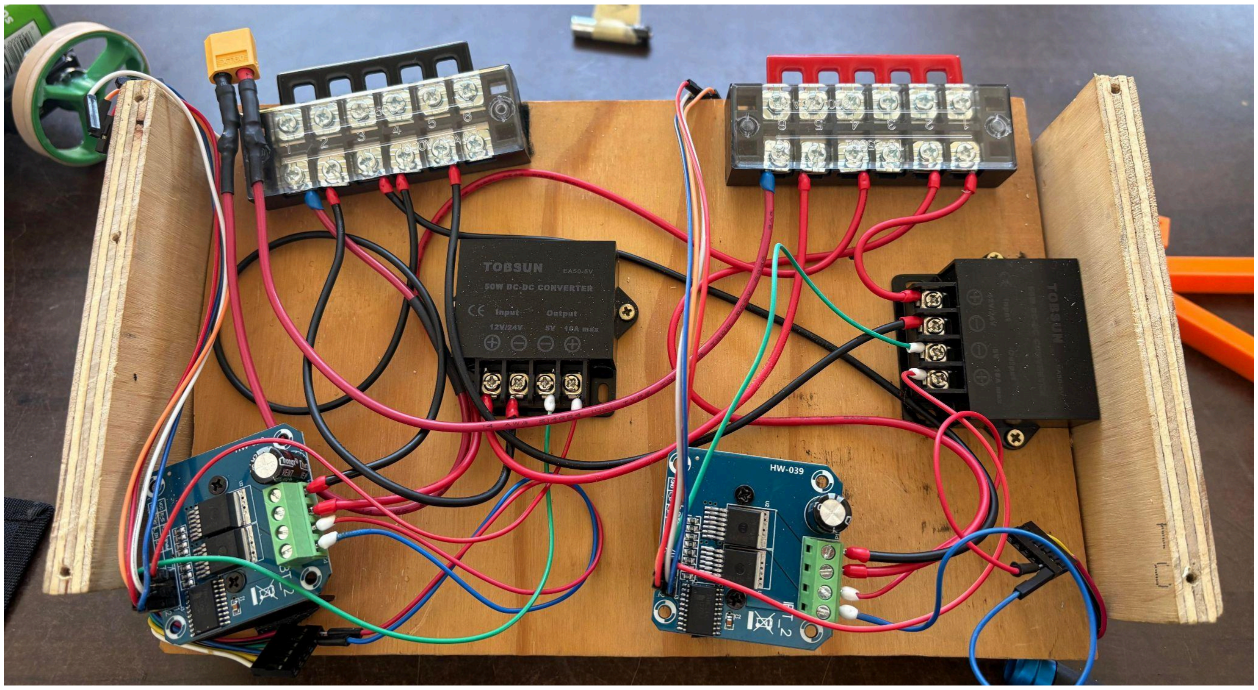
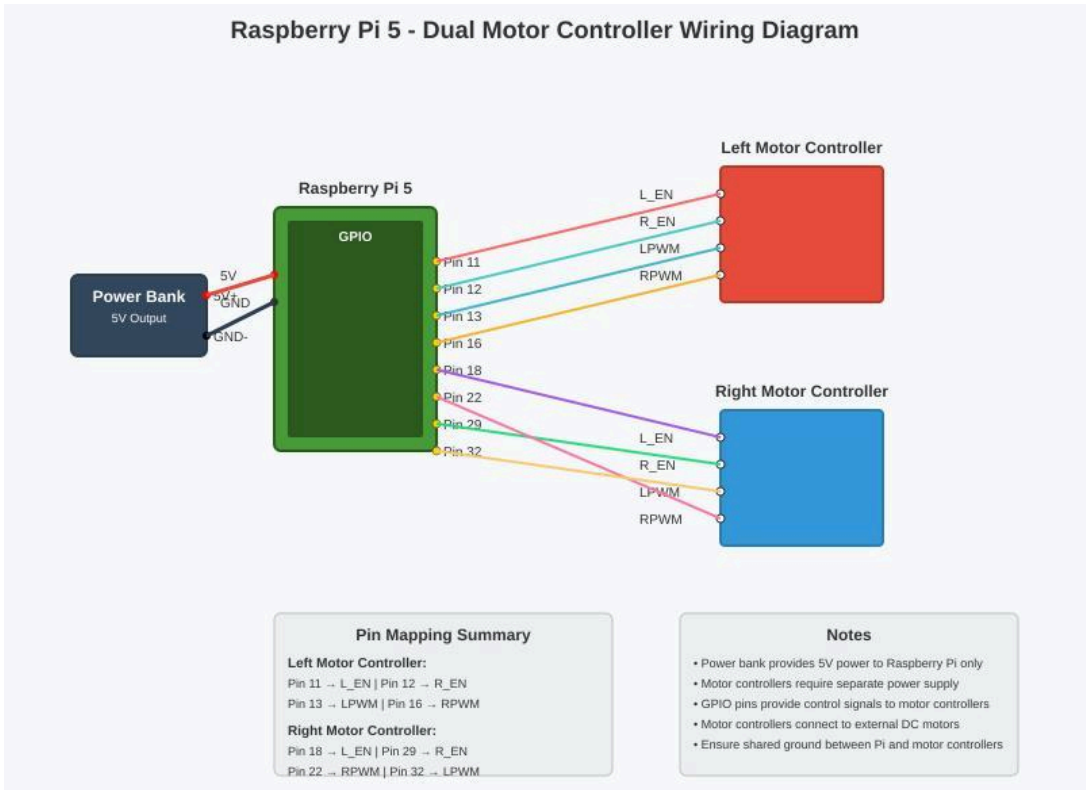


Figure 12: Electronic wiring layout (Second layer wooden plank)

The electronic components consist of two distinct but interconnected systems, the Raspberry Pi electronics system, and the 12 volt motor driving system. The Raspberry Pi system is powered by a 5 volt battery bank, while the 12 volt driving system uses the LiPo batteries listed in the bill of materials above. The Pi system connects to the motor controllers via the GPIO interface for control as listed in Figure 13. The rationale for powering this system separately, is that the Raspberry Pi is a very valuable and similarly sensitive part of this build, so we wanted to isolate its power supply and wiring from the rest of the system that was more amateurly wired as much as possible. This turned out to be a good idea, as we did ultimately accidentally short a wire in the 12 volt system line that could have been a major issue if it was a part of the Pi wiring system.



Pin Mapping Summary

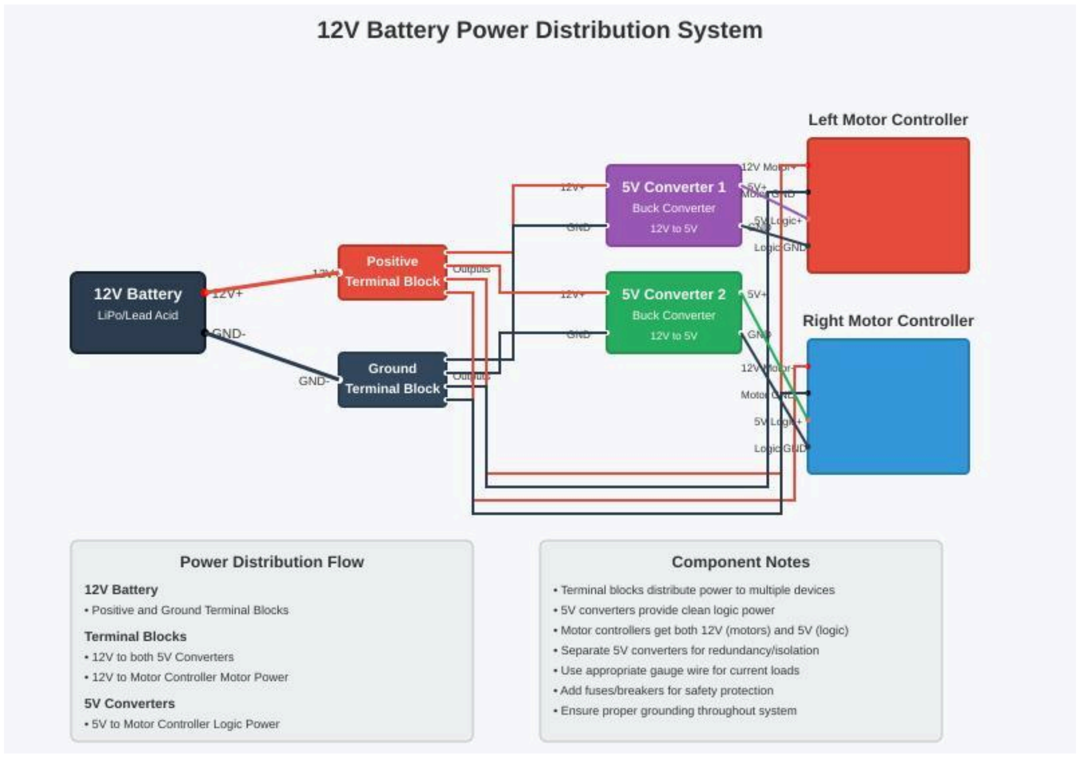
Left Motor Controller:
Pin 11 → L_EN | Pin 12 → R_EN
Pin 13 → LPWM | Pin 16 → RPWM

Right Motor Controller:
Pin 18 → L_EN | Pin 29 → R_EN
Pin 22 → RPWM | Pin 32 → LPWM

Notes

- Power bank provides 5V power to Raspberry Pi only
- Motor controllers require separate power supply
- GPIO pins provide control signals to motor controllers
- Motor controllers connect to external DC motors
- Ensure shared ground between Pi and motor controllers

Figure 13: Raspberry Pi System Wiring



Power Distribution Flow

12V Battery
- Positive and Ground Terminal Blocks

Terminal Blocks
- 12V to both 5V Converters
- 12V to Motor Controller Motor Power

5V Converters
- 5V to Motor Controller Logic Power

Component Notes

- Terminal blocks distribute power to multiple devices
- 5V converters provide clean logic power
- Motor controllers get both 12V (motors) and 5V (logic)
- Separate 5V converters for redundancy/isolation
- Use appropriate gauge wire for current loads
- Add fuses/breakers for safety protection
- Ensure proper grounding throughout system

Figure 14: 12V Battery Power Distribution

The 12 volt system connects to a terminal block to distribute 12 volt power to all of the components that need it, as well as to be turned into 5v in order to power both motor controllers respectively. The 12 volt power is plugged into the motor controllers in order to distribute them to each respective motor for speed control. The terminal blocks are the key in this system as they allow for all of the components to run off a single battery rather than requiring a distinct battery for each circuit.

Software

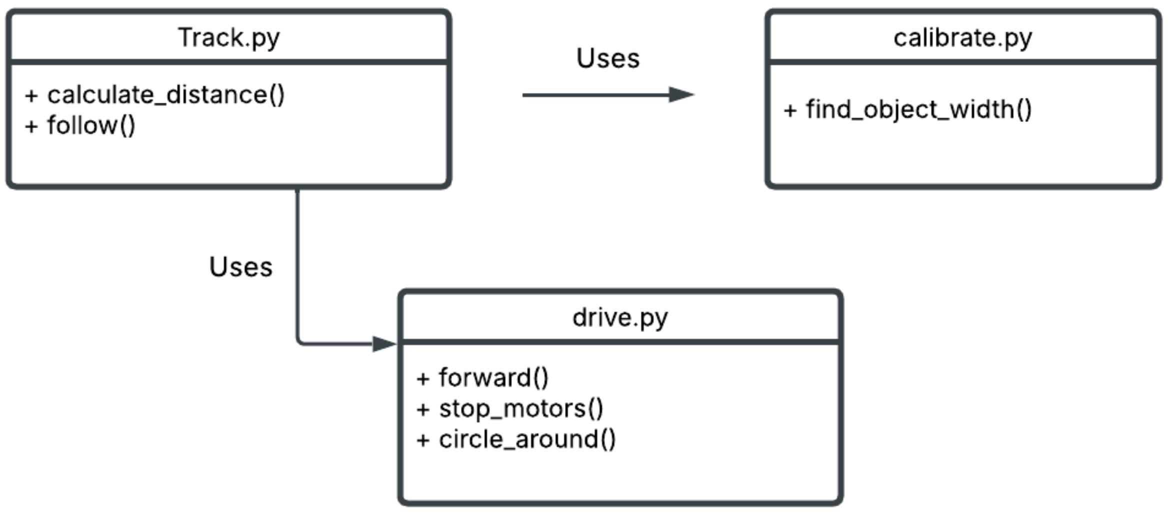


Figure 15. Overall architecture

The project consisted primarily of three programs: calibrate.py, drive.py, and track.py. The purpose of the calibrate program is to find the focal length of the camera in pixel units so that distance estimation may be used later in track.py. It works by pointing a webcam at an object whose distance is known, then finds the focal length using the equation:

Focal length = (width_in_pixels * known_distance) / known_width

After a few calibration runs, we took the average of the focal lengths and found that the ideal focal length is 750. From there, this value is set as a constant in track.py.

In track.py, the program utilizes a TensorFlow model for people detection. In particular, we opted to use Tensorflow's SSD Mobilenet V2 Object detection model that was trained on a 2017 dataset with 330K images. The dataset includes 80 object categories and 91 miscellaneous categories, we utilize the first class to detect humans. To control the robot's movement, our follow function issues drive commands to the robot based on the position of the detected person in the camera frame and their estimated distance from the robot. This process begins by first identifying a person and drawing a bounding box around them. The center of this bounding box represents the person's position in the frame. If a person is not within 50 pixel units from the frame's horizontal center, in both the x and y direction, then the function adjusts the robot's motor speed to turn toward the person. This adjustment uses a proportional controller (P-controller), where the error is the horizontal distance between the person's x-coordinate and the center x-coordinate of the frame. The turn factor is computed as:

Turn_factor = Kp * error

This code continuously captures video frames from a webcam and uses a TensorFlow object detection model (SSD MobileNet) to identify people in the frame. Each frame is resized and preprocessed before being passed to the model, which returns bounding boxes, classes, and confidence scores for detected objects. If a person is detected with a confidence level greater than 0.7, the code calculates the person’s position in the frame and estimates their distance using the size of the bounding box. Based on this information, the robot responds accordingly: it moves forward if the person is centered, turns if they are off-center, and stops if the person is too close. If no person is detected, the robot starts circling in place to search for someone. This enables the robot to actively follow a person while avoiding getting too close, and to look around when no target is visible. In drive.py, the program lays out basic robot commands of: forward, stopping, and circling around. The program uses the gpiozero library, which is specific for Raspberry Pi 5. Our plan for the robot’s action is to: circle around until we detect a human, follow said person, then stop when less than 150 inches from the person. This is depicted in the following diagram:

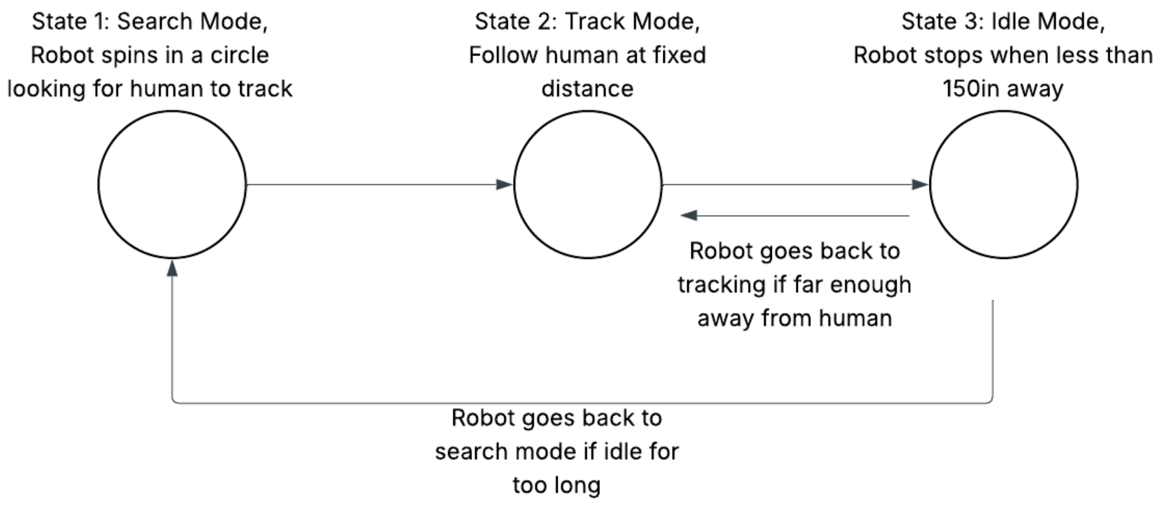

State 1: Search Mode, Robot spins in a circle looking for human to track
State 2: Track Mode, Follow human at fixed distance
State 3: Idle Mode, Robot stops when less than 150in away
Robot goes back to tracking if far enough away from human
Robot goes back to search mode if idle for too long

Figure 16. Diagram of Robot’s Movements

Lessons Learned

The first lesson we learned was the importance of developing an effective design for the robot. Our initial concept featured two stacked base plates, with the camera and LiDAR mounted on the top plate for optimal visibility, and the remaining hardware positioned below. However, after several discussions, we opted for a more streamlined approach. The final design placed the camera, LiDAR, battery, Raspberry Pi, and its power bank all on the top shelf, while a smaller second shelf—about a third the size of the top one—was installed beneath it to hold the motor controllers. We also added a hole in the top shelf to allow jumper wires to connect easily to the Raspberry Pi to improve accessibility. We switched to this design for ease of assembly, maintenance, and wiring access. By placing all the main components on the top shelf, the layout became more compact and accessible. This made it easier to troubleshoot, manage cables, and make adjustments without needing to disassemble layers of hardware. This design was especially helpful because, early in the process, we had to frequently rewire the Raspberry Pi to align with changes in the code. Having all the critical components—including the Raspberry Pi—on the top shelf made it much easier to access and adjust connections without disrupting the rest of the setup. In contrast, our original design would have required repeatedly removing the motor controller from the lower shelf, which was time-consuming and risked damaging components. The simplified layout reduced these issues and allowed us to iterate and troubleshoot more efficiently.

Furthermore, another learning curve was figuring out how to detect people appropriately. At first, we utilized OpenCV's library for detecting people; however, this was not efficient enough and would often detect random objects. It was also quite unstable and would not focus on one object or person. From there, we decided to switch over to Tensorflow's human detection algorithms. We were hesitant at first to use an AI model since it would be running on a Raspberry Pi. Because of this, we opted for Tensorflow's Lite version because it is optimized for speed and has lower latency for real-time applications. At first the model was also slow but highly accurate. To speed up the frame rate, we got rid of the graphical interface that showed us the camera feed, which worked well in the end.

From a hardware and mechanical perspective most of the challenges revolved around fitment and ensuring that all of the convoluted electrical wiring was consistent even when components were disconnected and reconnected. As previously mentioned, another challenge that arose was when I (Miguel) accidentally created a short in the 12 volt power system between the buck converter and motor controller that fried a wire. We were lucky enough to only have the wire be damaged, with the wire essentially acting as a, let's say temporary resistor.

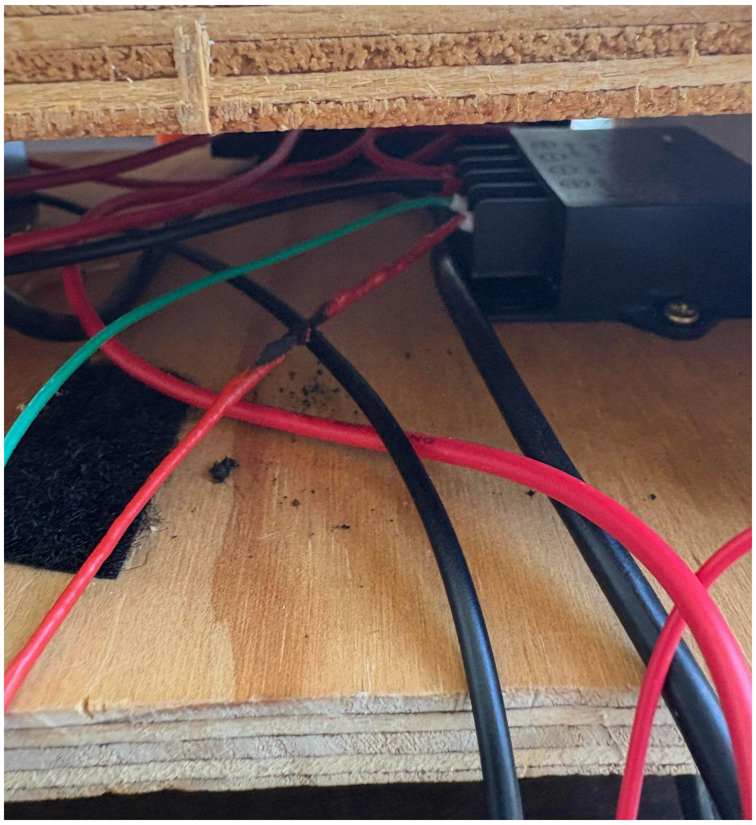


Figure 17: Fried Wire

From a consistency perspective with wiring, we were attempting to troubleshoot whether one or both of our motor controllers had overheated and died, and in the rush of trying to fix things we forgot to write down our motor pinout. This led to us determining that our motor controllers were functional, attaching them to the Pi in the configuration that we thought was correct, and running code that ultimately did not work. From there, we ensured that our pinout was consistent and documented, and no issue arose with that component again.

Conclusion

In the end, we successfully developed a fully functional robot capable of safely tracking and following a person using real-time object detection. By leveraging TensorFlow's library alongside a PID drive controller, we implemented autonomous driving logic that enables the robot to adjust its movement dynamically based on a person's distance and position within the camera frame. This project not only required us to rely on our previous coding knowledge, but also required us to venture into learning robotics hardware, motor control, and the basics of circuits. Throughout the development process, we gained valuable hands-on experience integrating software with physical components and learning to debug from both hardware and software perspectives. This dual-layered problem-solving approach helped us better understand how code interacts with real-world systems, deepening our appreciation for the complexities of robotics and embedded systems development.

Our project represents the first iteration of a fully operational pacer bot. Future improvements could include refining the tracking system for greater accuracy, developing a companion app to provide real-time performance analytics, and enhancing the robot's mobility to better navigate uneven or outdoor terrain.