TERM PROJECT

6/16/2023

TO:	Charlie Refvem, Cal Poly Mechanical Engineering Department
FROM:	ME 507- mecha05
	Tommy Xu
	Ryan Ghosh
SUBJECT:	Term Project

Mechanical Design

Component Selection

Two of the robot's four drive wheels are each driven by a gear motor. Two additional gear motors are used for the intake, which brings balls into the basket, and for the ball wheel, which lifts balls to the color sensor in a single file. We have two servos for directing the path each ball takes after leaving the color sensor: the ball can go either to the flywheel shooter, out the back of the robot to fall into a corral, or back into the basket.

We used an ambient Light Sensor for line detection and to avoid driving out of the arena. There is a breakout board for this sensor that gives a signal line as well as Vdd and GND. This is so that we can mount the sensor outside of the board where it can see the line. We used the VEML6040 Color Sensor for determining the color of each ball. We planned to use a hall sensor to detect when the robot is near the magnet, but we did not have time to implement this.

The motors and electronics are powered by a 3S LiPo battery. The driving motors, intake motor, and ball wheel motor use 12V, and the servos use 5V. The flywheel motor is borrowed from a nerf gun. We planned on widening the gap between the flywheel to allow it to shoot ping pong balls. Those motors need more than 9V supply, so using 12V supply should be good.

Robot Design

Our design consists mainly of a large basket filling up most of the 250mm x 250mm allowable footprint. The drive wheels, electronics, and battery sit underneath the basket, including the light and hall sensors. A pair of intake rollers, driven by a gear motor, bring balls from the floor up into the basket. A ball wheel at the back of the basket brings balls onto a shelf in a single file. Once a ball reaches the shelf, it passes through the color sensor and then to the first servo. This servo either knocks the ball back into the basket or allows it to continue rolling along the shelf. If the ball continues along the shelf, it reaches the second servo, which directs the ball either into the shooter or out the back of the robot to drop down into a corral or onto the floor of the arena.

The intention of the robot design was that during a match, the robot would first drive around picking up balls. The ball wheel would be turning while it does this, so that balls are constantly being moved through the color sensor. Balls that match our designated color would be knocked back into the basket to be deposited later. All other balls would be dropped out the back of the robot back onto the arena. Once the robot has picked up enough balls of our designated color and filtered out all other balls, it would pass the remaining balls in the basket either through the shooter to put them into the cylinder in the center of the arena, or out the back of the robot to drop them into our corral. Then the robot would start picking up balls again and the cycle would repeat.

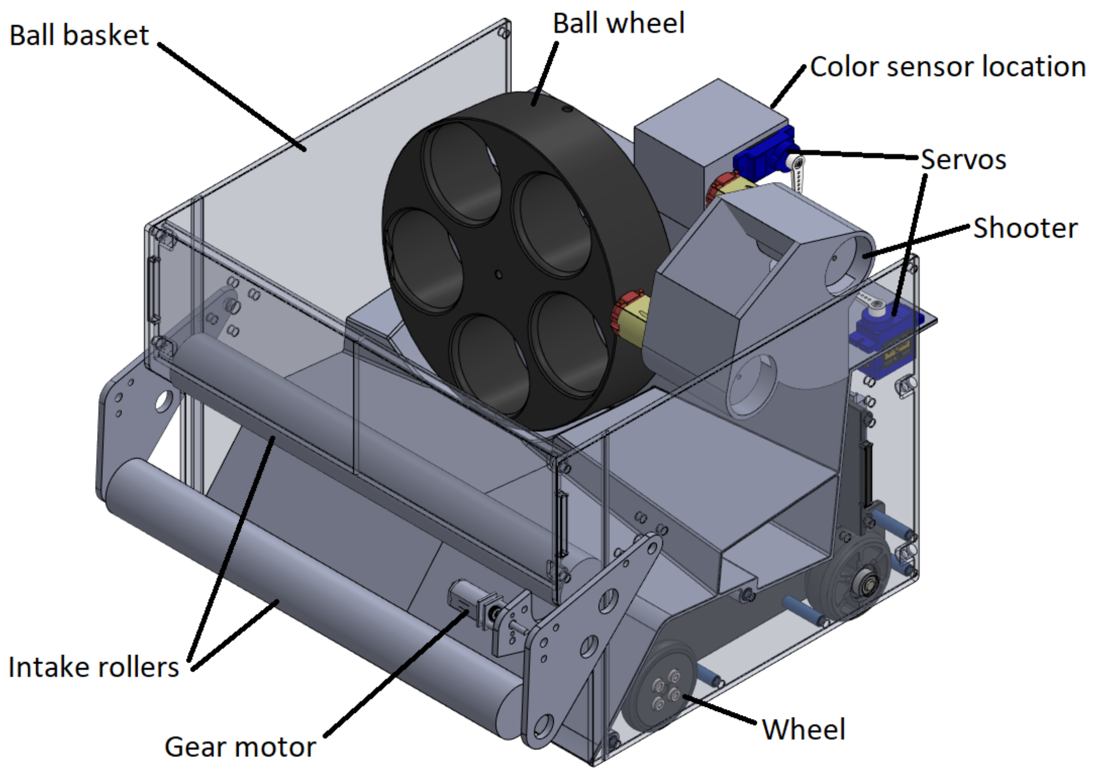


Figure 1: CAD Model of the Robot.

Electronics Design

Wiring and wireless communication

The ambient light sensor is ADC and comes in a breakout board separate from the PCB. This is so that the board does not have to be placed where it is facing the floor to sense the line. The Color sensor will be I2C. Only 1 I2C “channel” needs to be activated. The Hall Sensor is GPIO.

We plan on wirelessly communicating with the robot with a Raspberry Pi. The motivation is just because we own a raspberry pi with Wi-Fi and can start testing SSH right away. It is important because all of our calculations for how to move the robot will be done on a computer running OpenCV in Python. The data sent to the motors will be serial data passed through SSH and into the UART of the Raspberry Pi where it can be received by the microcontroller with wired UART. I double checked that the Pi has alternate functions Rx and Tx.

On the computer end, there will be a phone that will stream its camera feed into the computer, where OpenCV can capture and interpret the data.

Custom PCB

The custom PCB takes power from the 3S LiPo to power the robot. It uses a 5A switching regulator to supply 5V power to the Raspberry Pi and a 3A linear regulator to supply power to the microcontroller and sensors. The PCB has 5 motor outputs for the 2 drive motors, intake motor, ball wheel motor, and flywheel motor. Two of the motor connectors are duplicated in case we wanted to wire 2 motors to the same driver for the intake or flywheel. It also has 2 servo outputs and a hall sensor input. In addition, there are ports for communicating with the ST Link with UART6, communicating with the Raspberry Pi with UART1, and communicating with the color sensor via I2C. An extra header is connected to 9 unused pins on the microcontroller for extra functionality, and there are extra ports for supplying 3.3V, 5V, and 12V power. Lastly, there is a header connected to the motor driver inputs in case our microcontroller didn't work and we had to attach a Blackpill instead.

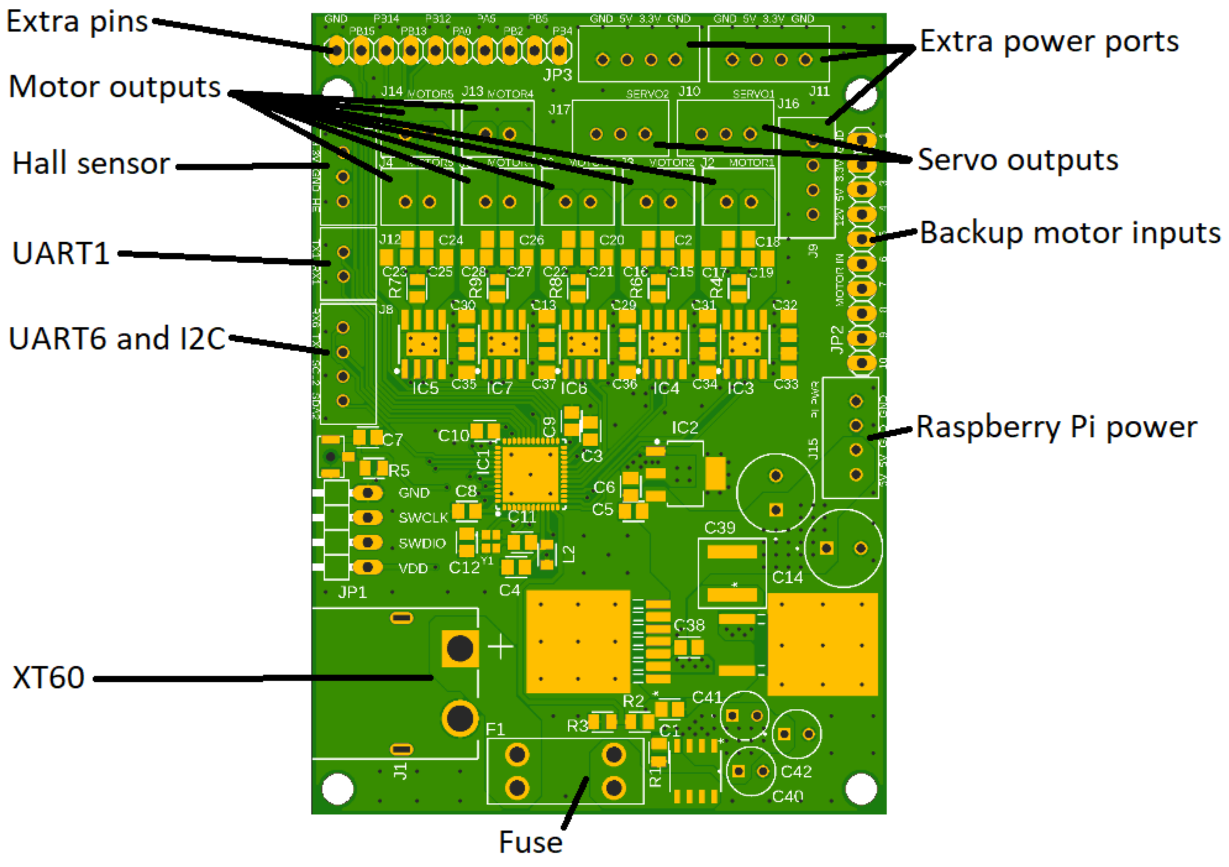
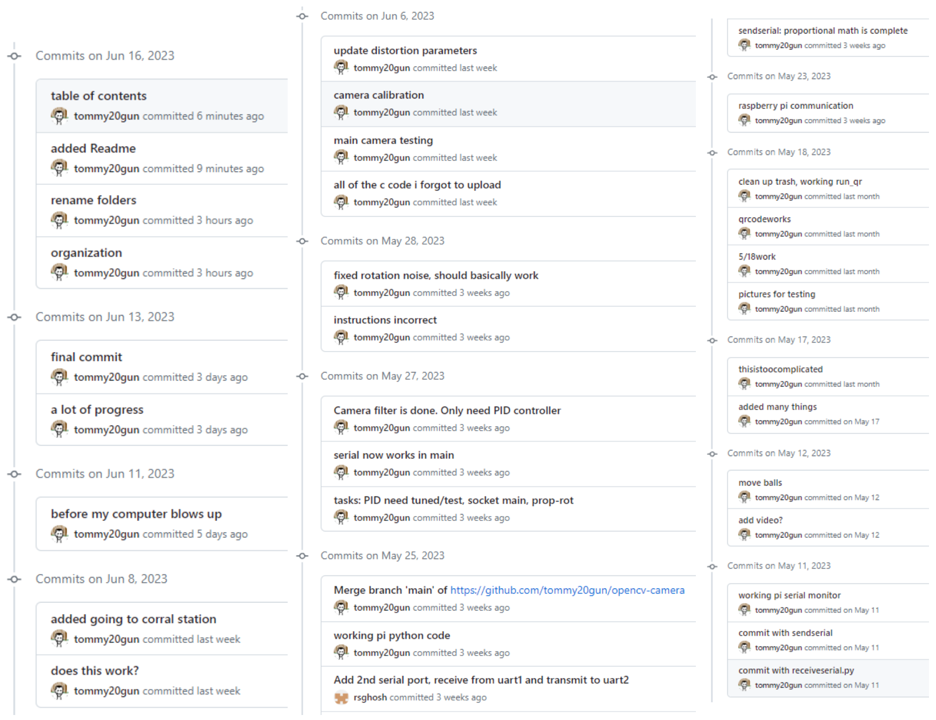


Figure 2: Custom PCB.

Software Implementation

There are two major sections of code for the robot: the C++ code for the microcontroller, and python code for the computer and Raspberry Pi.



Commits on Jun 16, 2023

table of contents
tommy20gun committed 6 minutes ago

added Readme
tommy20gun committed 9 minutes ago

rename folders
tommy20gun committed 3 hours ago

organization
tommy20gun committed 3 hours ago

Commits on Jun 13, 2023

final commit
tommy20gun committed 3 days ago

a lot of progress
tommy20gun committed 3 days ago

Commits on Jun 11, 2023

before my computer blows up
tommy20gun committed 5 days ago

Commits on Jun 8, 2023

added going to corral station
tommy20gun committed last week

does this work?
tommy20gun committed last week

Commits on Jun 6, 2023

update distortion parameters
tommy20gun committed last week

camera calibration
tommy20gun committed last week

main camera testing
tommy20gun committed last week

all of the c code i forgot to upload
tommy20gun committed last week

Commits on May 28, 2023

fixed rotation noise, should basically work
tommy20gun committed 3 weeks ago

instructions incorrect
tommy20gun committed 3 weeks ago

Commits on May 27, 2023

Camera filter is done. Only need PID controller
tommy20gun committed 3 weeks ago

serial now works in main
tommy20gun committed 3 weeks ago

tasks: PID need tuned/test, socket main, prop-rot
tommy20gun committed 3 weeks ago

Commits on May 25, 2023

Merge branch 'main' of https://github.com/tommy20gun/opencv-camera
tommy20gun committed 3 weeks ago

working pi python code
tommy20gun committed 3 weeks ago

Add 2nd serial port, receive from uart1 and transmit to uart2
rsghosh committed 3 weeks ago

sendserial: proportional math is complete
tommy20gun committed 3 weeks ago

Commits on May 23, 2023

raspberry pi communication
tommy20gun committed 3 weeks ago

Commits on May 18, 2023

clean up trash, working run_qr
tommy20gun committed last month

qrcodeworks
tommy20gun committed last month

5/18work
tommy20gun committed last month

pictures for testing
tommy20gun committed last month

Commits on May 17, 2023

thisistoocomplicated
tommy20gun committed last month

added many things
tommy20gun committed on May 17

Commits on May 12, 2023

move balls
tommy20gun committed on May 12

add video?
tommy20gun committed on May 12

Commits on May 11, 2023

working pi serial monitor
tommy20gun committed on May 11

commit with sendserial
tommy20gun committed on May 11

commit with receiveserial.py
tommy20gun committed on May 11

Figure 3: The Commit History.

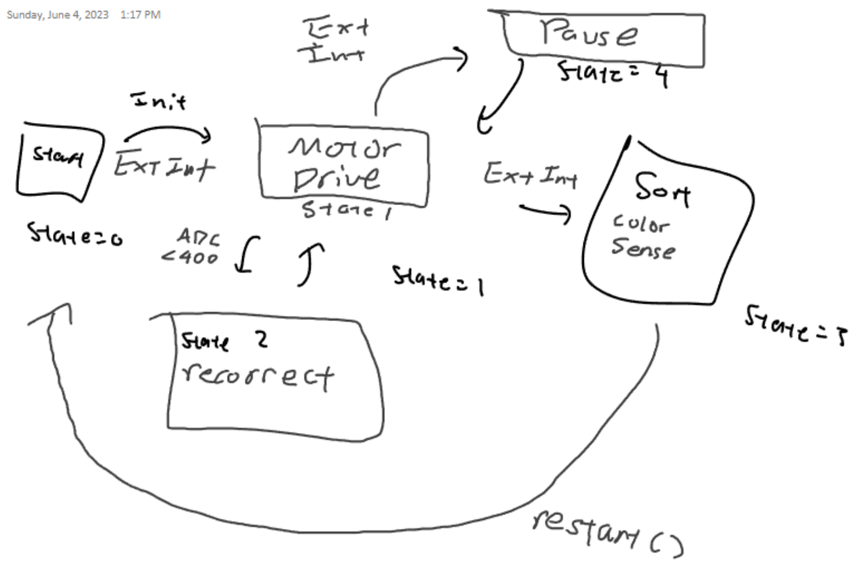


Figure 4: The Finite State Machine for C++.

STM32IDE - C++

The C++ code uses a finite state machine in the main loop with 4 states.

State 0: Start state. In this state, the robot does nothing except wait for a signal to start. A GPIO interrupt triggered by the Raspberry PI (triggered by the user) brings the state machine to state 1.

State 1: The robot looks for a ball and moves towards it if it sees one. Here, the crux of the logic is inside the python scripts that were written for the robot. Here is where the UART1 Receive Interrupt is active. The robot will actively take motor duty cycle data given by the CPython program through UART1 to know how to drive itself.

The only thing insde State 1 in the C++ code is to check the ADC reading of the light sensor. If the light reading falls below a certain threshold, the state machine will move into state 2. More on state 2 in the next section.

Video: https://photos.app.goo.gl/tfheLbLGFPe3WtNf7

State 2: The purpose of state 2 is to correct the robot if it runs out of the arena. The light sensor actually has a good threshold value of 400 units that, when faced downward, determine if the robot is in the arena (where the floor is white) or outside of the arena (where the floor is grey).

This is shown in a video demonstration where when we blast the light sensor with light, the state remains at 1. When the light sensor moves away from the light, the robot will go to state 2 and move backwards.

Video: https://photos.app.goo.gl/UTSYJjyMzUb4Tg8u6

State 3: This is the ball sorting state. Due to time constraints, we were not able to fully implement this portion of the robot. However, the color sensor and servos, as well as any motors function properly. One would only need to attach the ball carrier to make this work.

The color sensor is an I2C device that returns values of RGB based on what it sees. The code initializes all the necessary registers for I2C reading. Then the determineColor() function selects the RGB value with the biggest magnitude and determines the ball's color based on the color with the biggest signal. For example, if blue = 1052, green = 870, red = 213, then the color was determined to be "Blue".

The shortcoming of this algorithm was that it could not detect yellow balls, as yellow was a combination of blue and green. I wrote, but commented out, an algorithm where color values are compared to each other and then matched to a ratio, as all colors are just a ratio of RGB. This algorithm is good in theory but was not tested for accuracy.

Color sensor part 1: https://photos.app.goo.gl/r2p6N2vRuNmkGFyi7

Color sensor part 2: https://photos.app.goo.gl/Zmd4ECojo6uSsa898

State 4: This is the Pause state. The sole purpose of this state is to act as the dead man's switch. The code can be paused at any point in time where the user presses 'Space Bar' with the Python program opened. The computer would send an instruction to the Raspberry Pi that triggers a GPIO interrupt in the MCU to put the robot into its pause state.

CPython

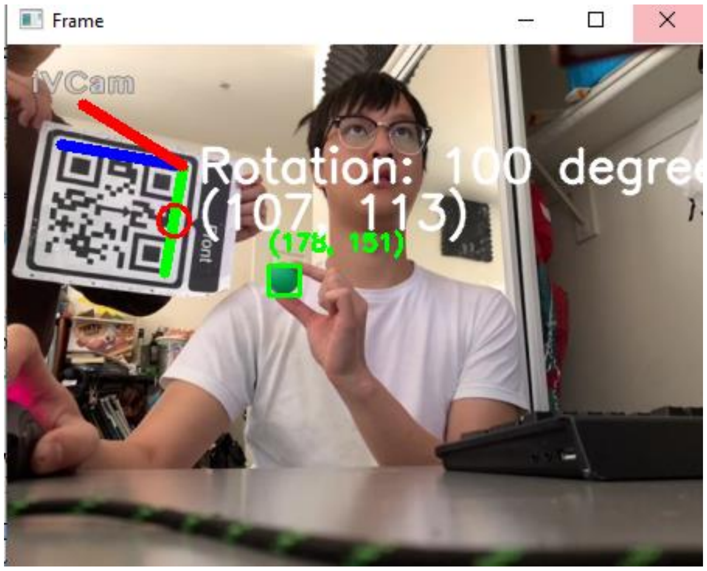
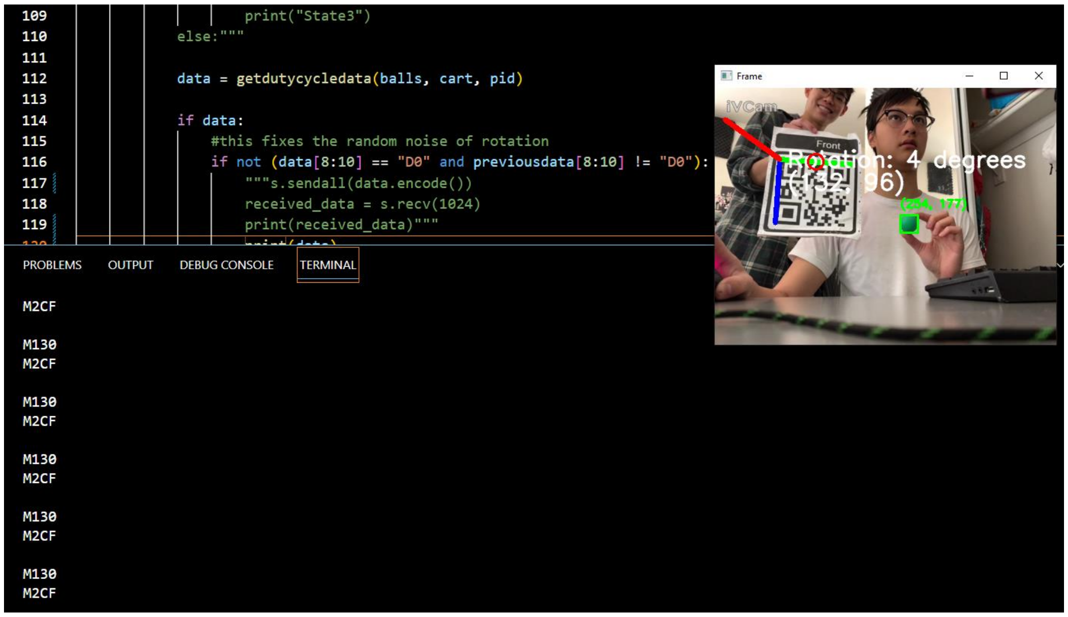


Figure 5: Demonstration of Camera capture

This portion of code took up the most time. However, I will not explain each class as it gets complicated quickly. I will only explain the logic of the Python code.

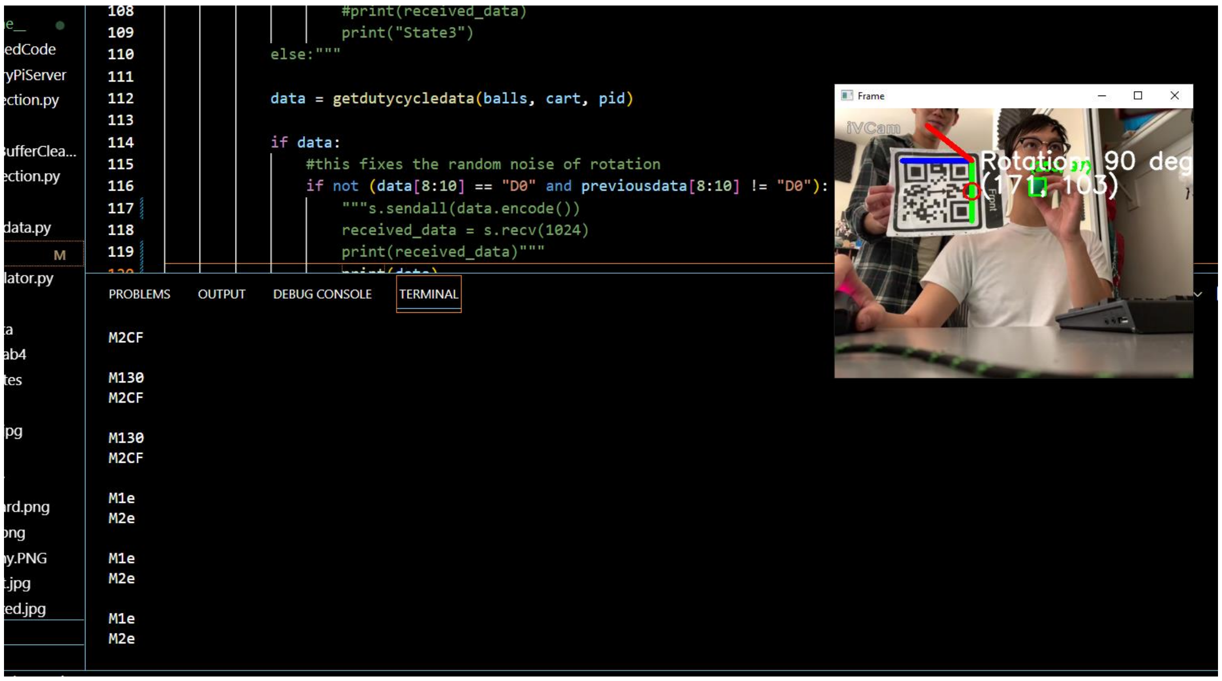


```
            print("State3")
        else:"""

        data = getdutycycledata(balls, cart, pid)

        if data:
            #this fixes the random noise of rotation
            if not (data[8:10] == "D0" and previousdata[8:10] != "D0"):
                """s.sendall(data.encode())
                received_data = s.recv(1024)
                print(received_data)"""
```

```
M2CF

M130
M2CF

M130
M2CF

M130
M2CF

M130
M2CF

M130
M2CF
```

Figure 6: Robot rotation instruction because the robot *is not* facing the ball.

Within Cpython, the OpenCV Library is implemented. The order of logic is as follows. The camera detects Ball and Cart objects and sets the robot to spin in a circle until the front of the robot faces the ball. The code to make the robot spin in circle is M130M2CF.



```
                #print(received_data)
                print("State3")
        else:"""

        data = getdutycycledata(balls, cart, pid)

        if data:
            #this fixes the random noise of rotation
            if not (data[8:10] == "D0" and previousdata[8:10] != "D0"):
                """s.sendall(data.encode())
                received_data = s.recv(1024)
                print(received_data)"""
```

```
M2CF

M130
M2CF

M130
M2CF

M1e
M2e

M1e
M2e

M1e
M2e
```

Figure 7: Robot rotation instruction because the robot *is* facing the ball.

When the robot faces the ball, then it will activate a proportional controller that drives it towards the ball to capture it. Further, if the robot is closer to the ball, the duty cycle will continuously decrease to 0 until the robot touches the ball. Since the ball was so close to the "cart". The motor duty cycle was M1E and M2E. 0x0E is in hexadecimal. If there are no balls, the Python program will send an instruction to the Raspberry Pi to trigger an interrupt for the MCU to move the state machine into state 3.

The Camera is wirelessly connected to the CPython program through the IPcamera app on the phone. Distortion matrices were used to calibrate our camera.

CPython is active at all times and sends motor duty cycle to the robot at all times, but the MCU makes it so that duty cycle data can control the robot ONLY in state 1. The program sends data to a Raspberry Pi hosting a server over iPV4 Wireless communication, which is a descriptive way of saying a Wifi Hotspot. Python's socket class was used to allow this to function.

Raspberry Pi Python

The Raspberry Pi hosts a server on a Wifi Hotspot that allows any users to connect to it. Thus the camera, computer, and Raspberry Pi needed to be on the same network for this to work. The Pi opens a socket and the computer sends data over the socket. Then, the Raspberry Pi determines whether the data sent was a motor duty cycle, an instruction to pause, or an instruction to trigger state3. Once it deciphers this, it then sends the correct instruction to the MCU.

If it is a motor duty cycle, the Raspberry Pi encodes the string into UTF-8 format and sends the byte data over wired serial UART1 into the MCU. An interrupt is triggered on the MCU to read this data. IF the Pi receives a Pause instruction, a GPIO pin is toggled that triggers a different interrupt on the MCU to pause. Finally, if the Pi receives a State 3 instruction, another GPIO pin is toggled to trigger an interrupt on the MCU to move the program into state 3.

The Raspberry Pi is the device that worked the most flawlessly. We were very lucky to not have to deal with RC or Bluetooth.

Challenges and Workarounds

Our PCB design had a few mistakes. We accidentally ordered a diode that did not match what we used in Fusion 360, so we had to desolder the diode from the board and attach it with wires so the legs and tab would be connected to the correct pads. We also accidentally connected the tab of the linear regulator to ground in the PCB when the tab should have been connected to 3.3V, so we had to desolder the regulator and solder it back on at an angle so the tab would not be touching the board. Lastly, we forgot to add enough bulk capacitance, so we soldered an extra capacitor inline with the power plug. After these fixes, our board seemed to work as intended.

The mechanical design for our robot might have been too ambitious for the time constraints we had. We completed the frame and drive base for our robot so we could mount the electronics and the robot was able to drive around. However, we did not have time to finish the intake or mount the color sensor and servos. We were still able to test that the servos could respond to the color sensor readings.

Within software implementation, the Python program with OpenCV caused the most headaches. Seeing that in another group, one person spent their entire time working on the Python program for the class, we realized that both completing a Python program and building the robot was too much to ask for in a pair. Therefore, after a certain point in time, we abandoned python to work on the actual robot. Using a good camera was challenging because the IP camera had lag, but the iPhone app camera, where it installs the camera as a local device on the computer, had too small of resolution. The resolution was 640*320. This was impossible for us to detect the QR code from far away.

Another issue we ran into was detecting the QR code at all. I initially used another method of determine the QR code, but that took too much processing power and cause lag. I tried my best over a week's time to optimize this and apply filters to it, but there was no success. We ended up using the regular QR code detection class within OpenCV's library.

Lastly, we ran into "integration hell". We needed to tune the duty cycle, and many parameters of the Python program with the mechanical robot once everything was built. Since Python was written a month before the robot was built, things that I had envisioned for Python to be able to control the robot did not come to fruition. We ran out of time adjusting Python parameters to the mechanics of the robot. It is unfortunate because it seemed like a job that could be complete with just a few more days.

Overall, we were still happy with the progress of our robot, its ability to spin around in circles for the class and drive forward really fast.