CSC 615 – UNIX Programming

LINE FOLLOWING CAR WITH OBSTACLE DETECTION

Professor: Robert Bierman

Group Team Name: Team Undecided

Team Roster & Github Information:

Bradley Justice - https://github.com/pambalos
Shem Cheng - https://github.com/jeshem
Vito Gano - https://github.com/vgano1
Eric Leow - https://github.com/ericmeow

Working Repo:
https://github.com/jeshem/CSC615-Team1

Final Repo:
https://github.com/CSC415-Fall2020/group-term-project-pambalos

Media Drive:
https://drive.google.com/file/d/1YfblVjqDc4dAfWtTkQJeKLvcVddsDzU7/view?usp=sharing

Makefile:

```
output: main.c motorcontroller.c
        gcc main.c echosensor.c motorcontroller.c irSensor.c lineSensor.c
-lwiringPi -lpthread -o driver
        ./driver
```

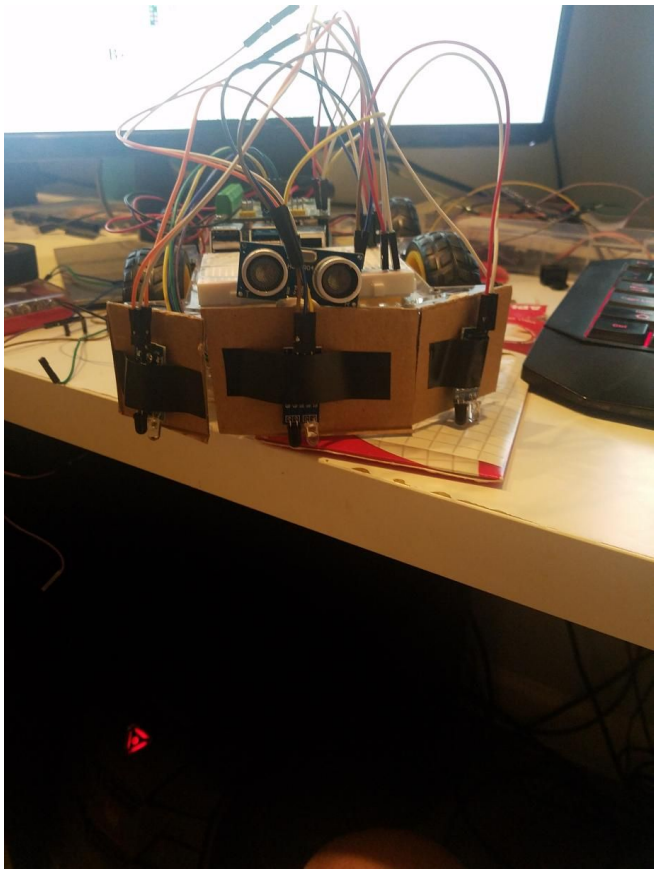
Task Description:

In the Group Assignment we were tasked to build a self-driving car that would be able to traverse through a course and be able to go around an obstacle. This process involved creating a car from barebone components - with a frame, a number of motors, and a variety of sensors to pick and choose from. This meant that the entire car design and

code design was entirely up to us. We got to decide to build any design that we felt was most likely to follow the line, and that meant coming up with initial designs and reiterating on them repeatedly after testing.

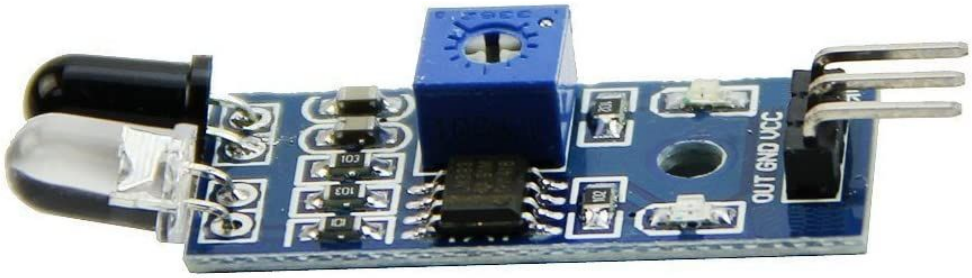
There were three primary pieces to this project. The first was simply to create a physical car that could pick up and detect a line as well as to detect obstacles that blocked the car's way. The second piece was to make the car successfully follow a line on its own based on whatever sensor inputs it was reading. The last piece was to also give the car the ability to avoid obstacles.

Building the Robot (include photos)



Parts / Sensors Used (include photo, and part numbers where applicable, such as HC-SR04 for the sonic echo sensor):

1. 3 IR Sensors used as line sensors (OSOYOO 10 Item Model Number LYSB01I57HIJ0-ELECTRNCS)



Specifications:

- Board Size: 3.1CM * 1.5CM / 1.22 * 0.59"

2. 2 Ultrasonic Sensors (Part Number HC-SR04)

Specifications:

- Use voltage: DC 5V
- Level output: high 5V low 0V
- Detection range: 0.78~196 in/ (2cm~500cm)

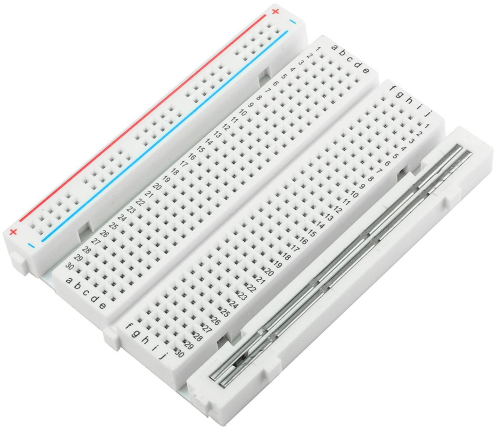
- High accuracy: up to 0.12 in/(0.3 cm)
- Trigger Input Pulse width: 10uS
- Dimensions: 1.3 x 0.4 x 0.1 inch/3.03 x 1 x 0.25 cm (L*W*H)

3. 2 DC Motors (Amtrader ASIN Number: B07DDC3ZBK)

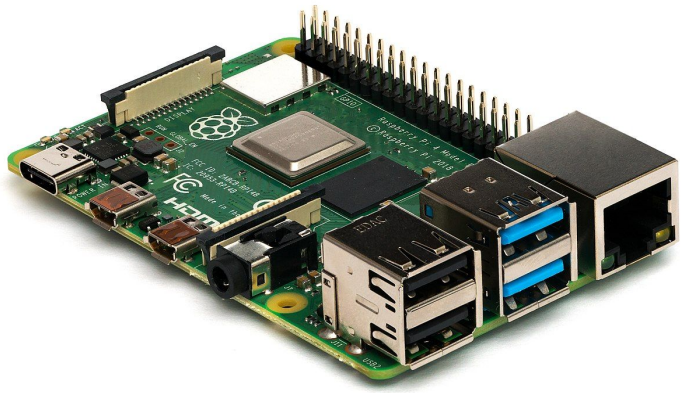
Specifications for the DC Motors:

- Size of Motor: 7 x 2.2 x 1.9cm/2.76" x 0.87" x 0.75"
- Maximum torque: 800gf cm min (3V)
- No-load speed: 1:48 (3V time)
- 3V~12VDC

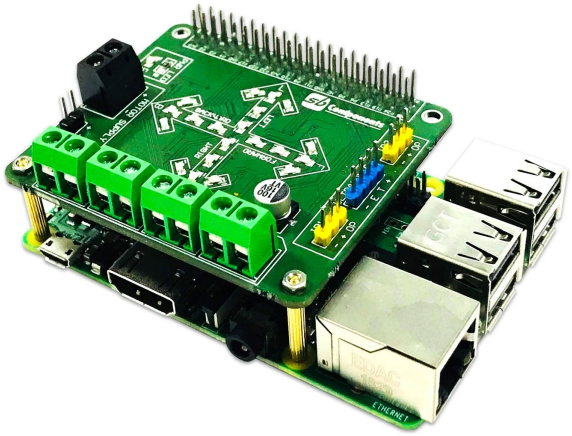
4. 1 Bread Board (deyue Item Model Number: 7545924028)



5. 1 Raspberry Pi 4 (Model B)



6. 1 Raspberry Pi Motor Shield (SB Components Motor Shield)



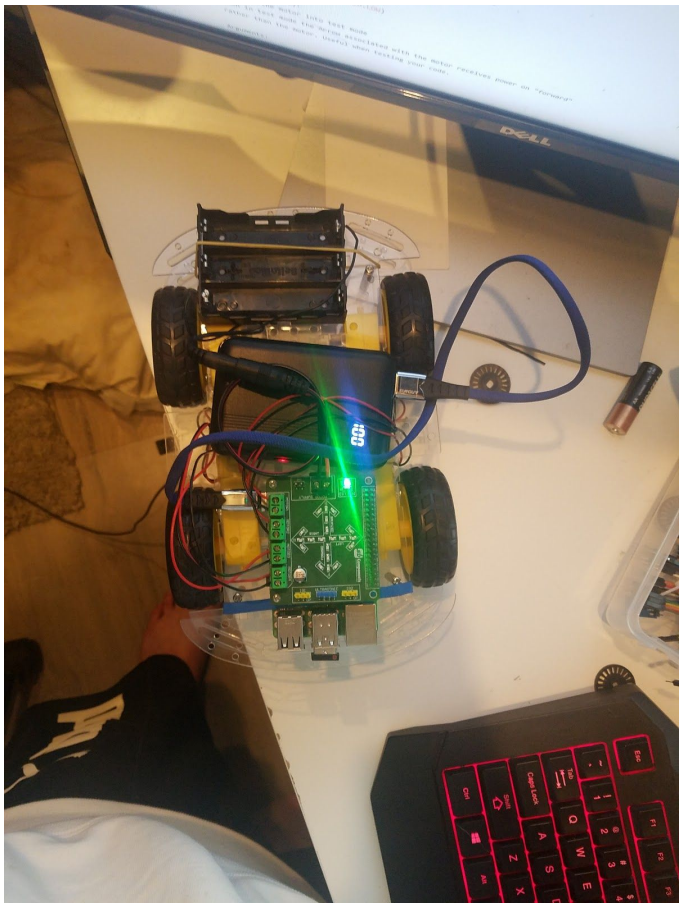
7. 1 Plastic Car Chassis (Perseid ASIN Number: B07DNXBFQN)

   This contains the whole kit but we used this brand for the car chassis.

How was bot built (photos good to include):

Some of the first things to figure out was creating a frame for our car and figuring out where to put all of the different controllers and batteries. This was a bit of a process that took a lot of finagling and tweaking on our hardware managers part. Sadly he had to do it all alone.

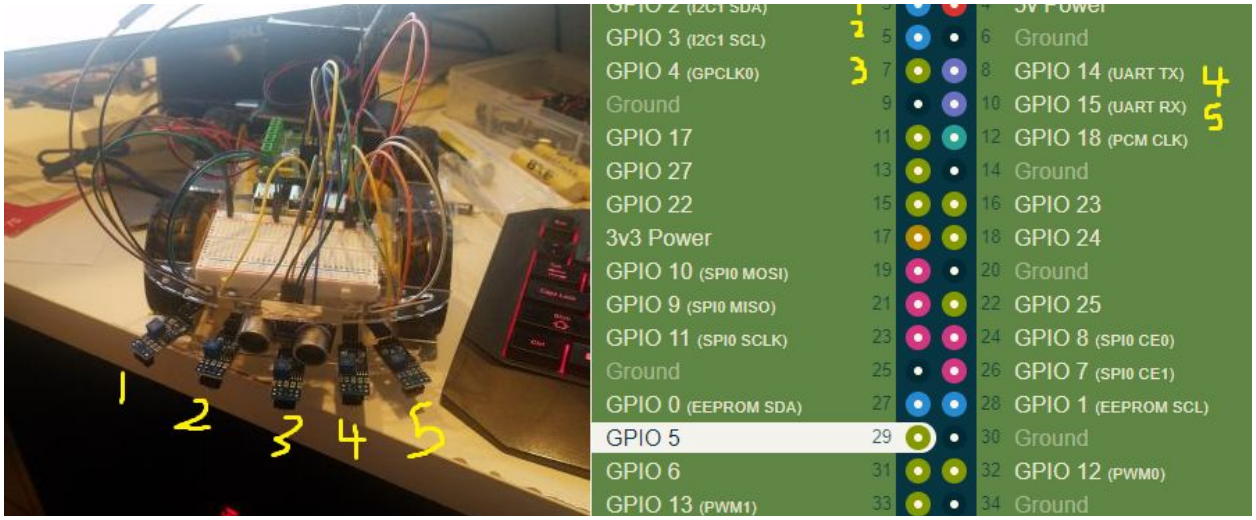
We started off with a basic frame design with all four motors attached at the expected positions at each corner:



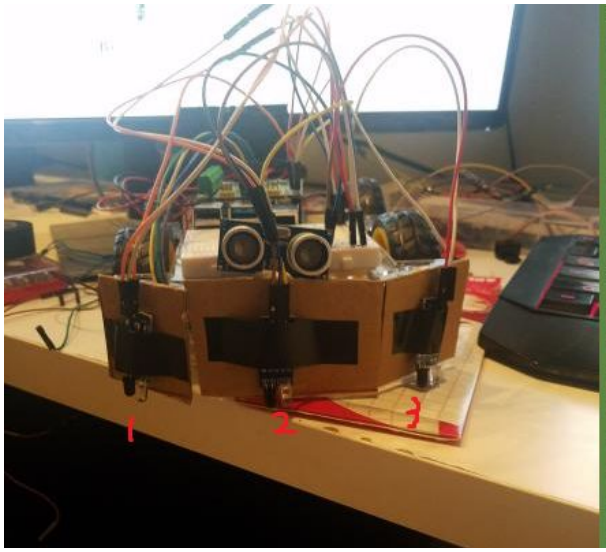
After putting this very basic car together, we started testing some basic movement code to ensure that our motors were in good working condition. A good deal of that test code still sits in the final repository.

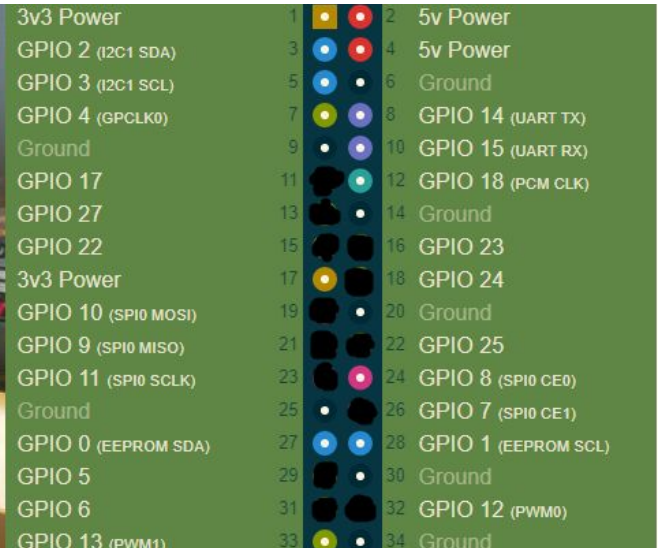
The next thing we had to decide was what sensors we wanted to include and how we would place those on the car such that we could build a control loop that would sense and follow the line.

The first design that we came up with was that of using 5 line sensors on the front of the car, one in the middle and two on either side to track the line. In addition to that, we planned on using two ultrasonic sensors to sense the distance to obstacles in front and to the left of the car.



We did implement this design at first, but we found that the line sensors were not behaving quite as we expected them to. As a result, we ended up switching to three IR sensors instead of line sensors to track the line.





We also ended up just ditching the front two motors of the car, extremely last minute, because they apparently were not working as expected. I ended up having to adjust and change the code on the day due to the unexpected loss of the two front motors... failures with miscommunications like this happened to us repeatedly and this caused a very strange dynamic wherein I would code expecting a certain physical setup only to discover that we had changed the car. I originally wrote code for a car with four motors, five line sensors on the front, and two ultrasonic sensors, but the car that we ended up with was a car with two motors, three infrared sensors on the front to track the line, and another four on the corners to detect obstacles. As a result, I had to constantly adjust for the newest configuration and try to write code that worked with the given setup at any moment.

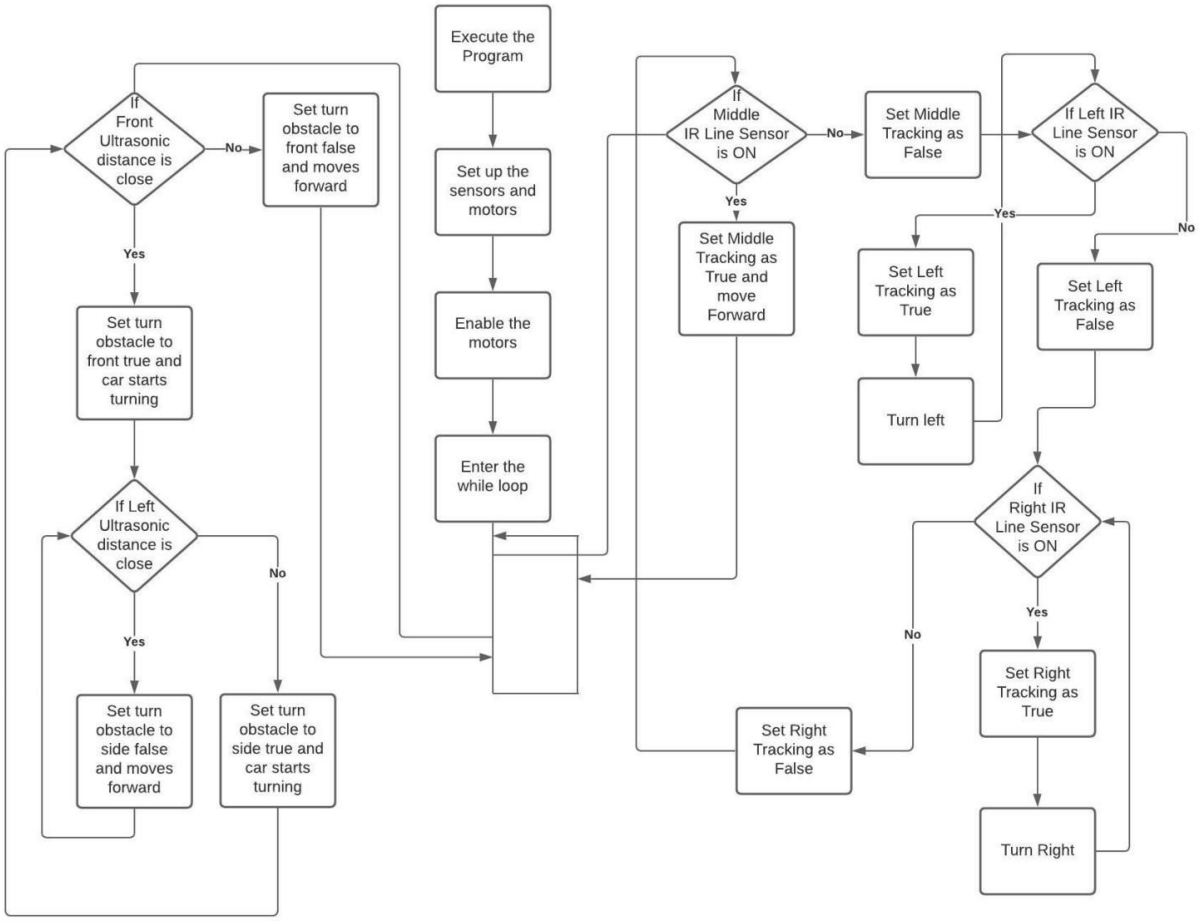
What libraries/software did you use in your code (include full reference)

Libraries that we used for the development of the car included:

- WiringPi.h
- SoftPWM.h
- Stddef.h
- Time.h
- Pthread.h
- Stdbool.h

The main library that we overall used within the project was wiringPi.h and the SoftPWM.h. The wiringPi libraries were meant to be able to interact with the motors as well as with the other devices like the sensors. The SoftPWN library is meant for us to control the power output of the motors that we have.

Flowchart of your code



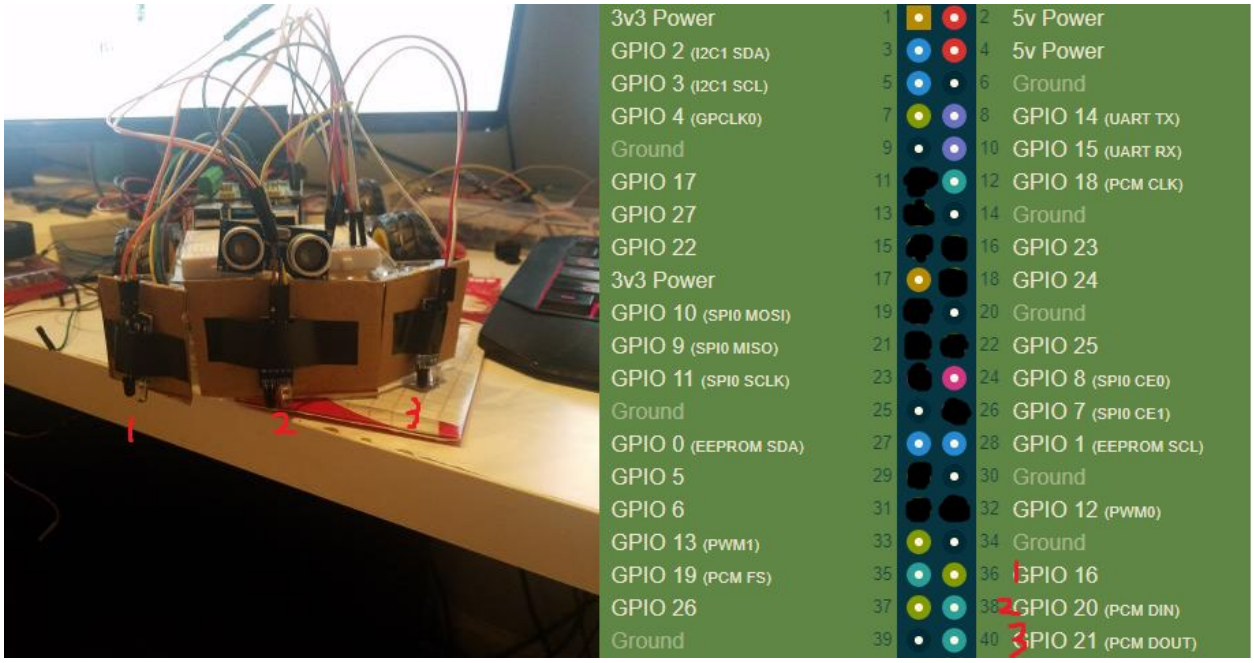
Pin Assignments you used

MOTORS

	Enable	**Control**
Front Left Motor	11	13,15
Back Left Motor	22	16,18
Back Right Motor	19	23,21
Front Right Motor	32	24,26

SENSORS

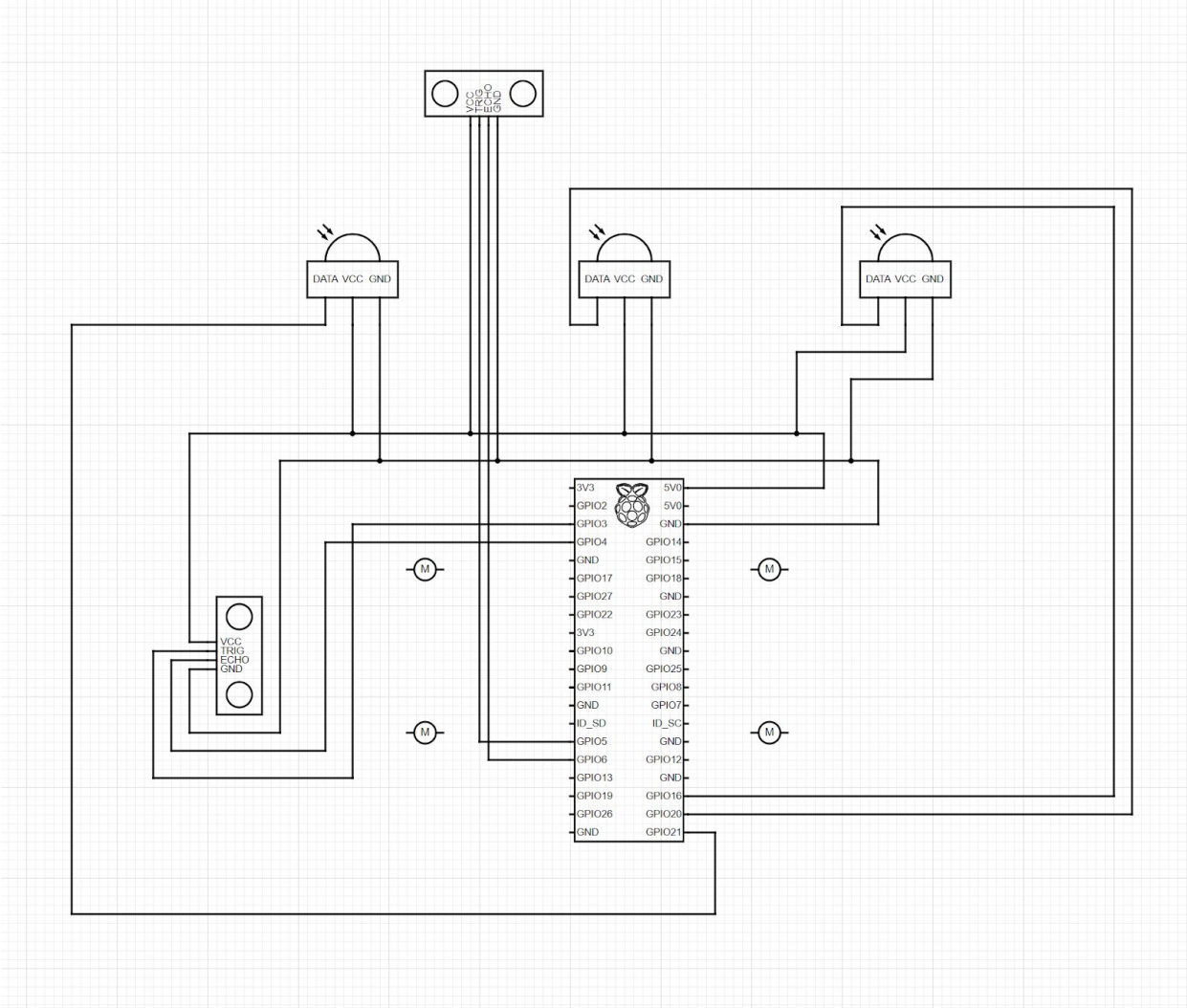
	Trigger	Echo
Front Ultrasonic Sensor	29	31
Back Ultrasonic Sensor	5	7
Left Line Sensor		40
Middle Line Sensor		38
Right Line Sensor		36



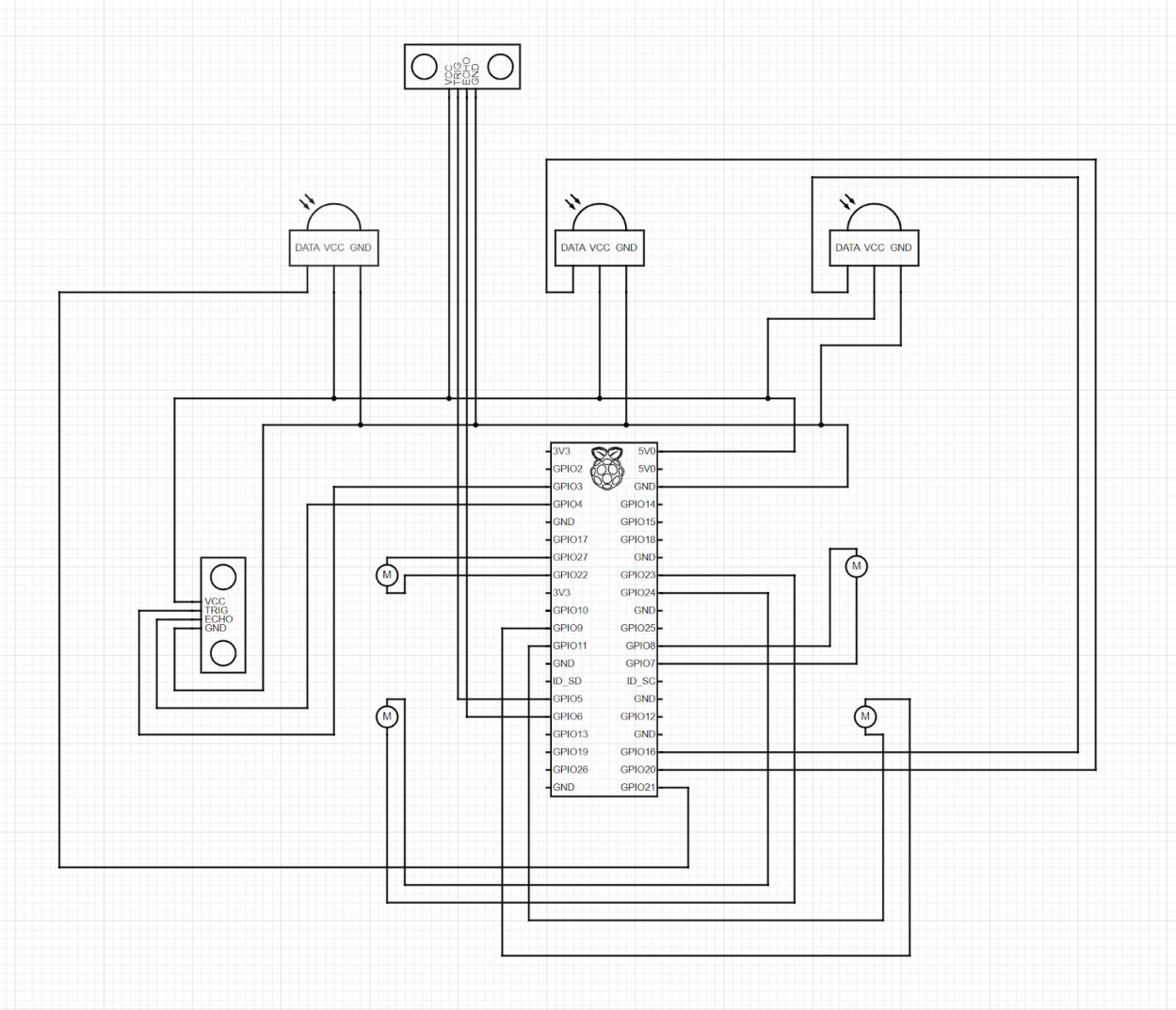
We often found ourselves resorting to this sort of diagram in order to share pin schematics, all of which had to be generated by our hardware manager Shem. We found this to be the best method for sharing pins as there was often confusion about pin numbers versus GPIO pin numbers, etc. This method allowed us to have no ambiguity.

Hardware Diagram

1st Diagram w/o Motors Connected to Pins



2nd Diagram with Motors Connected to Pins (Control Pins)



We were not able to connect the motors to a template Motor Shield as the website that we used to create these diagrams did not have a Motor Shield template that would allow us to show and illustrate where the wires were connected in the Motor Shield. Originally, we had 4 motors on the hardware however due to not being able to have the front wheels working, we had to work with only two wheels. That's why in the hardware diagram, it is shown that there are four wheels since that was initially what we originally planned to have in our car.

What worked well

What worked well with the car was that we were able to get the sensors to be able to distinguish whether the middle IR sensor was lighted up or not since it is able to distinguish whether or not the car was within the line or not. In addition, from the test run that we made for the project.

What were issues

There were some issues within the car as well. One problem was that the front motors were not working as intended and we didn't want the car to be unbalanced. We were able to resolve that issue by placing a tupperware box right beneath the front frame in order to balance out the level of the wheel. This happened more than once, wherein the behaviour of various components was unexpected and this threw off our code. Another example is simply that of how our rotation methods seemed to not always work properly. For example, both rotation, left and right methods used the same speed definitions to actually perform their rotations, but in order to make the car physically rotate, I had to force a higher rotation speed on one side than the other.

Some other issues that we faced were simply due to the nature of this project. As our hardware manager was the only one with the car, he had to take on the majority of the work when it came to actually putting the car together and testing all of the connections. I did my best to write and send test code over github and have the Shem test it on the car, and then report back to me on the behaviour, but overall this made the entire process slower than it should have been. Iterations on test code and new code took significantly longer than desired, and the feedback process was slowed to a snail's pace. If we could have worked in person, I believe that we would have been significantly more successful in fully completing this project, which included an avoid obstacles piece that we failed to complete.

Another issue that we faced was simply that not all of our group members contributed to the code base or even participated in the design and group discussions. This is easily seen from looking at the contributors on the working repository. As a result, the burden of writing the code fell unfairly on some. If I could repeat the course, I would simply pick to be the hardware manager so that I could control the rate of feedback iteration.