Final Project

Run of the Roadsters

Team Members:
Prince Lucky F. Santos,
Avinh Anthony Huynh,
Hilary Lui,
Jacob Vazquez

Github Username:
Jacob9610

Bierman Poem

Bierman Bierman

Bierman Bierman

Bierman Bierman

Bierman Bierman

Multithreading, Pipelining,

and Filesystems too

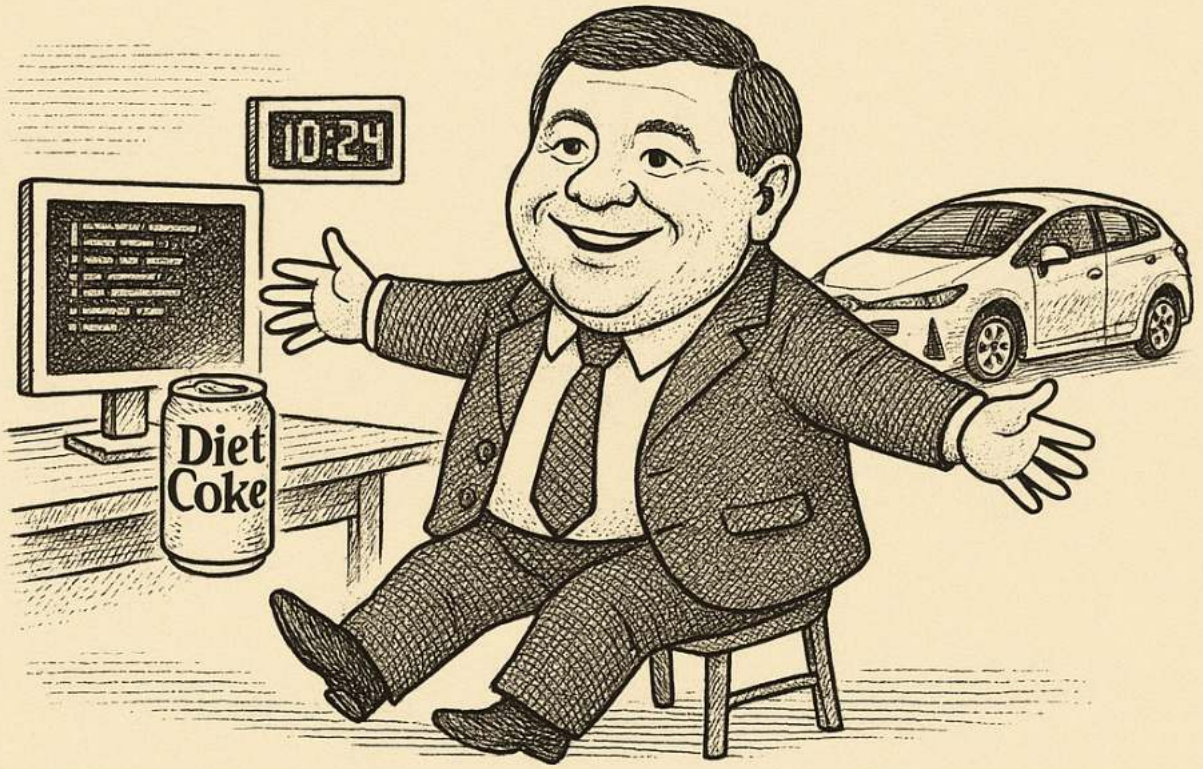
If you don't P-L-A-N

he will get you

Bierman Bierman

Bierman Bierman

A BOOK OF BIERMAN



There once was a professor Robert Bierman,
Whose code lectures all had great cheer in.
With Diet Coke near,
And his Prius Prime in gear,
He taught bits and bytes without fearin'.

Description of the Task:

The goal of this project was to design and program an autonomous robot using a Raspberry Pi to navigate a black-tape-lined course, avoid physical obstacles, and stop at a red tape marker. The robot integrates multiple sensors and embedded control logic to operate reliably in real-time.

At startup, the robot remains stationary until its onboard push button is pressed. Once activated, it uses its IR sensors to detect and follow the black tape. The robot is capable of handling sharp turns (including 90-degree corners) while staying aligned with the path and avoiding deviation or reversal.

The HC-SR04 ultrasonic sensor monitors the path ahead for obstacles. When an obstacle is detected within a predefined distance, the robot initiates an avoidance routine and then re-aligns itself with the black tape to continue along the course.

The RGB color sensor plays two key roles: detecting a red tape marker at the end of the course (which triggers a complete stop and graceful program exit), and—as a surprise requirement during the final demonstration—detecting a green tape marker, upon which the robot must perform a full 360-degree turn before continuing on the path.

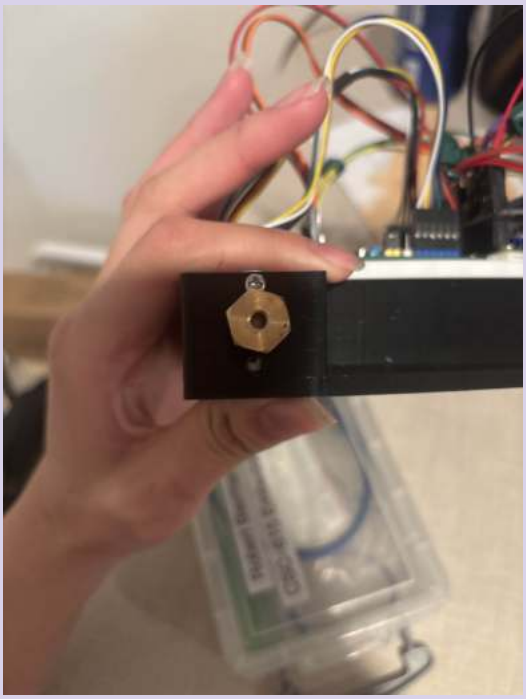
This last-minute challenge tested our ability to modify the control logic in real time.

Building the Robot:

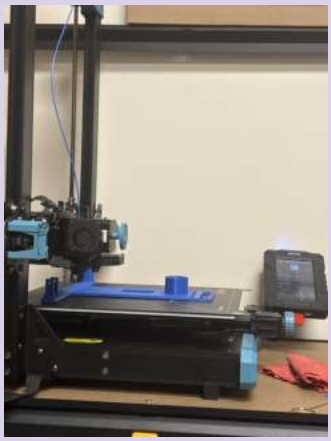
For building the robot, Jacob worked on it for like 2 days 5 days before it was due, and then I hopped on the project and we spend the next 3 days not sleeping to grind the whole project out. In some of the pictures you can see me sleeping on the floor with tape after we were testing it in our house. Hilary bought me a 4 pack of coffee and earbuds and I consumed like 1000mg of caffeine of 6 coffees in one day. We spent a lot of time connecting sensors, reprintgin the car over and over, rewriting the i2c driver and then realizing we cant have the rgb sensor on the same bus as the motor driver, so we had to last second pivot to the pi4.

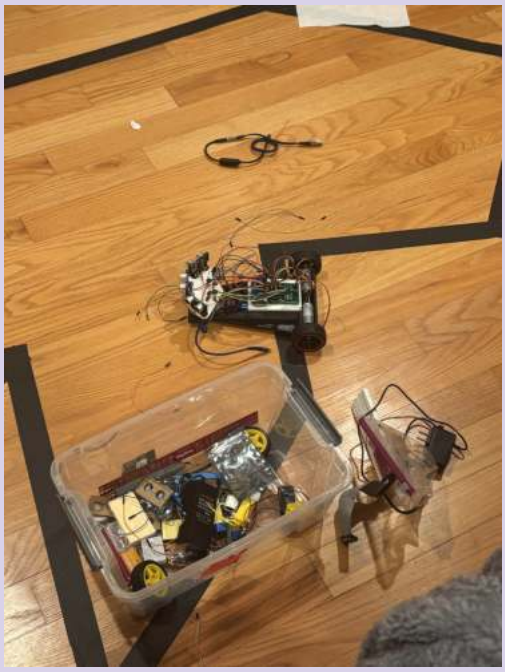
A Whole Lotta photos:


MODE
ON



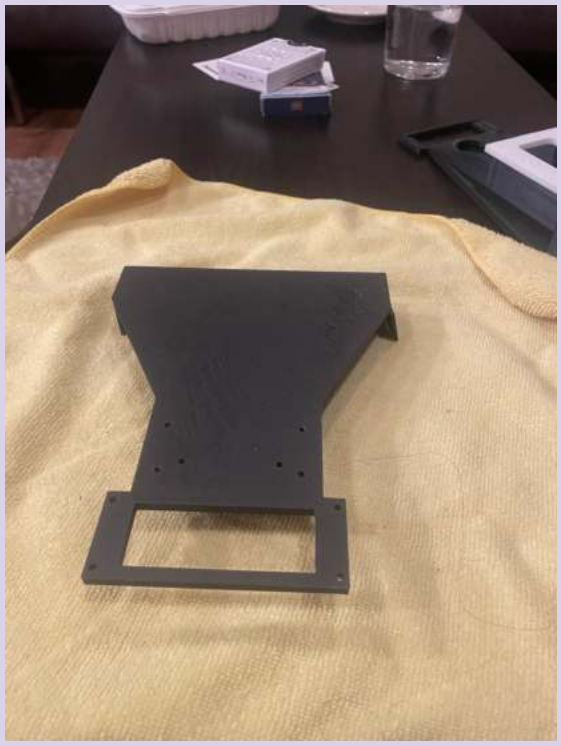

MODE
ON
OFF
ZERO





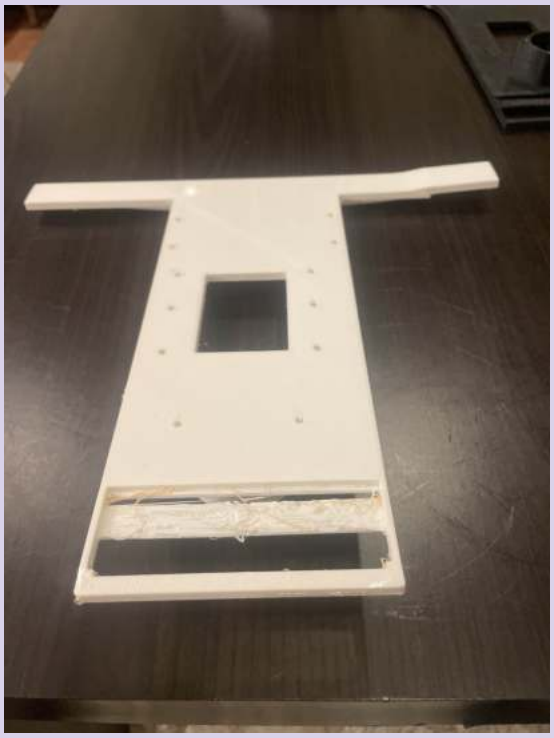
Final Version:

Iteration:

Iteration:


Iteration: Jacob V2 of yellow motor chassis

Iteration: Jacob V4 of yellow motor chassis


Parts / Sensors used:

IR SENSOR:

- https://osoyoo.com/fr/2020/09/01/5-channel-ir-tracker-sensor/
- TCRT5000 (provided)

ECHO SENSOR:

RGB SENSOR: TCS whatever

MOTORS:

- Metal motors provided by bierman
- Hook up guide: https://drive.google.com/file/d/1oODl1KrFR_dQSquRnKi8Wmh0uSoA_Low/view

How was the bot built:

First we considered our options for various different parts. We knew that we couldn't use the chassis provided because we had the goal of being better than everyone else; if we used the same parts, we would end up the same quality.

We began the build process by designing the robot chassis digitally. Team members used different CAD tools depending on their expertise and available devices — Jacob used Shapr3D on an iPad, while Avinh worked in Blender to refine various parts of the robot's frame.

The 3D-printed components went through multiple iterations to ensure proper fitting of sensors, motors, and the Raspberry Pi. These print jobs were completed using a mix of printers available at the SFSU Innovation Hub, including Creality and Sovol SV07+ models. We chose whatever machine was available at the time, leading to flexible but sometimes inconsistent print settings that required troubleshooting and reprints.

We took into consideration the weight distribution, the distance from wheel to center of mass to ball caster to get efficient rotations that did not sway our whole car.

Initial prints had issues such as IR sensor mounts and motor holders not fitting correctly. Later, the team adjusted the CAD designs for these mounts and reprinted the parts with better tolerances. The top half of the chassis broke during testing, which prompted a redesign to thicken the structure and increase durability.

We also used a ball caster because after researching efficient line racing robots, that seemed to be the status quo. We also had our own IR Sensor (that in the end didn't really make a difference), and metal motors because they are more precise (but also didn't really matter because it was slower than plastic motors).

The final design included:

- Mounts for five front-facing IR sensors and two angled wing IR sensors to improve line-following performance.

- **Echo sensor holders on the front and side of the chassis for forward and lateral obstacle detection.**

- **A Raspberry Pi mount for clean internal wiring and physical protection.**

After assembling the final print, components were mounted and secured using screws and zip ties, especially in areas with high stress or alignment risk. The switch to metal motors further increased the structural needs, which were met by strengthening the motor mounts in the CAD model.

What libraries/software did we use in our code:

Of course! Here's a breakdown of the libraries and software you're using:

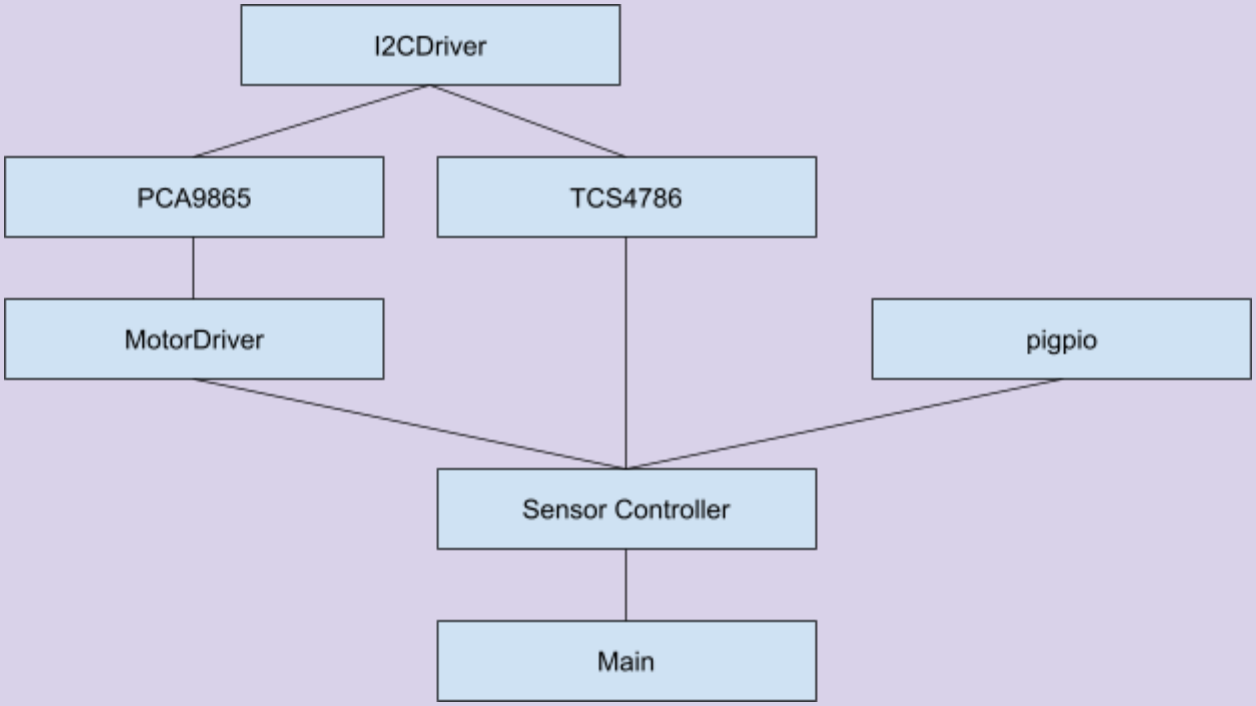
- PIGPIO Library: A versatile library for Raspberry Pi that enables low-level GPIO control, including PWM, interrupts, and precise timing functions. It's great for working with sensors, motors, and other hardware components.
  >https://abyz.me.uk/rpi/pigpio/
- Linux I2C Interface: The Linux kernel provides a way to communicate with I2C devices, allowing you to interact with sensors and drivers using a standardized interface. It's a fundamental tool for embedded systems.

  >https://www.kernel.org/doc/Documentation/i2c/dev-interface

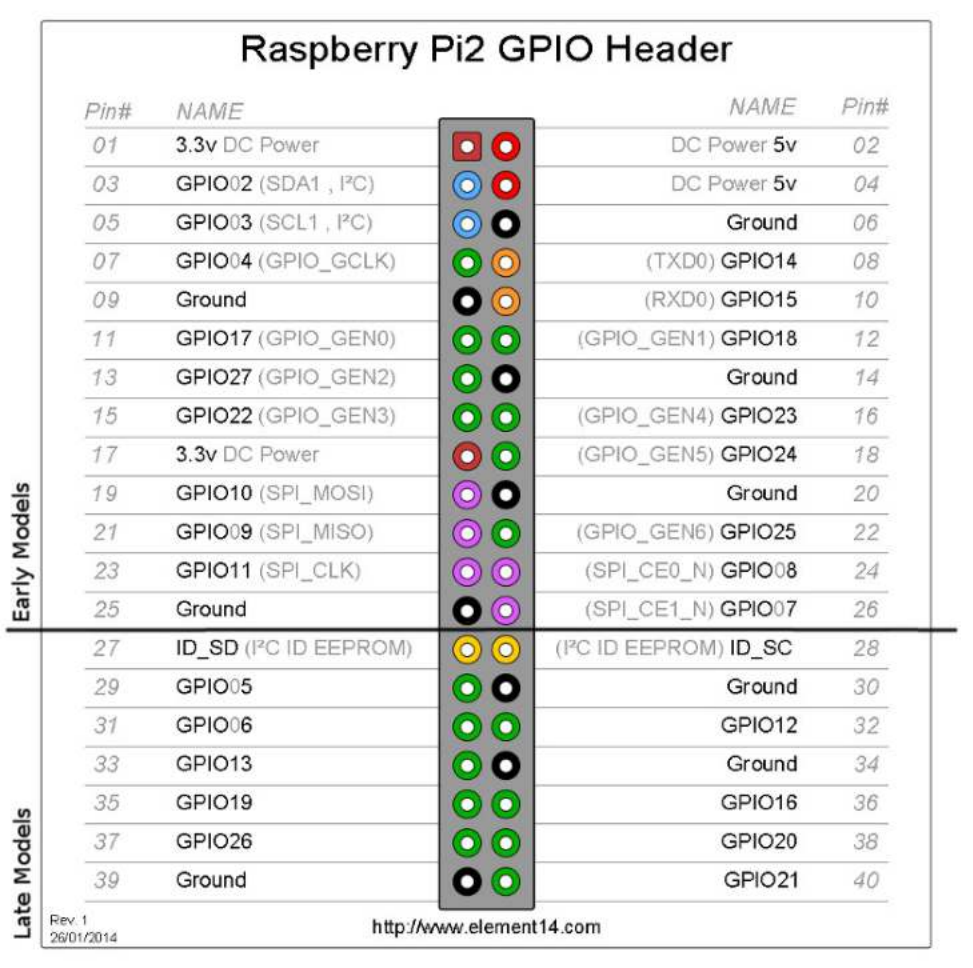
- Waveshare Motor Driver HAT: This board helps control DC motors via I2C, making it easy to manage speed and direction without complex wiring. It's useful for robotics and automation projects.
  >https://www.waveshare.com/wiki/Motor_Driver_HAT
- Waveshare TCS34725 Color Sensor: A high-accuracy color sensing module that detects RGB colors and provides readings through the I2C bus. It's often used for color detection in robotics and industrial applications.

  >https://www.waveshare.com/wiki/TCS34725_Color_Sensor

Flowchart of our code:



Pin assignments:



Button

Physical Pin / GPIO Pin	Button Pins
Physical Pin … / GPIO …	VCC (Power)
Physical Pin … / GPIO …	IN (Input / Ground)

IR Sensors

VCC Pin for all Echo Sensors: Physical Pin 4 / 5V Power

GND Pin for all Echo Sensors: Physical Pin 6 / GND

Physical Pin / GPIO Pin	Associated IR Sensor For IN
Physical Pin 29 / GPIO 5	Leftmost (No. 1)
Physical Pin 31 / GPIO 6	Left from middle (No. 2)
Physical Pin 33 / GPIO 13	Middle (No. 3)
Physical Pin 35 / GPIO 19	Right from middle (No. 4)
Physical Pin 37 / GPIO 26	Rightmost (No. 5)

Bierman Board for Metal Motors

Physical Pin / GPIO Pin	Bierman Board Pins
Physical Pin 4 / 5V	5V
Physical Pin 6 / GND	GND
Physical Pin 19 / GPIO 10	MOSI
Physical Pin 21 / GPIO 9	MISO
Physical Pin 23 / GPIO 11	SCLK

Echo Sensors

VCC Pin for all Echo Sensors: Physical Pin 2 / 5V Power

GND Pin for all Echo Sensors: Physical Pin 14 / GND

Physical Pin / GPIO Pin	Associated Echo Sensor for TRIG
Physical Pin 13 / GPIO 27	Left
Physical Pin 15 / GPIO 22	Middle

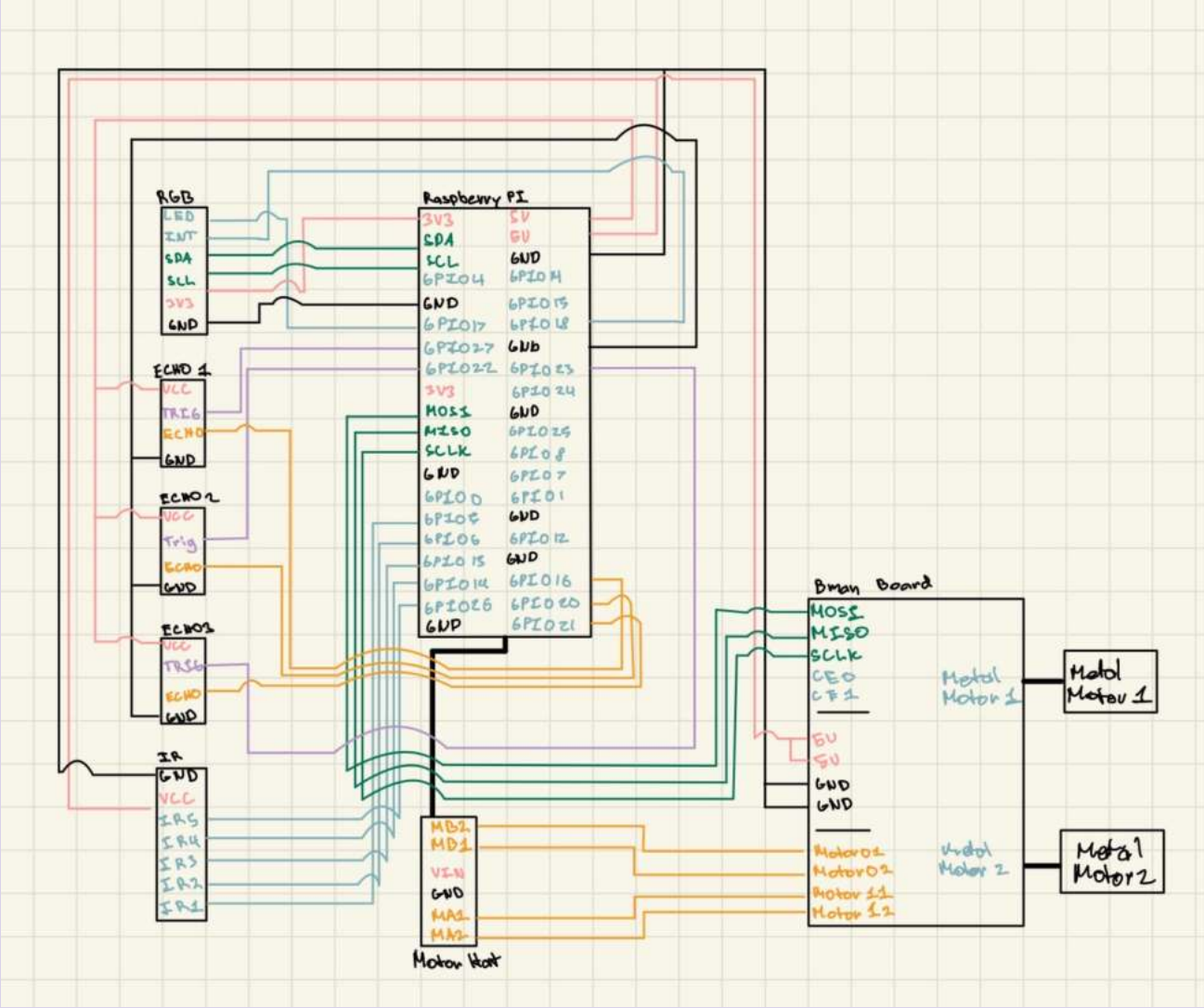
Physical Pin 16 / GPIO 23	Right

Physical Pin / GPIO Pin	Associated Echo Sensor for ECHO
Physical Pin 36 / GPIO 16	Left
Physical Pin 38 / GPIO 20	Middle
Physical Pin 40 / GPIO 21	Right

RGB Sensor

Physical Pin / GPIO Pin	RGB Sensor Pins
Physical Pin 1 / 3V3	3V3
Physical Pin 9 / GND	GND
Physical Pin 3 / GPIO 3 (SDA)	SDA (I2C Data Input)
Physical Pin 5 / GPIO 4 (SCL)	SCL (I2C Clock Pin)
Physical Pin 23 / GPIO 11	Interrupt Output (Open drain output)
Physical Pin 32 / GPIO 12	Light emitting Diode

Hardware Diagram:



What worked well:

The PID-based line-following logic worked consistently after fine-tuning, especially with the corrected sensor inversion logic. Our sensors return HIGH on black tape, so we inverted the values before passing them to the PID controller.

Upgrading to metal motors and powering them separately with an external battery noticeably improved speed and maneuverability.

The modular sensor mounting system made it easy to iterate on hardware design.

Adding wing IR sensors significantly improved the robot's ability to handle sharp (90+ degree) turns.

Implementing a physical button for starting the robot was simple but effective for testing and demonstration.

Follow the Line PID algorith => Found a function to do things and translated that into our algorithm

Threading => IR Sensors could be done all at once as well as Echo Sensors

Issues and Solution we came across:

Sensor Management and Threading:

- **Initially, each sensor had its own thread, which overwhelmed the Raspberry Pi Zero. We resolved this by consolidating sensor reads into grouped threads — one for IR sensors and one for Echo sensors — which improved performance and stability.**

I2C Bus Conflicts:

- **Sharing the same I2C bus for the motor controller and RGB sensor caused communication failures. To resolve this, we switched to a Raspberry Pi 4, which allowed for separate I2C buses and isolated device control.**

CAD and Print Issues:

- **Several redesigns were required due to misfit parts and printing errors. We refined the CAD models after testing, learned proper 3D printing settings, and reprinted components successfully.**

IR Sensor Logic:

- **At first, the PID was behaving incorrectly due to misunderstanding the sensor output logic (default high). Once we realized this, we corrected the data processing by negating IR sensor readings before passing them to the PID loop.**

Obstacle Avoidance:

- **Our original forward-facing ultrasonic sensors had limited detection range for lateral objects. We relocated one sensor to the side (perpendicular to the forward sensor) for better obstacle detection during avoidance maneuvers.**

Structural Failures:

- The top half of the robot chassis cracked during testing. We reinforced this area in the next print iteration and secured weak points using zip ties, which held up well.

Sharp Turns:

- The robot struggled with 90-degree turns. Initially, we tried a simple pause method during turns, but it proved unreliable. We implemented logic to "repeat the last extreme turn" (e.g., hard left or right) when the robot lost line detection, which improved cornering performance.

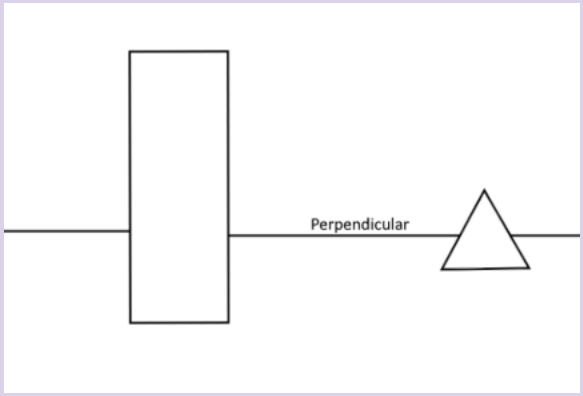
Power Limitations:

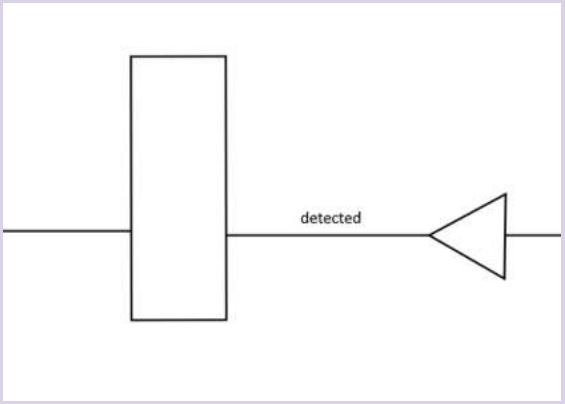
- The motors were underpowered when driven from the Pi's onboard power. Introducing an external power source to the motors dramatically improved the robot's performance.

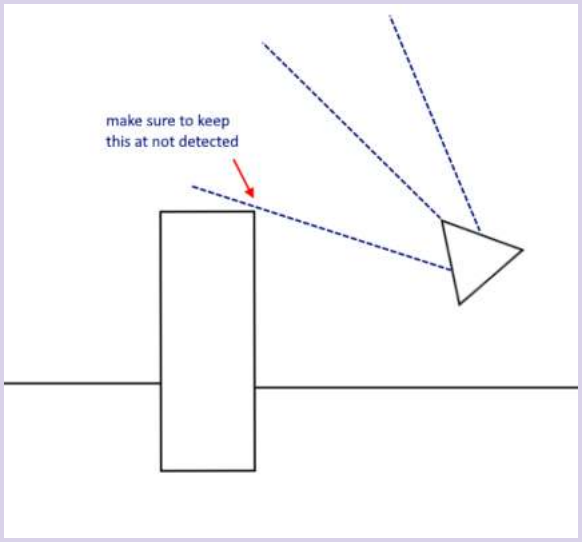
Realized that the Pi might not be able to handle the number of thread we have for each individual sensor => Combine all reads of the IR sensors and Echo sensors into their own one thread

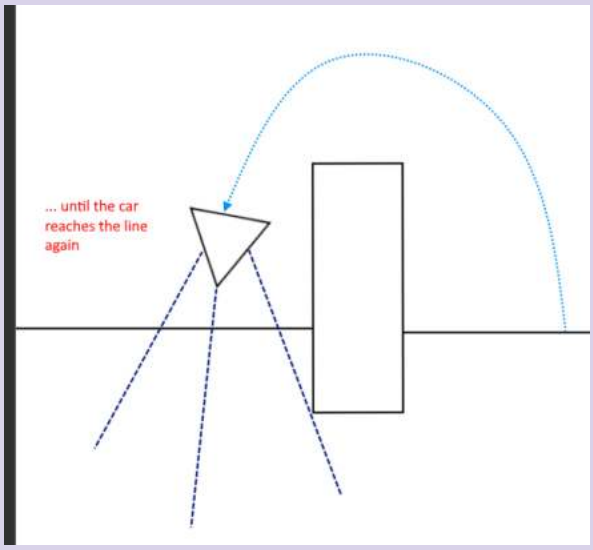
Having trouble using our Rasbperry Pi Zero for our car => Switched to the larger Rasbperry Pi 2 for multiple I2C buses

How to do Object avoidance with a car that have its echo sensors in front of it









Having trouble with detecting 90 degree turns with our current 5 sensor array => Need to add 2 new wings IR sensors to allow for this

Object detection might be difficult without current array of echo sensors => Move the rightmost one to be perpendicular to the one facing forwards // Have a sensor be on the side of the car