Team Cesla Pirates

CSC 615

5/21/21

Team members:

Feras Alayoub (Manager), ID: 917942134, Github ID: ARM-Cortex-M4

Rasul Imanov, ID: 920668590, Github ID: rimanov

Wameedh Mohammed Ali, ID: 920678405, Github ID: wameedh

Wilfredo Aceytuno Jolon, ID: N/A , Github ID: N/A

Task Description:

In this project, we are tasked to use our knowledge learned in this class about hardware and software (embedded systems) to build a functional car that is able to move freely, follow a line with the use of line detection sensors, and detect/avoid any obstacle that it faces with the use of IR sensors along its route through the line it follows. The way to approach this was through simple step oriented assignment that utilized various components of the car such as the motors, IR sensors, line following sensors, ultrasound sensors, and encoders, counters, and LIDAR sensors.

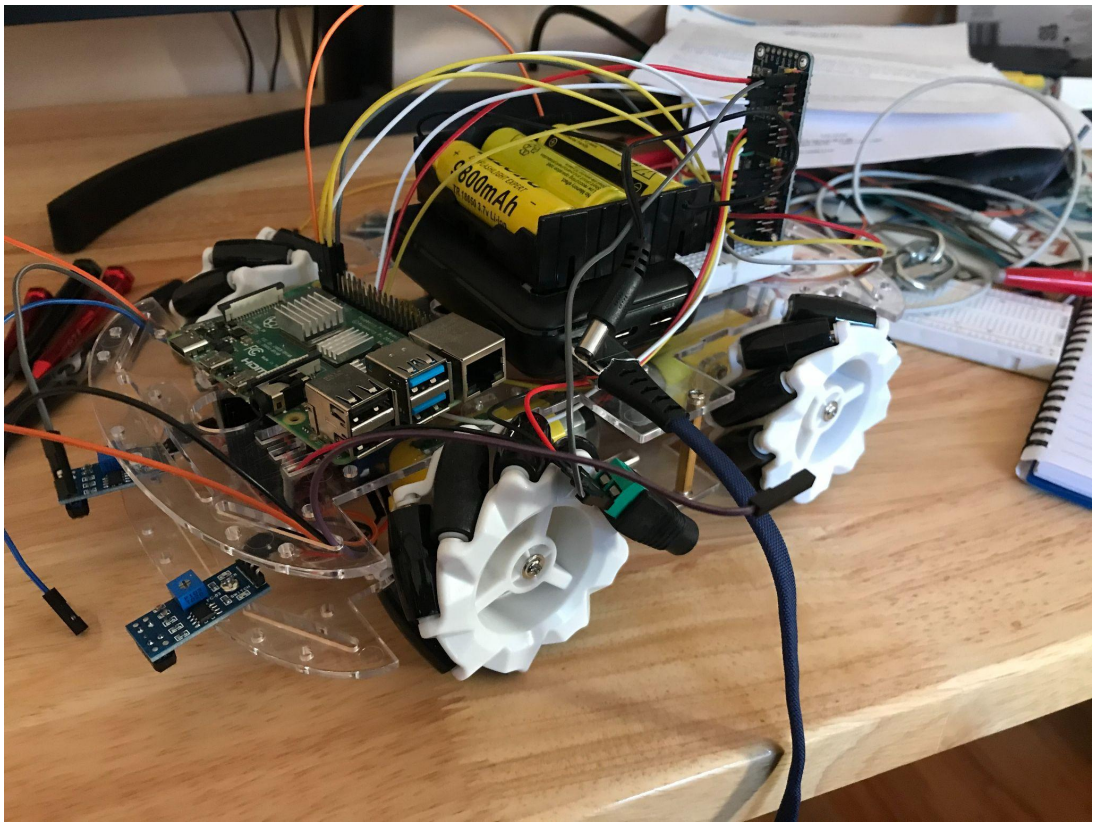
Building the Robot:

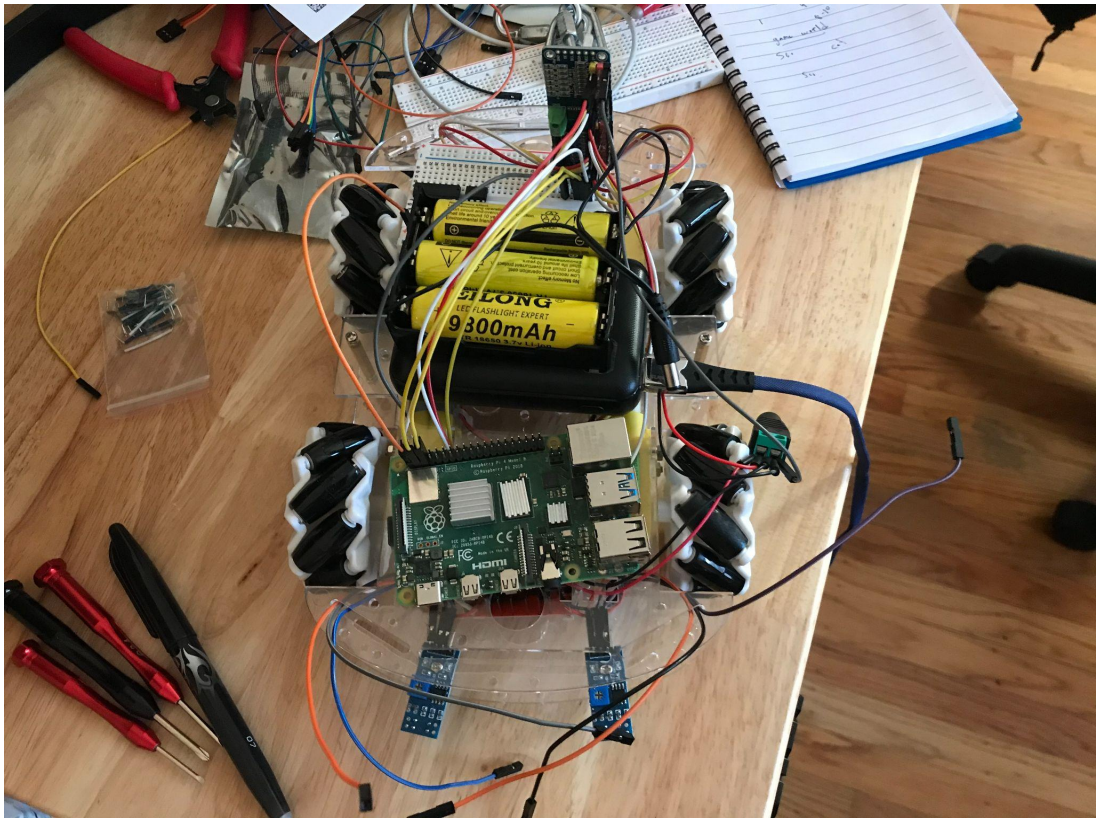
First thing our manager did was connect wheels and motors to the body of the car. Once the wheels were coupled to the body, we did a basic system integration and validation test ensuring each motor ran properly and they were all within tolerance with a 12V power source. Furthermore, the next steps were to connect the wiring to the motor drivers, and ensure even and variable power delivery. Once complete, The next

steps were to integrate and validate simple code to ensure that the previously setup hardware components were functioning as they should and run through a simple test loop. After making sure that the basic components were functional, the next step was to clean up code and test the whole unit to make sure everything was correctly aligned and the process was validated. Once the car moved properly and the wheels were properly debugged the next step was to add a control loop that ensured the motor behaved in such a way the line was followed. This was done by using two line following sensors on the bottom of the car. There is an illustration showing this below.

Once integration with motor feedback using the line sensor was established we then moved onto obstacle detection and avoidance. To do this we use a IR sensor or an ultrasonic sensor to see the obstacle in front of us and avoid it. We mount these peripheral devices on the top front of the car since it will be driving in that direction.

Pictures of car components and car build: And overall completed build missing the IR and ultrasonic Sensor. We then wire the components together with the battery pack and the pi battery source and begin testing our system as a whole unit.





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

Ultrasonic sensor HC-SR04

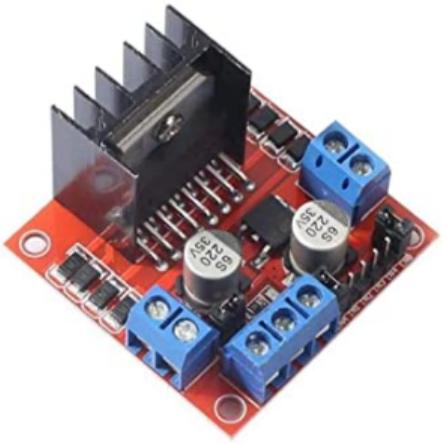
4 DC Gear motors

2 TCRT 5000 line sensors

1-2 IR obstacle avoidance sensors

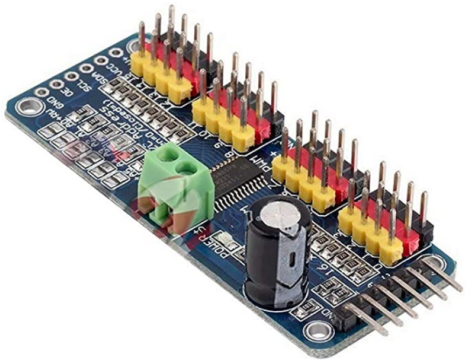
4 Mecanum wheels

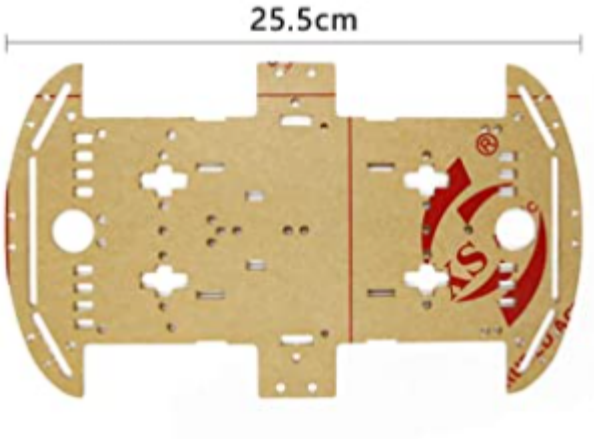
2 L298 h-bridge motor drivers



16-channel pwm PCA9685

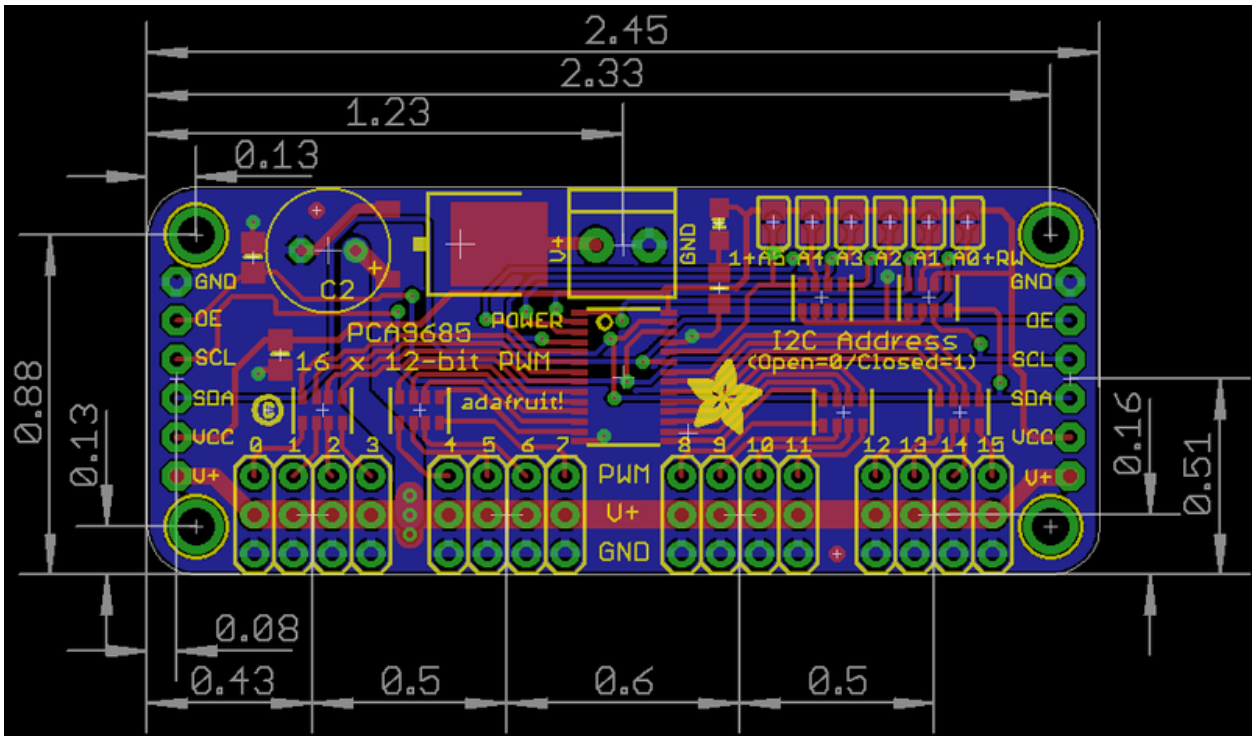
Car Chassis



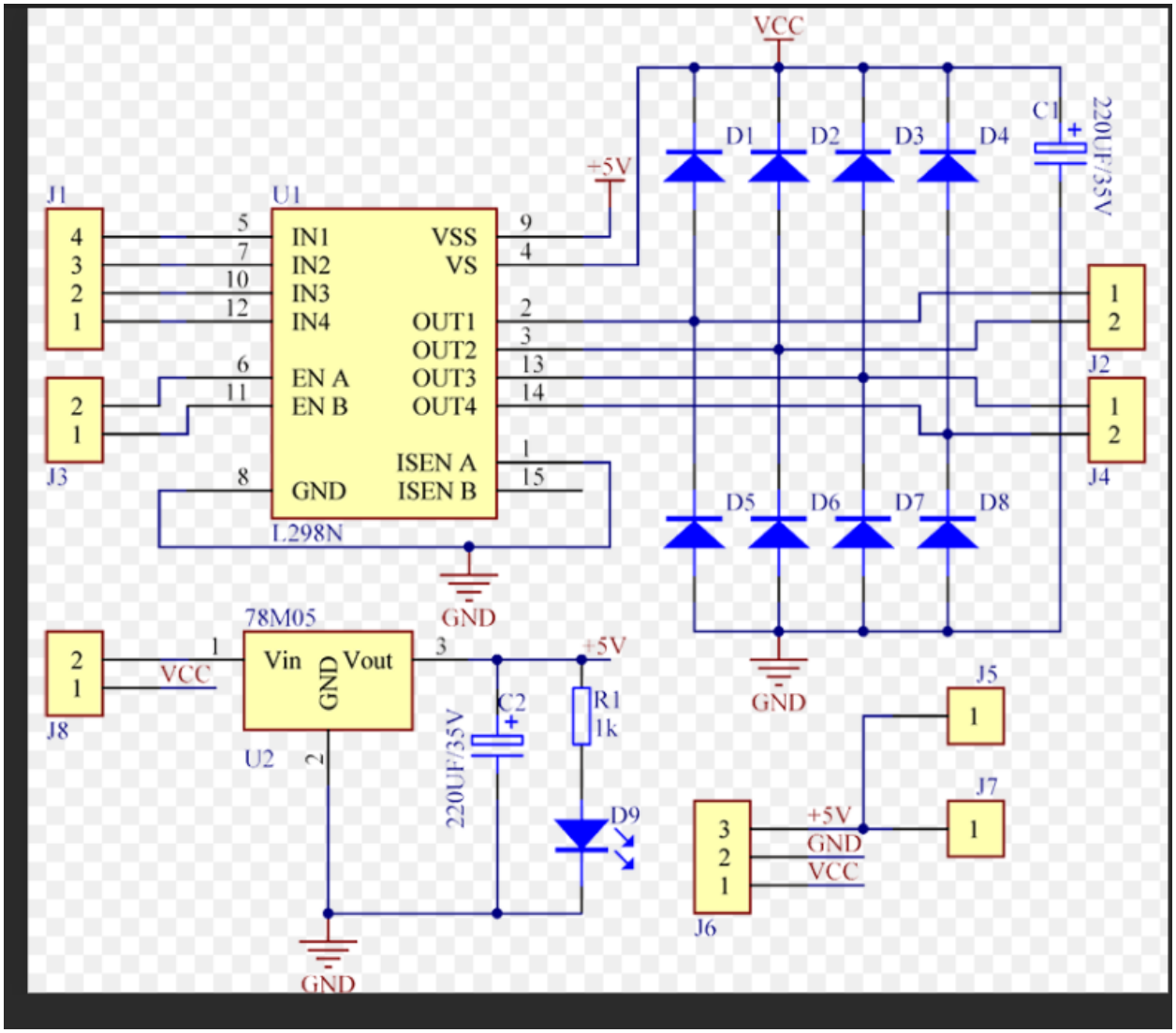


Below are also the schematics and data sheets for the l298n motor drivers and the PCA9685.

[PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller (adafruit.com)]



L298N H-Bridge Motor Driver Schematic



[L298N Datasheet(PDF) - STMicroelectronics (alldatasheet.com)]

How was the bot build:

As stated above we took a more logical approach regarding validation and integration. We first build the robot doing tiny steps that are to be tested easily and troubleshooted without much complexity. Since the robot is fully built I was unable to obtain pictures to document the process of building the robot step by step. However, as stated above the step by step process includes building a mock design and testing the components. The first component was the motors we tested the PCA9685 for the correct PWM signal. As I had an oscilloscope and Digital Multimeter, this step with debugging the PCA9685 and the L298N was fairly simple and easy to do. Once the motors had a certain tolerance that was achieved using the PWM driver and the H bridge motor drivers. I integrated line following sensors into the mix. Once again to Test and integrate these was a fairly simple task that was not too difficult, since the debug was a printf statement and the indicator that the process went well was the motors running due to proper logic this was validated and integrated. The last and final step which we had difficulties on was the Obstacle detection and avoidance. We debated using an IR for its simplicity. However, We noticed that became a mistake as IR had very little range and was easily distirubed by ambient noise and light.

What libraries/software did you use in your code:

```
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <wiringPi.h>
20  #include <wiringPiI2C.h>
21  #include <math.h>
22  #include <stdint.h>
23  #include <pthread.h>
24  #include <stdbool.h>
```

reference: http://wiringpi.com/reference/

http://wiringpi.com/reference/i2c-library/

The function we used included

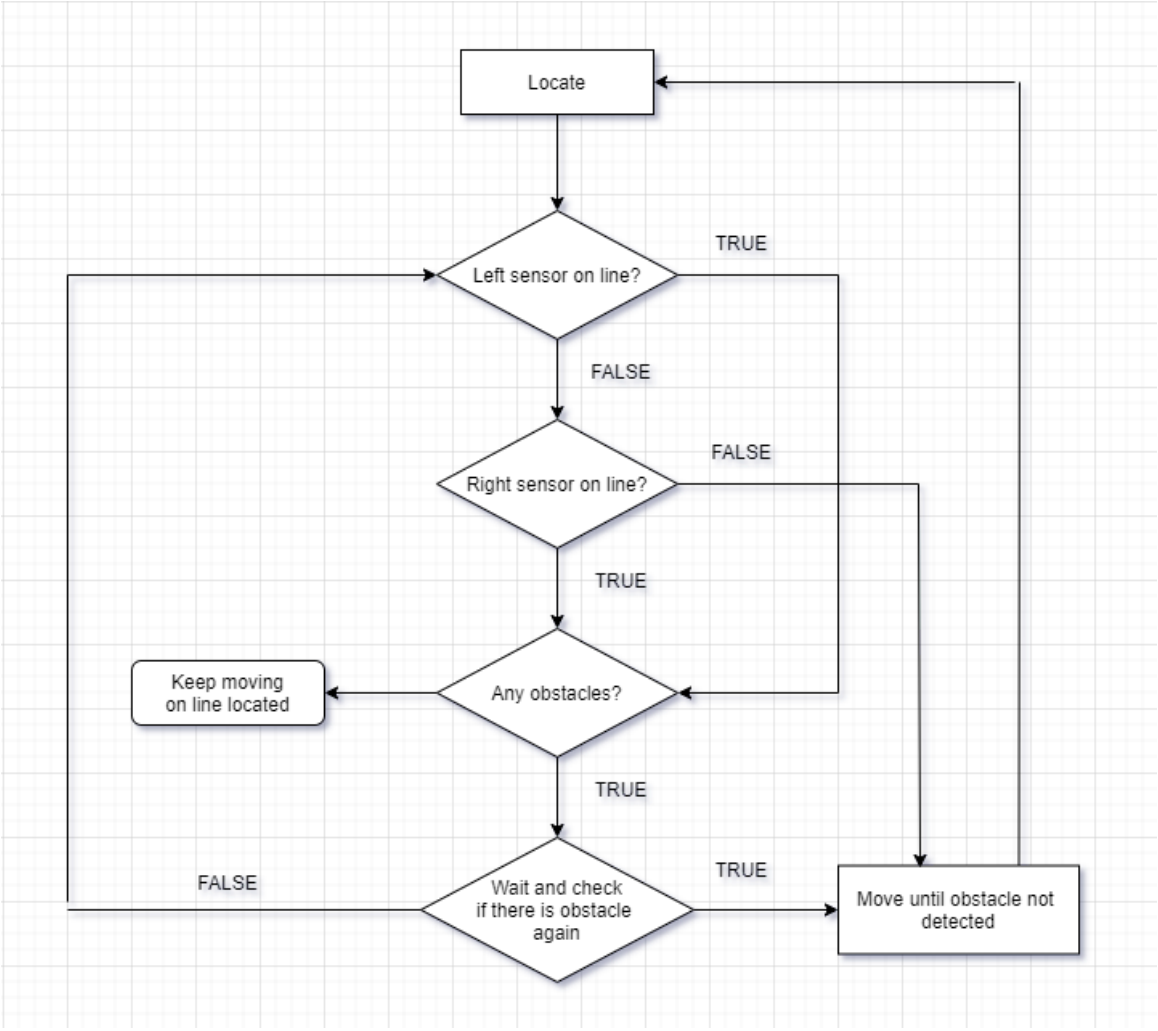
wiringPisetup() this returns a file descriptor with the description of the pins that we are able to write and read into. We need to initialize this function call before we use the digitalRead or digitalWrite() functions.

For the WiringPisetupI2C() method we do the same thing as aforementioned above and work with returning a file descriptor and checking for the return value in case of error. Once we confirm that the functions works, we then move onto calling calling I2CWrite function to write to the PCA9685 and set it's PWM register to enable PWM control of the motors.

The other libraries are C standard libraries, that introduce certain math functions, new types of data such as bool, but the most important one was the pthread header.

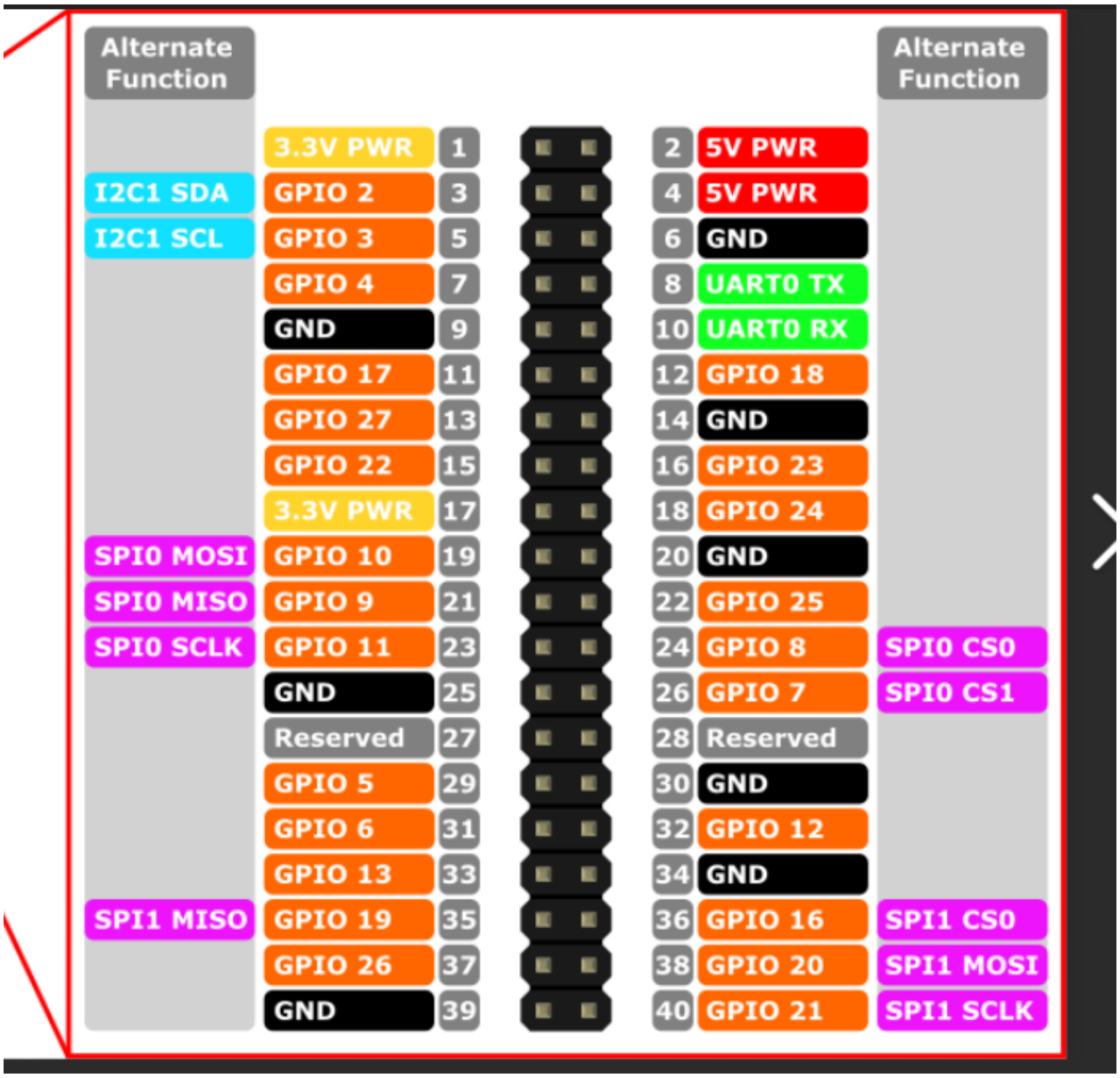
The pthread header

Flowchart of your code:



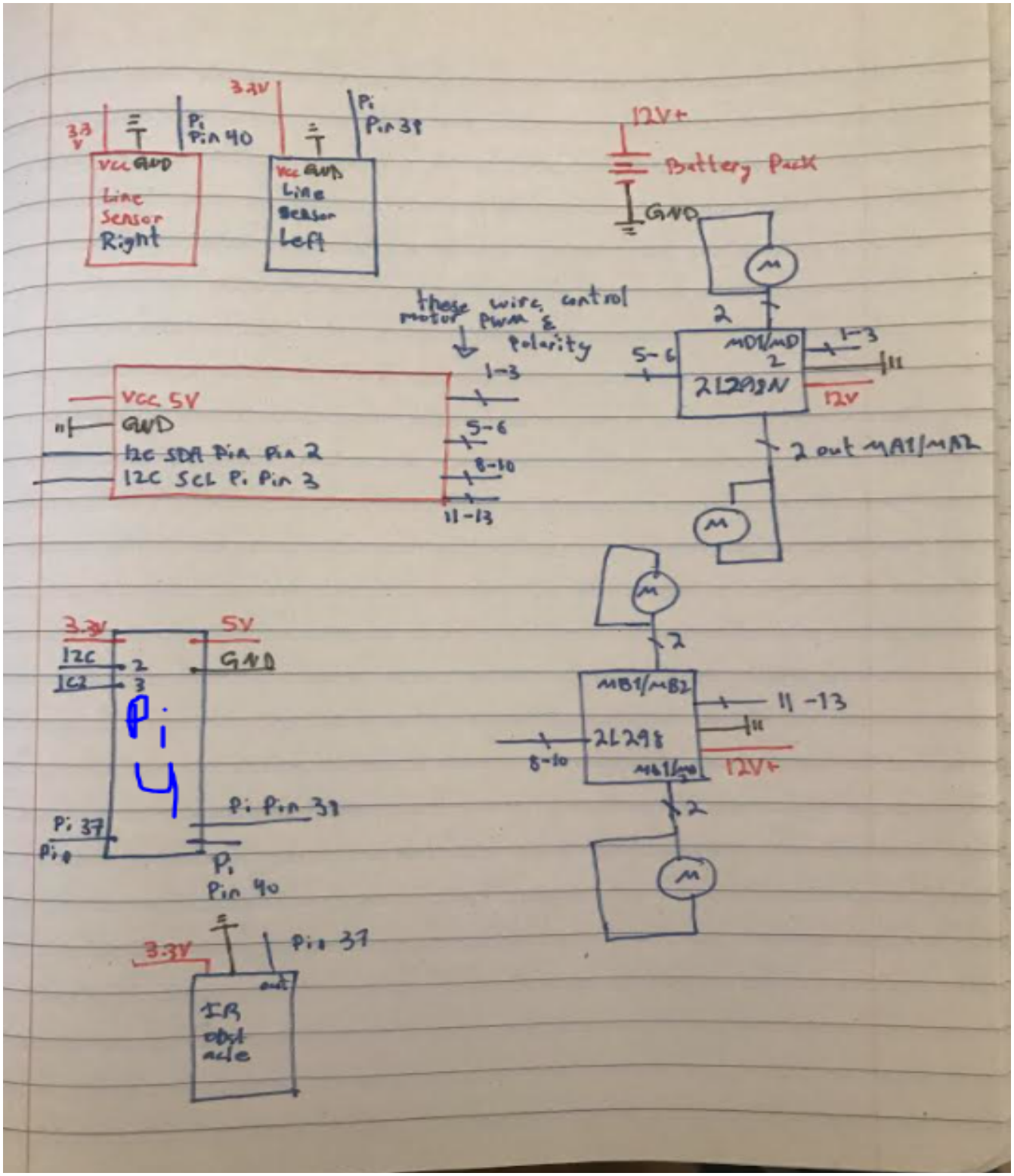
Pin Assignments you used:

We used the bottom pins of the pi to correspond to GPIO physical pins 40,38, 36 37,35. These were either enabled as OUTPUT or INPUT based on if they were reading the input or sending a trigger signal. I also used the I2C pins connection for the PCA9685 to control the L298N H-Bridge Motor Driver. Since Each L298N controls two motors with polarity and PWM power, I ended up using 12 Channels on the PCA9685 to control 4 motors independently. These pins corresponded to physical pins 3 and 5 for I2C connections. The GPIO pins on the Pi 4 are shown below.



Hardware Diagram:

For the Hardware diagram if i were to include all the wires it would be over run so I simply turned the pins and voltage sources into nodes that way it is much easier to connect and the wires are a nuisance that get in the way and make the picture more convoluted. I have included the components pins and what nodes they connect to on the Pi.


3.3V
3.3 V
Pi Pin 40
Vcc GND
Line Sensor Right
3.3V
Pi Pin 38
Vcc GND
Line Sensor Left
12V+
Battery Pack
GND
these wire control motor PWM & polarity
1-3
5-6
8-10
11-13
VCC 5V
GND
I2C SDA Pi Pin 2
I2C SCL Pi Pin 3
2
MOA/MOB
1-3
5-6
2L293N
11
12V
2 out MA1/MA2
3.3V
5V
I2C 2
I2C 3
GND
Pi
Pi 37
Pi Pin 38
Pi Pin 40
3.3V
Pi 37
IR obstacle
out
MB1/MB2
11-13
2L298
8-10
MA1/MA2
12V+

What worked well:

What worked well for us was distributing the weight along the vehicle evenly. We figured that the biggest challenge we'd face would be balancing the car so it could move in a uniform pattern. After balancing the car however, we came across another challenge that we will discuss in the issues section. Another part of the build that worked well for us was cleaning up the wiring. This is really important because if we wanted to produce an optimal output, we would need to keep everything organized. Organizing the wiring really helped us in the general picture of things. The additional purchase of Mecanum omnidirectional wheels also really helped us. Instead of using the wheels provided with the manager kit, we chose to go with Mecanum wheels which provided us with easier motion and more flexible range. Also, by using threads, we were able to split everything within threads so we could get constant data from sensors and run the motor simultaneously without having to deal with writing unnecessary loops.

What were issues:

One of our biggest problems early in the development of the car was to get the car to go straight. To put it into perspective, we made sure that the car was balanced so the wheels were not unevenly working. Therefore, we moved everything around the car including the wiring and their position to achieve an optimal weight where we thought everything was balanced. However, later on, after a couple brainstorming sessions with the group, we decided, instead of controlling the weight distribution across the chassis of the car, we would control the power and torque each wheel produces. Since this seemed like the main control of direction and its linear path. After playing around with the code, we decided to change the power output wheel by wheel until we achieved a result in which the car was going in multiple directions in a linear path instead of deviating. Another important mention for hardships encountered during this project was the collaboration. We were able to schedule and conduct meetings effectively, however since we are all operating in an online remote environment, we were not able to efficiently collaborate due to the physical nature of the course having to build a car with different moving components. Most of the credit behind building the car goes to our hardware manager Feras for handling everything so well in this remote environment.

Final Discussion:

All in all this was an amazing class and the project was challenging yet amazing and fun to play around with and debug. Me and my team had a wonderful time taking this course and enjoyed the frustration and the long nights staying up debugging what went wrong with the car. This was a fun class and I enjoyed every second of it as a hardware manager. Thank you Prof. Bierman.