CSC615 - Line Following Robot
Team Dammit, Bobby!

Group Members

Arjun Bhagat | 917129686
Matt Stoffel | 923127111
Joseph Behroz | 923625817
Varun Narravula | 923287037

Primary Github

water-sucks
https://github.com/CSC615-2025-Spring/csc-615-term-project-water-sucks

Task Description

Our goal was to design and implement an autonomous line-following robot capable of:

1. Following a high-contrast line on the floor
2. Detecting and avoiding obstacles, and resuming line following after
3. Responding to color-based commands via an RGB color sensor
    - Red stops the robot.
    - Green triggers a “barrel roll” maneuver.

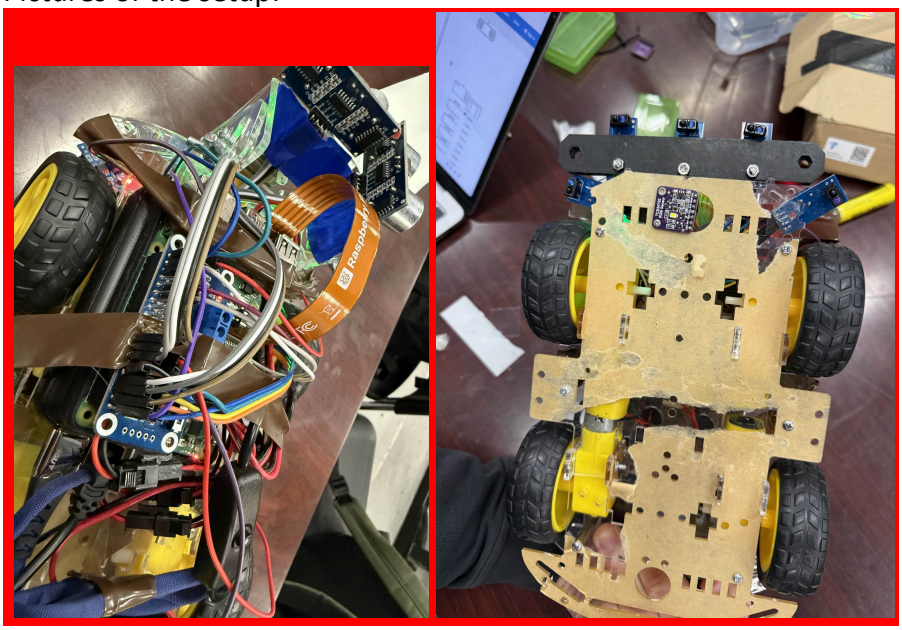
Building the Robot

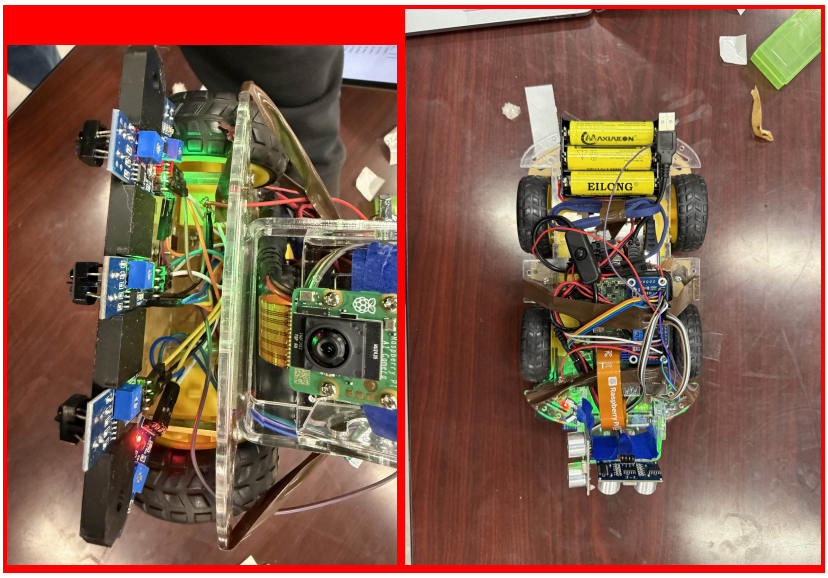
A summary: we built the robot with hopes and dreams.

The professor supplied pretty much 100% of the parts for this car. The frame was a laser-cut piece of plastic, and the motors are connected to it with terrible plastic mounts. Each pair of motors on each side shared a connection to a single motor HAT. All parts were connected with screws when it was easy, and tape otherwise; in fact, most of the connections were made with tape, since we were too lazy otherwise.

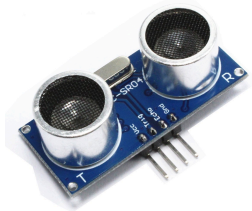
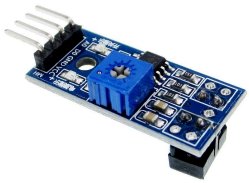
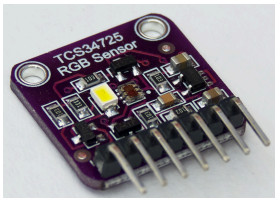
There are 5 line sensors mounted to the front; three primary, and two secondary, with a 3D bracket. Additionally, we used the camera mount to hold up the sonar sensors, with some more tape to secure the sonar sensors. The camera used a ribbon cable that we threaded through the mount; however, we never ended up using it in the program because we couldn't get it to work properly with our drivers.We reduced the number of wires by splicing together wires and having a common VCC and ground pin for all line sensors. There is also no breadboard involved; all wired connections are directly to the Pi.

Pictures of the setup:





Parts/Sensors Used

		
HC-SRO4 ultrasonic sensor (x2) - used for obstacle detection and avoidance	TCRT5000 IR sensor (x5) -used for detecting line based on reflectivity	TCS34725 RGB color sensor (x1) - to run actions based on detected line's color

Libraries/Software Used

- C standard library (<math.h>, <signal.h>, etc.)
- A home-grown GPIO pin interaction library, based on past assignments
- Vendored libtcsoup code from Assignment 5 group work for color sensor
- Zig build system (via build.zig) and GNU Make for build orchestration

Code Implementation

To start, we set up a C project, and used the Zig build system to cross-compile the project for the `aarch64-linux-musl` platform so that we could work on our host machines (macOS and Linux `x86_64`) and only copy the needed executable to the machine. From here, we can copy the I2C layer code and motor code from assignment 3 to run the motors. Our build only uses a single motor HAT, so there's no need to change I2C addresses or anything like that.

For line following, we implemented a control loop using a process called proportional-integral-derivative (PID) control that continuously reads input from five of our line sensors and computes the robot's deviation from the center of the line. Low reflectivity indicates a line, while high reflectivity indicates a non-line state. The PID controller uses this error to generate a correction value based on proportional and derivative terms (with integral disabled to avoid windup). This correction is then used to steer the robot by adjusting the speed of the left and right motors. If the robot veers left or right, the PID output compensates by speeding up one side or reversing direction on the other when necessary.

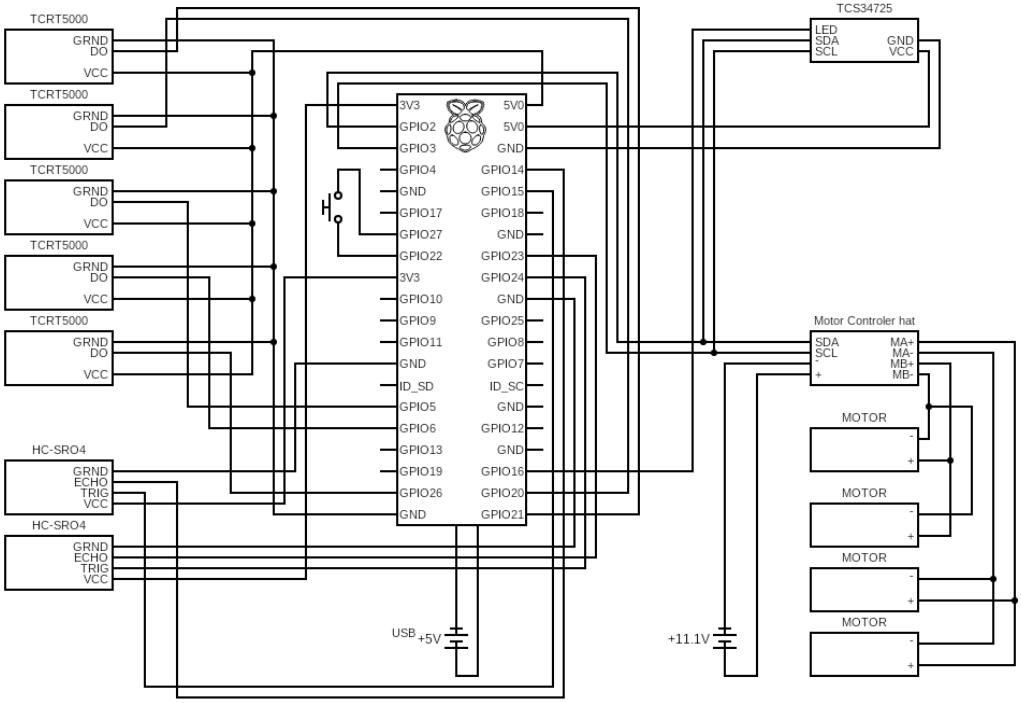
After PID works, we can then attach the color sensor code from assignment 5. Instead of bringing it in like an actual dependency with the Zig build system, we decided to vendor it and keep the source code directly in the repository for simplicity. That way, we could also refactor it slightly to use the I2C layer that we already use for our motor HAT, just with a different target address. Reading the color values will be done in a separate thread from the PID calculations on the main thread, since the TCS34725 sensor is quite slow with its readings. We need to detect two colors: red for stopping the program, and green for doing a donut. In order to detect only these two primary colors, we'll use a comparative approach: if there's more red or green above a certain threshold than the other colors, it's most likely that color. It's rough, but it should work.

Lastly, we need obstacle detection and avoidance. This requires the two sonic sensors. In order to initially detect the object, we can just check if the distance is low enough to trigger the obstacle avoidance code. We'll switch off PID for this, since we're not trying to follow a line at this point, and then turn ~90 degrees to the left and move off the line in this direction. Once we've moved a certain amount, we can turn it back to the original position, and continue moving forward. This is where the side sensor comes in: we'll keep moving forward until the side object is not detected anymore, and after it's not detected, start returning to the line. Once a line is detected with two sensors that are on, we can switch back to the PID routine, and continue as normal, since the line has been found.

Pin Assignments

20	Line sensor 1
21	Line sensor 2
5	Line sensor 3
26	Line sensor 4
6	Line sensor 5
23	Ultrasonic front echo
24	Ultrasonic front trigger
14	Ultrasonic side echo
15	Ultrasonic side trigger
27	Button input
22	Button power (3.3V output)
16	LED control for color sensor

Hardware Diagram



What Worked Well

Cross-compilation with Zig and subsequent copying with `rsync` was super efficient and enabled our team to work extremely fast, with tight feedback loops. We also worked pretty well together as a team, especially since we weren't stuck in a bad dev environment.

We managed to get PID and line following working in a single day, so that was quite nice.

Also, object detection was reliable in almost 100% of cases, given the object was able to be seen by the sensor (i.e. if it was tall enough).

Issues/Resolutions

PID would be shaky at times with non-straight lines (i.e. old tape, badly painted lines on the cloth) due to no smoothing. This wasn't quite an issue with the real track, but it was more so an issue during development and slowed down the time and disguised other problems.

Batteries vs. non-batteries would require different speeds to be changed in the code, which means if the batteries were changed or degrading, we had to recompile and recopy. This slows us down a bit as well.

Obstacle avoidance didn't quite work in some instances where the bot would catch the line in reverse and start going backwards; we fixed this by playing around with the values for turning back to the line after avoiding the obstacle, as well as by only using the presence of two sensors in order to turn on proper PID again.

The most grating issue is that the color sensor didn't work for green values properly, and it also tended to detect false positives for red and green unless we changed the sensitivity thresholds for them significantly, and this varied from the real to testing tracks too significantly to the point where it ended up detecting false positives in the real runs; we ended up scrapping the green detection code and only going with red stopping, since that was a little more reliable.

Group Photos