دانشگاه شهید باهنر کرمان

Report for the Computer Architecture Course

Related Instructors:

Dr. Vahid Jamshidi
Dr. Araqi Pour

Group Members:

Amir Ali Mirzaei
Mahdi Borhani
Esmaeil Bagheri
Babak Fathi

Fall 2023

Project Objective:

Building a footballer robot that can be controlled by a game controller, pick up and carry the ball, and shoot it from a few centimeters distance from the goal. It should also be able to distinguish between the goal and the ball.

Required Hardware:

1.Metal chassis and wheel We used 7 chassis pieces (6 bent pieces and 1 flat piece).

2. Wheels 4 grooved wheels with traction wrapped around them. Two front wheels are free-spinning, and two rear wheels are connected to the motor.

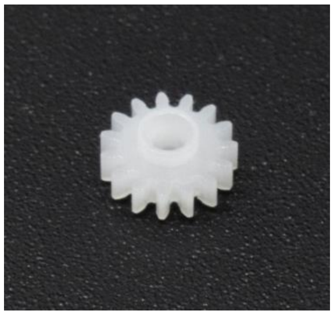
3. Geared motor Two geared motors that we used 6-volt motors with 1/5 amperage.

4. **Screw and Spacer This item allows electronic components to be placed at a distance from the metal chassis. If you use a plastic chassis, you don't need spacers.**

5. **Raspberry Pi 4 board**

6. **Servo motor To pick up the ball, we need a motor that can rotate at a constant speed through an angle.**

7. **Shooter We used several gears, a special case, and a spring for shooting.**

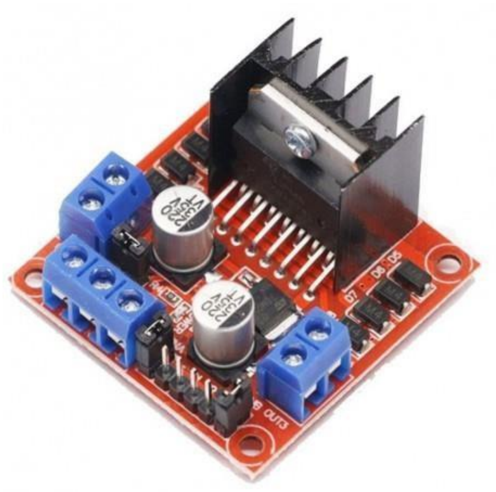


8. **Raspberry Pi case It's better to use a case to prevent short-circuiting the Raspberry Pi.**

9. **PlayStation controller For controlling the robot**

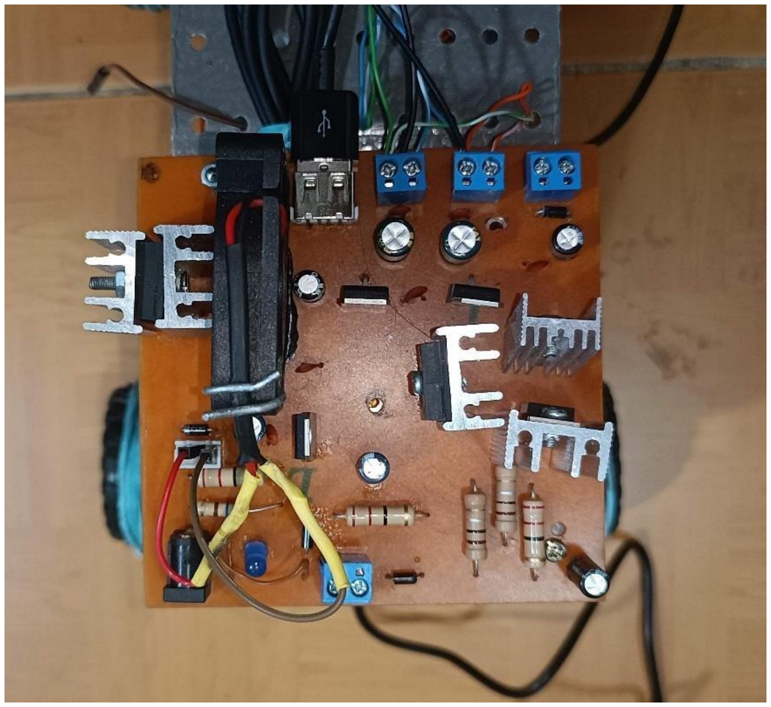
10. Wires Jumper wires: male-to-male, female-to-female, and male-to-female

11. L298 driver We need two L298 modules to control two motors of the robot and also the shooter.



12. Adapter We need a 12V and 3A adapter because our power supply input is this amount.

13. Power supply (PCB)



14. Camera We use two mobile phones, one acts as a camera and sends the front view of the robot, while the other receives and displays the image, allowing the group to control the robot remotely.

15. SD card One 32 GB SD card

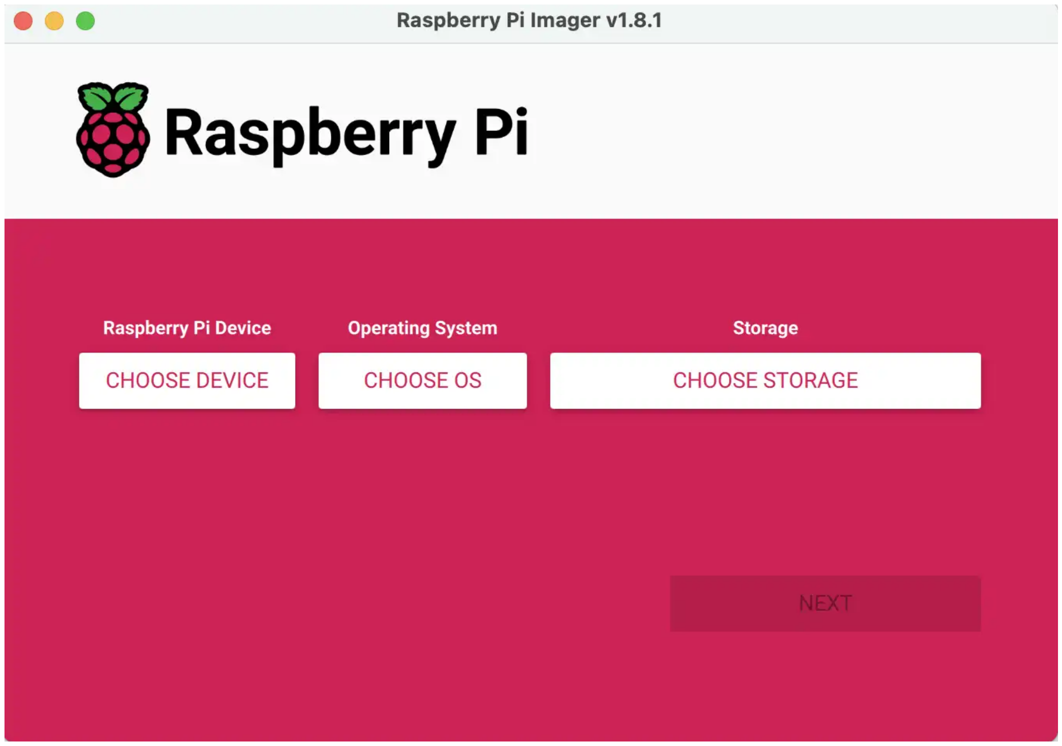
16. Raspberry Pi operating system (Raspbian) Downloadable and installable from the official Raspberry Pi website.

17. VNC Viewer software

18. VNC Server software

19. One ball

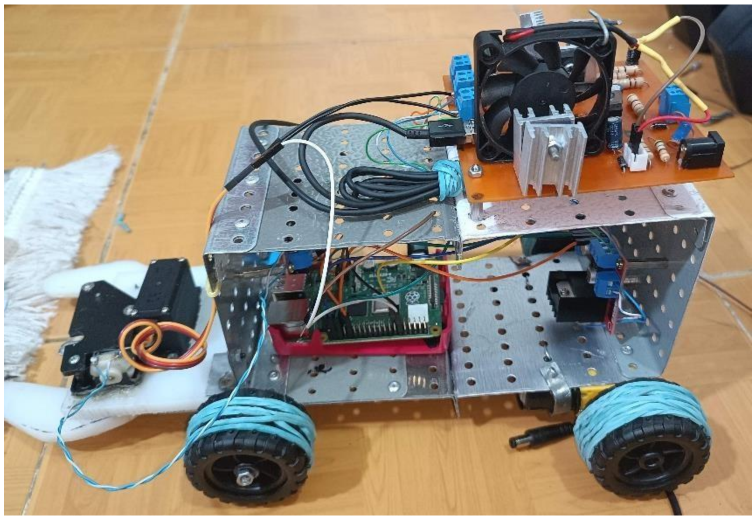
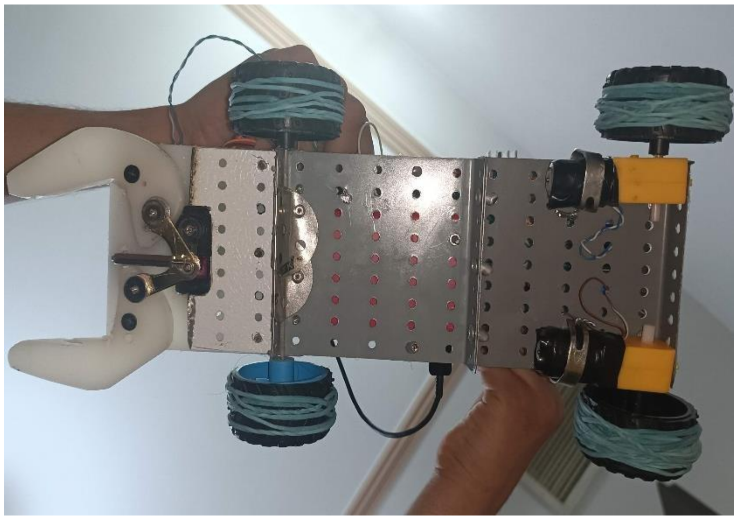
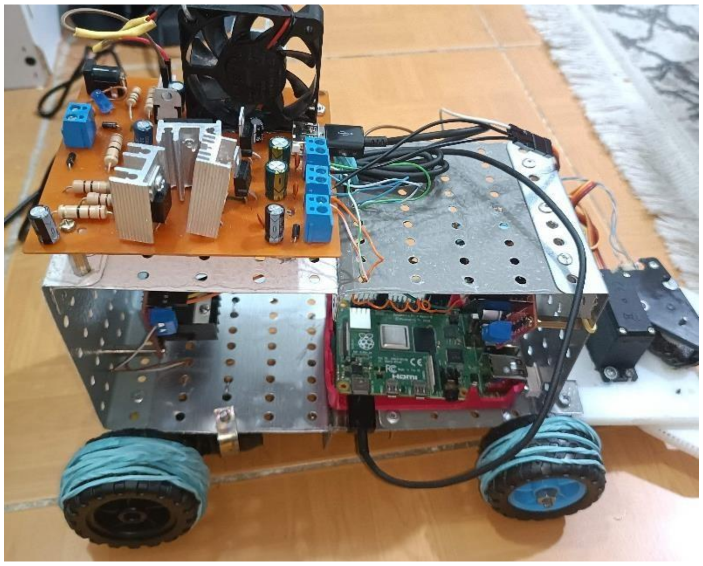
20. Raspberry Pi Imager Using this software, you can easily install the downloaded operating system on the SD card and set it up.



Robot Report Description

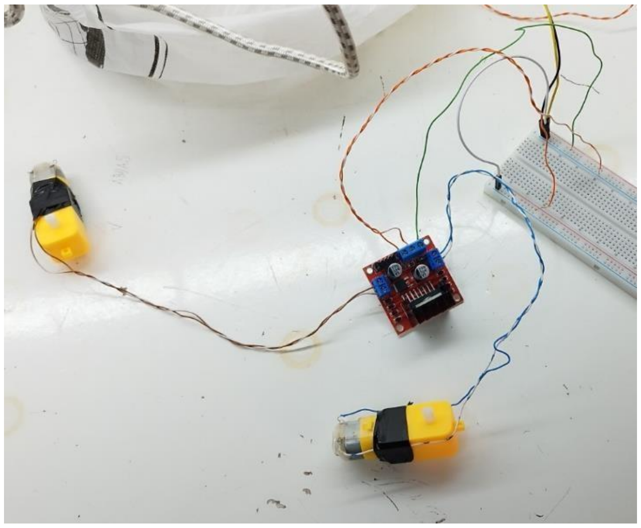
Robot Body Assembly:

First, we assemble the metal chassis pieces together.

Robot Motion System:

For movement, the robot needs wheels, motors, motor drivers, power supply, and a Raspberry Pi board. We connect two geared motors to the robot body using special brackets, then attach two free-spinning wheels to the front.



The wires from both motors are connected to the L298 to determine the direction of motor rotation.

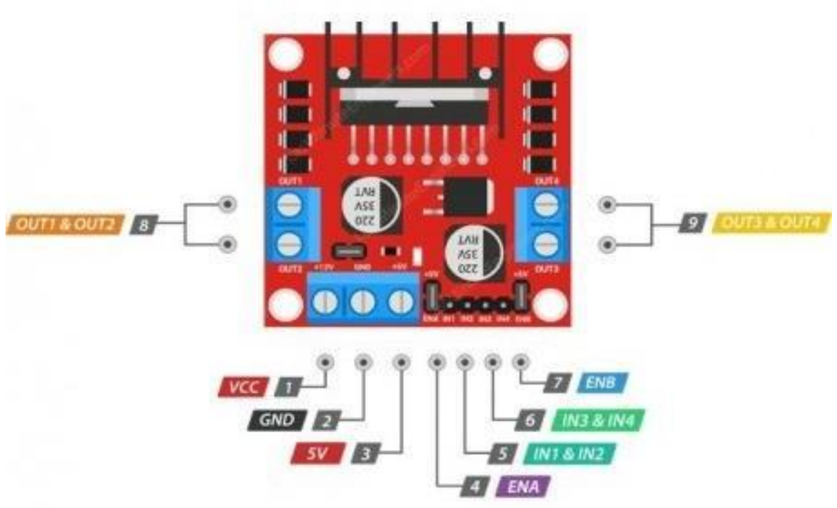
Note: After soldering the wires to the motors, first use electrical tape to prevent the wires from contacting the body, then wrap electrical tape around the motors.

Motor Driver:

The motor driver is an electronic component that acts as an intermediary between the microcontroller and the wheels.

The Raspberry Pi can directly control the direction and speed of the wheels through the control pins, but it has limitations in providing the required current for the motors. If the current passing through the Raspberry Pi is too high, there's a risk of it burning out.

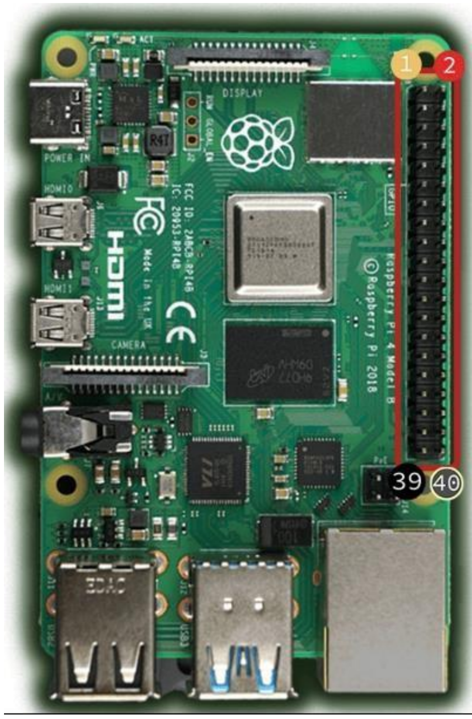
Since we are using the L298 module, which has 3 input pins and 4 output pins (these are for energy, of course, as it also has 6 control pins that we'll discuss later).
 the first input, Vcc, provides the required voltage for the motors. The second is GND, which connects directly to the PCB. The third input is 5V, which also connects directly to the PCB.

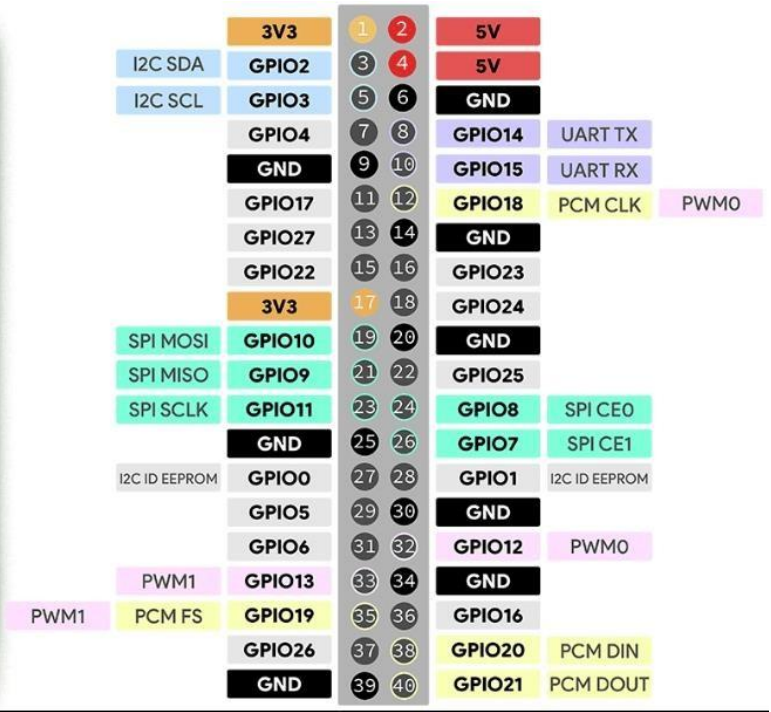


Now we need to set the motors in motion, which our control system, the Raspberry Pi 4, will manage.

Control System (Raspberry Pi 4):

We connect the power supply to the Raspberry Pi's Type C power input. The Raspberry Pi board has 40 pins (Pin), which are numbered by the software. We have two methods, PCM and Borad, of which we used PCM.

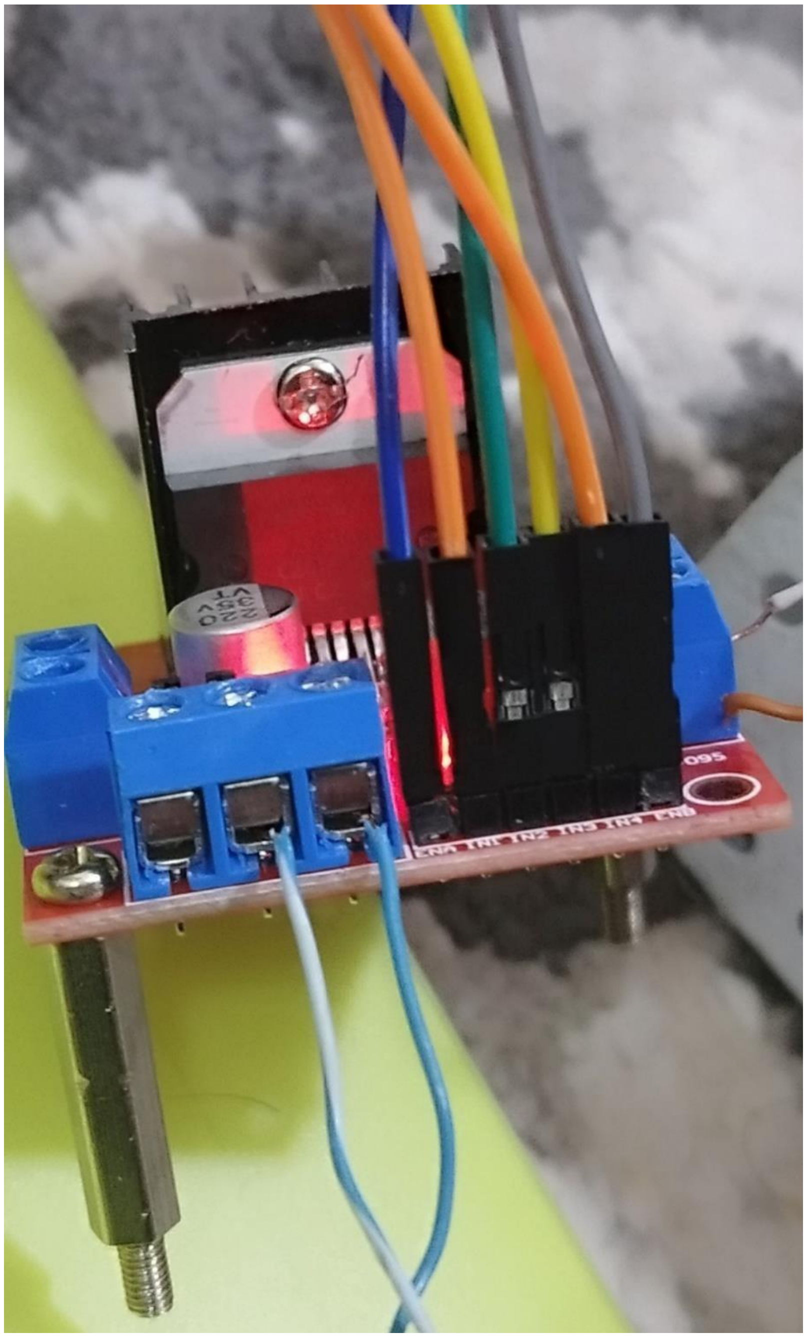




		3V3	1	2	5V		
	I2C SDA	GPIO2	3	4	5V		
	I2C SCL	GPIO3	5	6	GND		
		GPIO4	7	8	GPIO14	UART TX	
		GND	9	10	GPIO15	UART RX	
		GPIO17	11	12	GPIO18	PCM CLK	PWM0
		GPIO27	13	14	GND		
		GPIO22	15	16	GPIO23		
		3V3	17	18	GPIO24		
	SPI MOSI	GPIO10	19	20	GND		
	SPI MISO	GPIO9	21	22	GPIO25		
	SPI SCLK	GPIO11	23	24	GPIO8	SPI CE0	
		GND	25	26	GPIO7	SPI CE1	
	I2C ID EEPROM	GPIO0	27	28	GPIO1	I2C ID EEPROM	
		GPIO5	29	30	GND		
		GPIO6	31	32	GPIO12	PWM0	
	PWM1	GPIO13	33	34	GND		
PWM1	PCM FS	GPIO19	35	36	GPIO16		
		GPIO26	37	38	GPIO20	PCM DIN	
		GND	39	40	GPIO21	PCM DOUT	

Since we also need to control the motor speed, we need 6 GPIO pins on the Raspberry Pi. For the catcher (Catcher), we need one PWM GPIO pin, and for the shooter (Shooter), we need one regular GPIO pin.

We need to connect the 6 ports that are PWM to ENA and ENB, and we should first remove the jumpers on them and connect 4 other control pins.

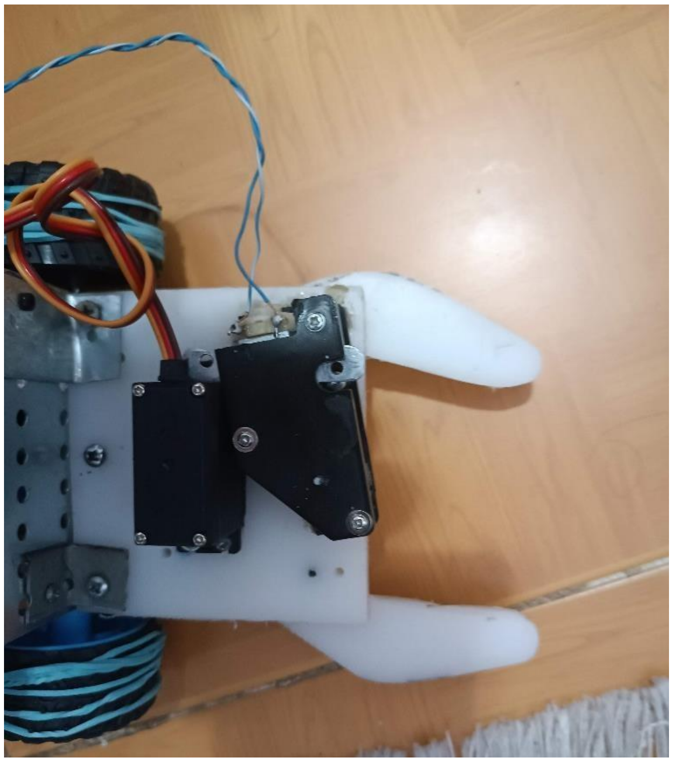


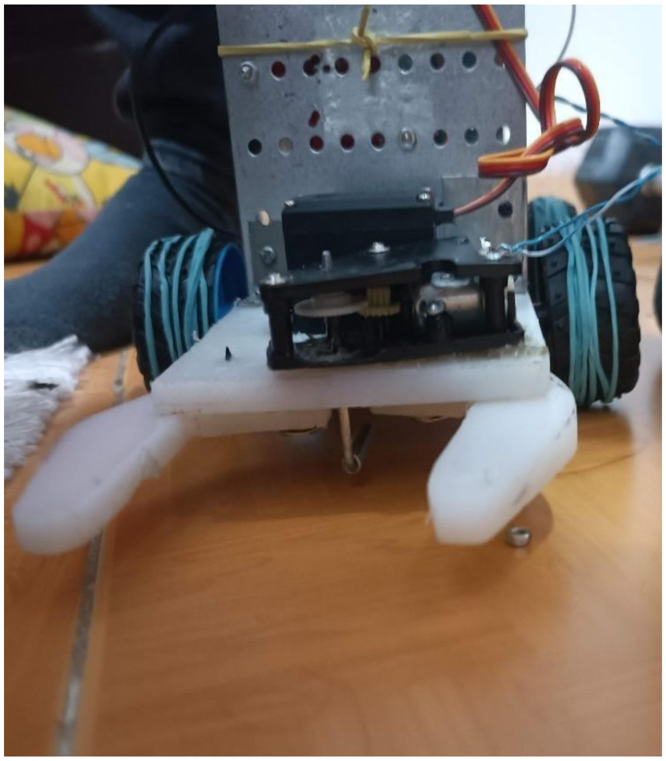
The robot's motion system works as follows: if the GPIOs connected to Raspberry Pi, IN2 and IN3 are activated, the robot moves forward, and if IN1 and IN4 are activated, the robot moves backward.

For rotation to the left and right, (IN1, IN3) and (IN2, IN4) respectively should be activated. Activation means becoming True, and deactivation means becoming False for the GPIO pin on the Raspberry Pi board.

For the catcher (Cacher), we also have a PWM signal that sends a specific angle of movement if activated, catches the ball, and releases the ball at another angle.

For the shooter (Shooter), we only have a regular GPIO signal, and the speed of movement is controlled by gears. We don't need PWM, and whenever the GPIO becomes 1, this motor starts to move.





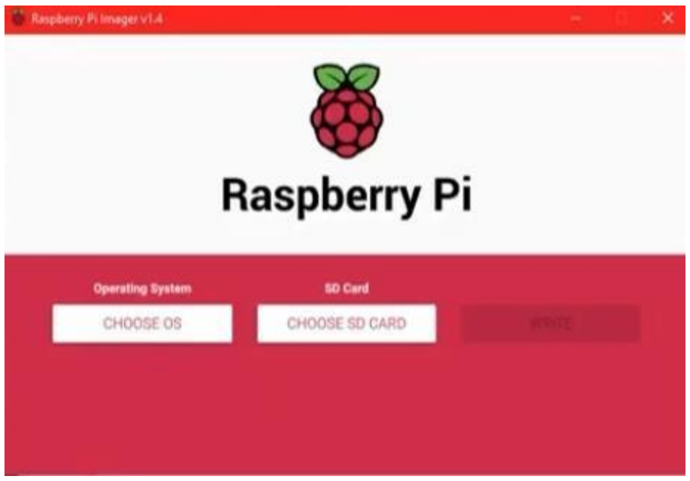
Explanation of Robot Software and Code

Raspberry Pi Setup: First, to set up the Raspberry Pi, you need to install the operating system on the SD card. For this, insert your SD card into an SD card reader and connect it to your computer.

Then, open the Raspberry Pi Imager program that was mentioned earlier.

In this section, choose the operating system you've downloaded for installation.

After selecting the memory card in the software, choose the "write" option. This operation may take a few minutes.



After that, connect the memory card to your Raspberry Pi. After this, connect peripheral devices such as keyboard, mouse, and monitor to your Raspberry Pi and finally turn it on.

After turning it on, you should select the country and time zone.

Enter the username and password on the created page. Then connect your Raspberry Pi to your desired internet. Finally, restart your Raspberry Pi for the settings to take effect.

Setting up VNC VIEWER and VNC SERVER

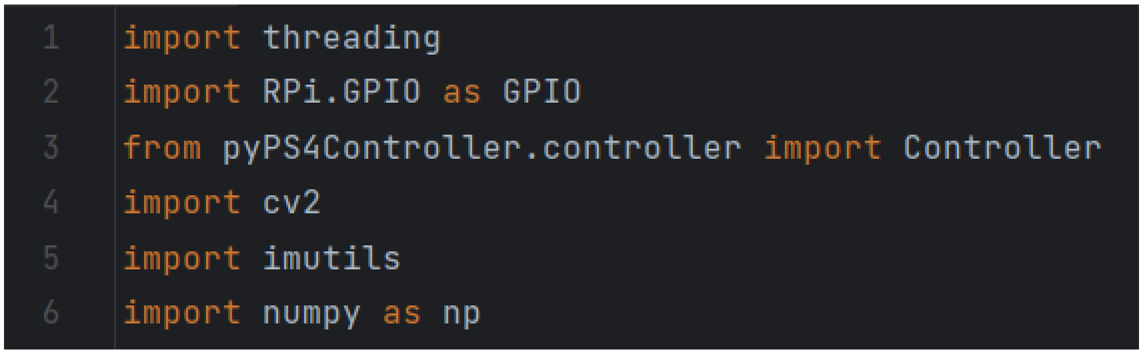
First, install VNC Viewer on the Raspberry Pi using the following command:
"sudo apt-get install real-vnc-server“

Then, run "sudo apt-get update" to update, and then activate VNC through the following command:
"sudo raspi-config“

Then install VNC Viewer on your computer.
In the next step, enter the IP shown on the Raspberry Pi in VNC Viewer, and then enter your Raspberry Pi username and password to access your system through your own system.

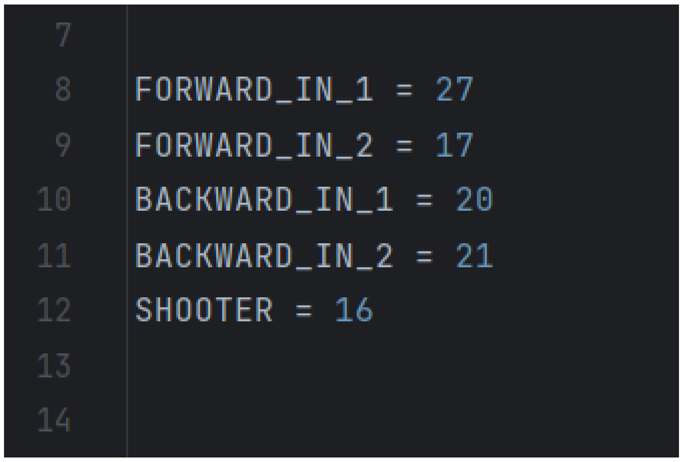
Robot Codes:

As shown in the image below, we need the following libraries to control the robot with a PlayStation controller, access the camera, and the Raspberry Pi pins:



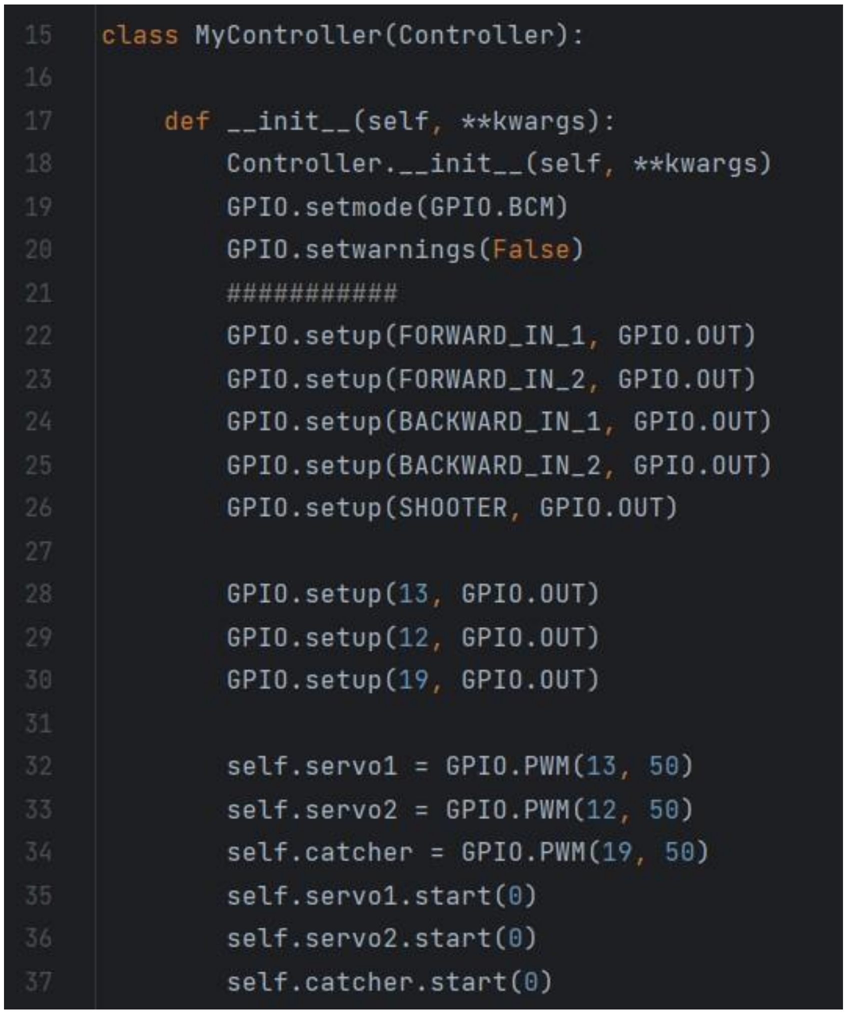
```
import threading
import RPi.GPIO as GPIO
from pyPS4Controller.controller import Controller
import cv2
import imutils
import numpy as np
```

The variables below show the pins we need to control the robot that have been defined:



```

FORWARD_IN_1 = 27
FORWARD_IN_2 = 17
BACKWARD_IN_1 = 20
BACKWARD_IN_2 = 21
SHOOTER = 16


```

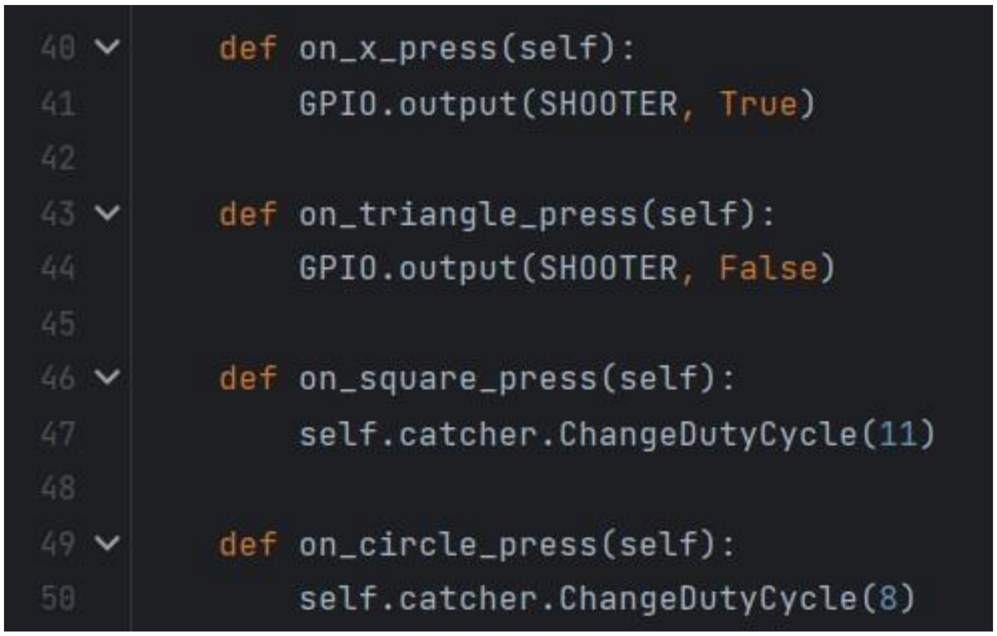
In the MyController class, we set up all the necessary pins. We set the GPIO mode to BCM and activated all pre-defined GPIO pins along with the servo and catcher.



```python
class MyController(Controller):

    def __init__(self, **kwargs):
        Controller.__init__(self, **kwargs)
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        ###########
        GPIO.setup(FORWARD_IN_1, GPIO.OUT)
        GPIO.setup(FORWARD_IN_2, GPIO.OUT)
        GPIO.setup(BACKWARD_IN_1, GPIO.OUT)
        GPIO.setup(BACKWARD_IN_2, GPIO.OUT)
        GPIO.setup(SHOOTER, GPIO.OUT)

        GPIO.setup(13, GPIO.OUT)
        GPIO.setup(12, GPIO.OUT)
        GPIO.setup(19, GPIO.OUT)

        self.servo1 = GPIO.PWM(13, 50)
        self.servo2 = GPIO.PWM(12, 50)
        self.catcher = GPIO.PWM(19, 50)
        self.servo1.start(0)
        self.servo2.start(0)
        self.catcher.start(0)
```

We defined the function of each PlayStation controller button as follows:

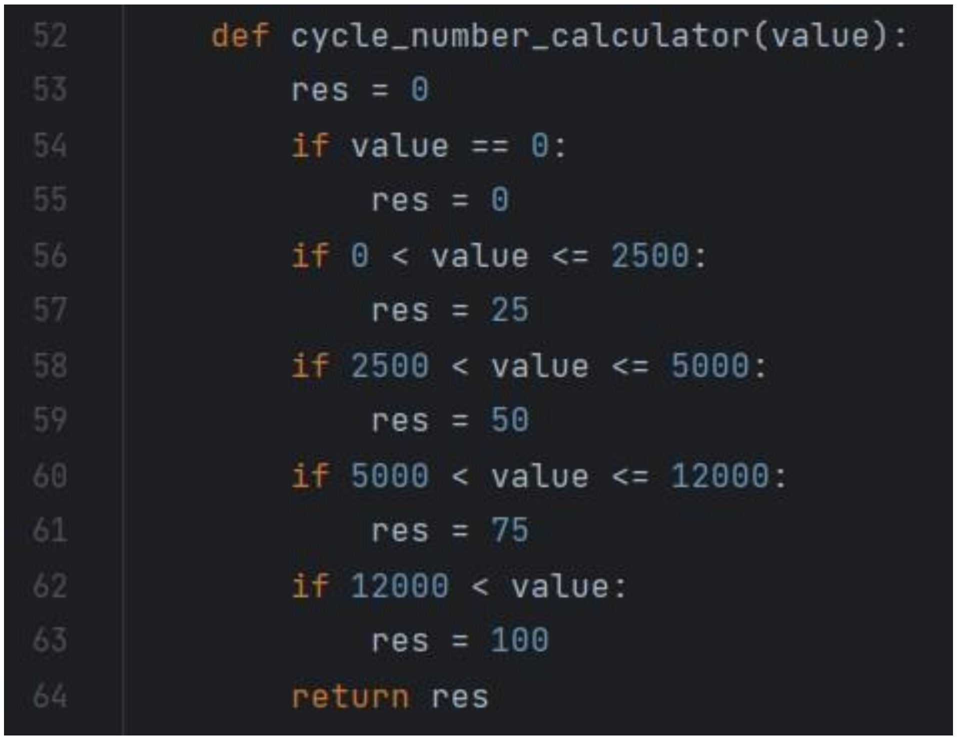
Cross (X): Activate the shooter
Triangle (△): Deactivate the shooter
Square (□): Close the catcher Circle
(○): Open the catcher



```python
    def on_x_press(self):
        GPIO.output(SHOOTER, True)

    def on_triangle_press(self):
        GPIO.output(SHOOTER, False)

    def on_square_press(self):
        self.catcher.ChangeDutyCycle(11)

    def on_circle_press(self):
        self.catcher.ChangeDutyCycle(8)
```

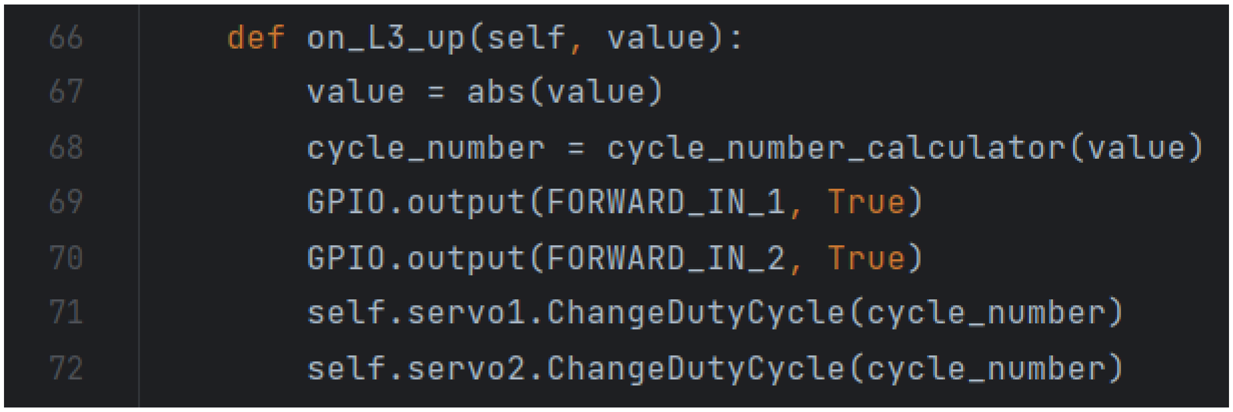
Robot Movement:

We've considered the left analog stick or L3 for the robot's movement. Depending on how much it's moved from its neutral position in any direction, the robot's speed changes. The more it moves away from the neutral position and gets further, the higher the movement speed becomes. In the function shown below, we've determined this speed, which is divided into four states:



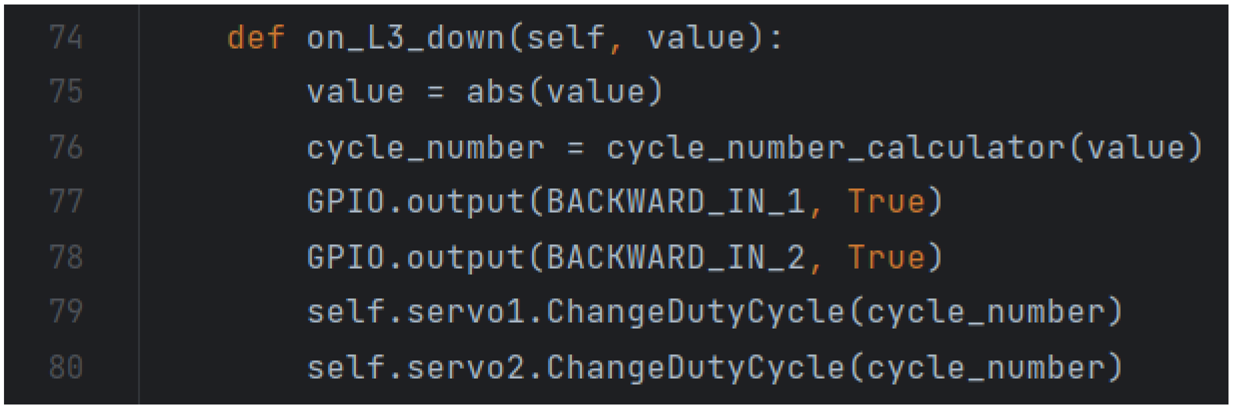
```python
def cycle_number_calculator(value):
    res = 0
    if value == 0:
        res = 0
    if 0 < value <= 2500:
        res = 25
    if 2500 < value <= 5000:
        res = 50
    if 5000 < value <= 12000:
        res = 75
    if 12000 < value:
        res = 100
    return res
```

Forward movement:



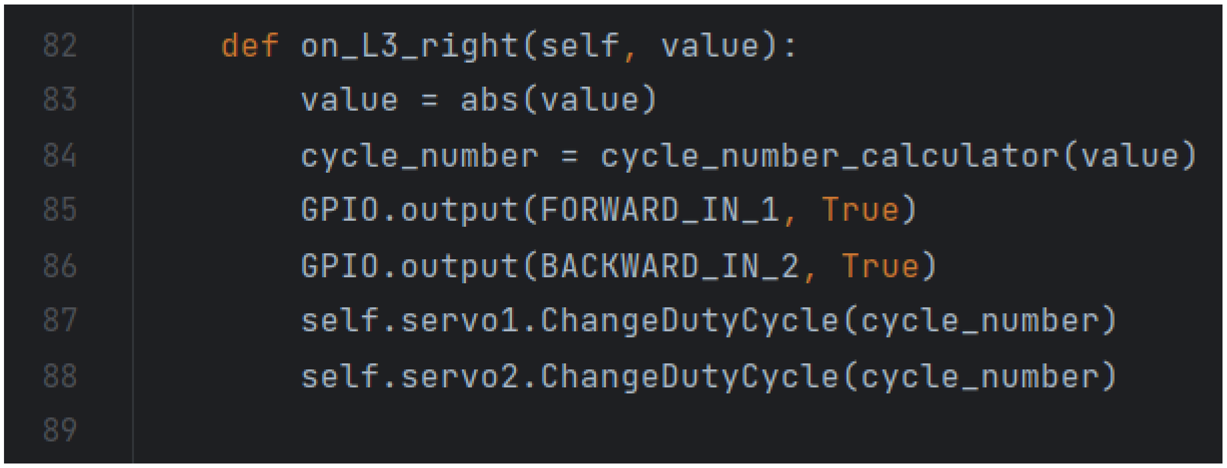
```python
    def on_L3_up(self, value):
        value = abs(value)
        cycle_number = cycle_number_calculator(value)
        GPIO.output(FORWARD_IN_1, True)
        GPIO.output(FORWARD_IN_2, True)
        self.servo1.ChangeDutyCycle(cycle_number)
        self.servo2.ChangeDutyCycle(cycle_number)
```

Backward movement:



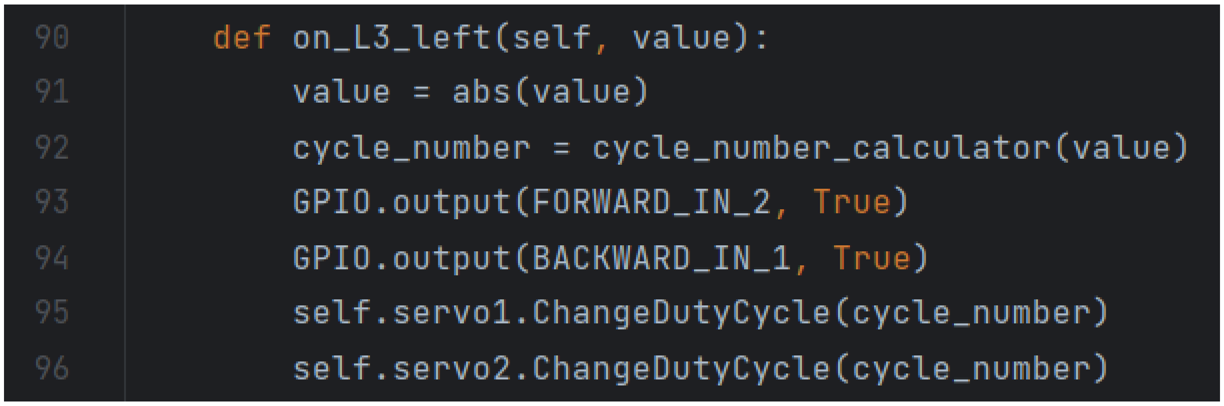
```python
    def on_L3_down(self, value):
        value = abs(value)
        cycle_number = cycle_number_calculator(value)
        GPIO.output(BACKWARD_IN_1, True)
        GPIO.output(BACKWARD_IN_2, True)
        self.servo1.ChangeDutyCycle(cycle_number)
        self.servo2.ChangeDutyCycle(cycle_number)
```

Right turn:



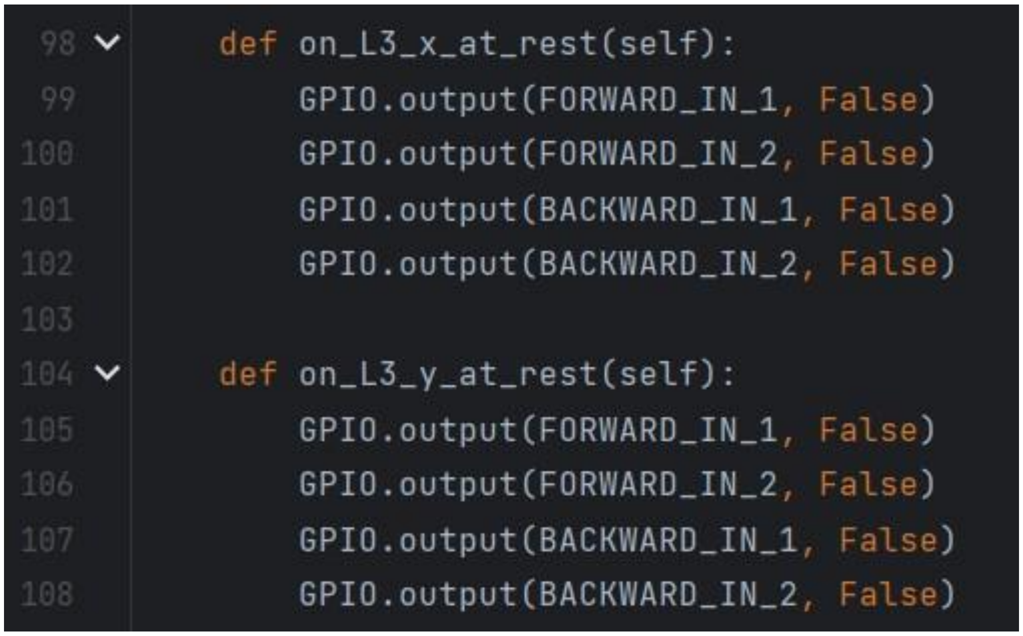
```python
    def on_L3_right(self, value):
        value = abs(value)
        cycle_number = cycle_number_calculator(value)
        GPIO.output(FORWARD_IN_1, True)
        GPIO.output(BACKWARD_IN_2, True)
        self.servo1.ChangeDutyCycle(cycle_number)
        self.servo2.ChangeDutyCycle(cycle_number)

```

Left turn:



```
    def on_L3_left(self, value):
        value = abs(value)
        cycle_number = cycle_number_calculator(value)
        GPIO.output(FORWARD_IN_2, True)
        GPIO.output(BACKWARD_IN_1, True)
        self.servo1.ChangeDutyCycle(cycle_number)
        self.servo2.ChangeDutyCycle(cycle_number)
```

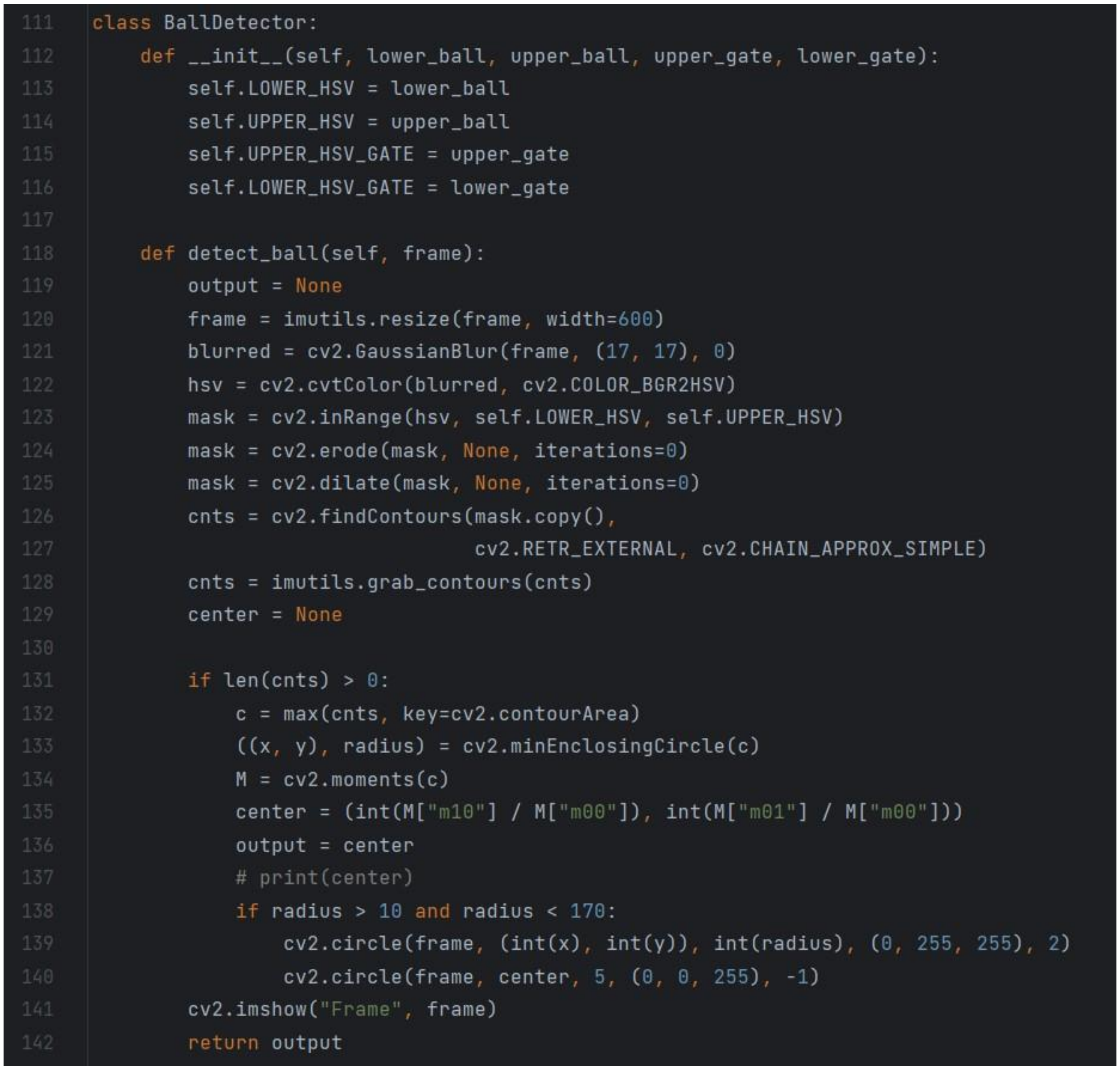
After returning the analog stick to its initial position, we deactivate all activated pins to stop the robot from moving:



```
    def on_L3_x_at_rest(self):
        GPIO.output(FORWARD_IN_1, False)
        GPIO.output(FORWARD_IN_2, False)
        GPIO.output(BACKWARD_IN_1, False)
        GPIO.output(BACKWARD_IN_2, False)

    def on_L3_y_at_rest(self):
        GPIO.output(FORWARD_IN_1, False)
        GPIO.output(FORWARD_IN_2, False)
        GPIO.output(BACKWARD_IN_1, False)
        GPIO.output(BACKWARD_IN_2, False)
```

In the next class, we have written functions to detect the ball and the goal.

Ball Detection:



```python
class BallDetector:
    def __init__(self, lower_ball, upper_ball, upper_gate, lower_gate):
        self.LOWER_HSV = lower_ball
        self.UPPER_HSV = upper_ball
        self.UPPER_HSV_GATE = upper_gate
        self.LOWER_HSV_GATE = lower_gate

    def detect_ball(self, frame):
        output = None
        frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (17, 17), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, self.LOWER_HSV, self.UPPER_HSV)
        mask = cv2.erode(mask, None, iterations=0)
        mask = cv2.dilate(mask, None, iterations=0)
        cnts = cv2.findContours(mask.copy(),
                                cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        center = None

        if len(cnts) > 0:
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
            output = center
            # print(center)
            if radius > 10 and radius < 170:
                cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
                cv2.circle(frame, center, 5, (0, 0, 255), -1)
        cv2.imshow("Frame", frame)
        return output
```

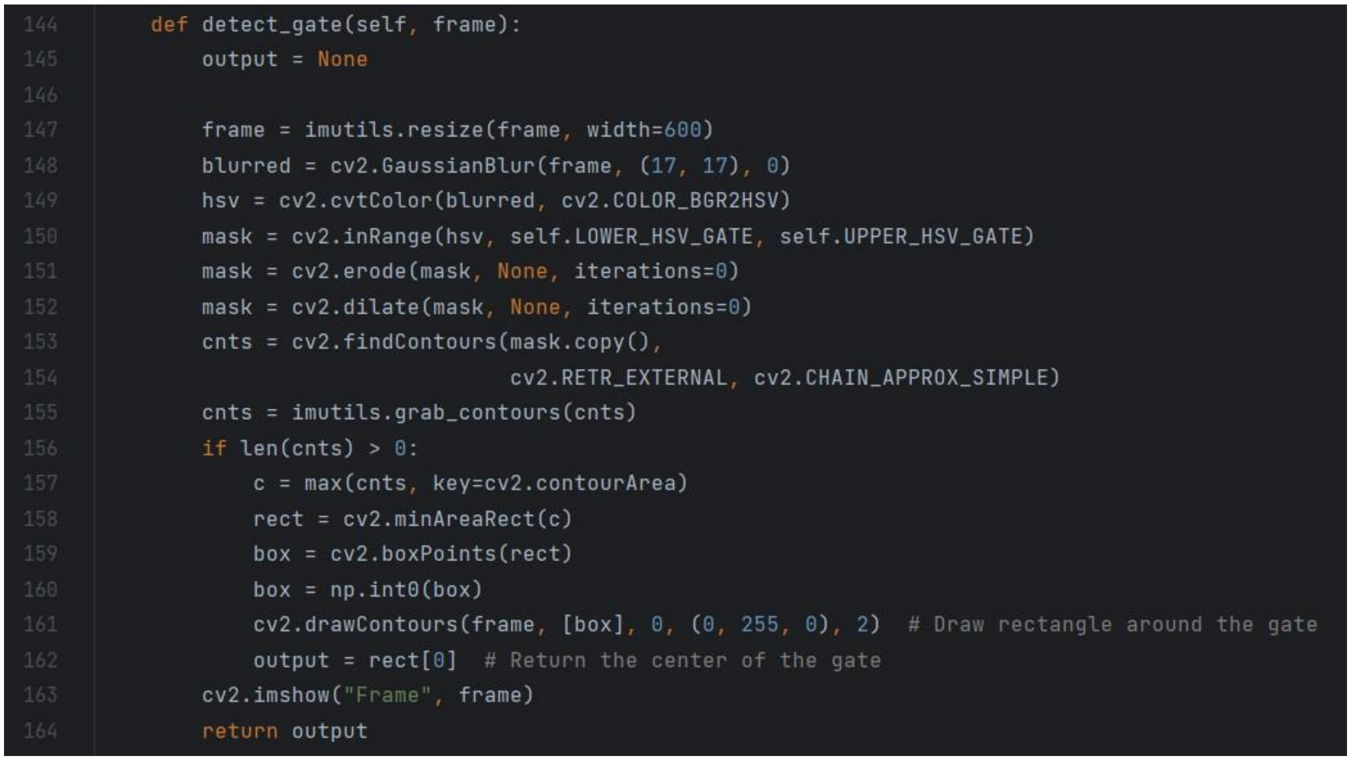
In the detect_ball function, the color and edge of the ball are identified in the surrounding environment. For the ball to be detectable in both low-light and bright environments, we need to specify the upper and lower HSV color range for the ball.

Therefore, in each frame, we look for a circular object with a specific color range.

If there is a circular environment around that area, we measure the pixels and then find the center of the circle and show it in the output.

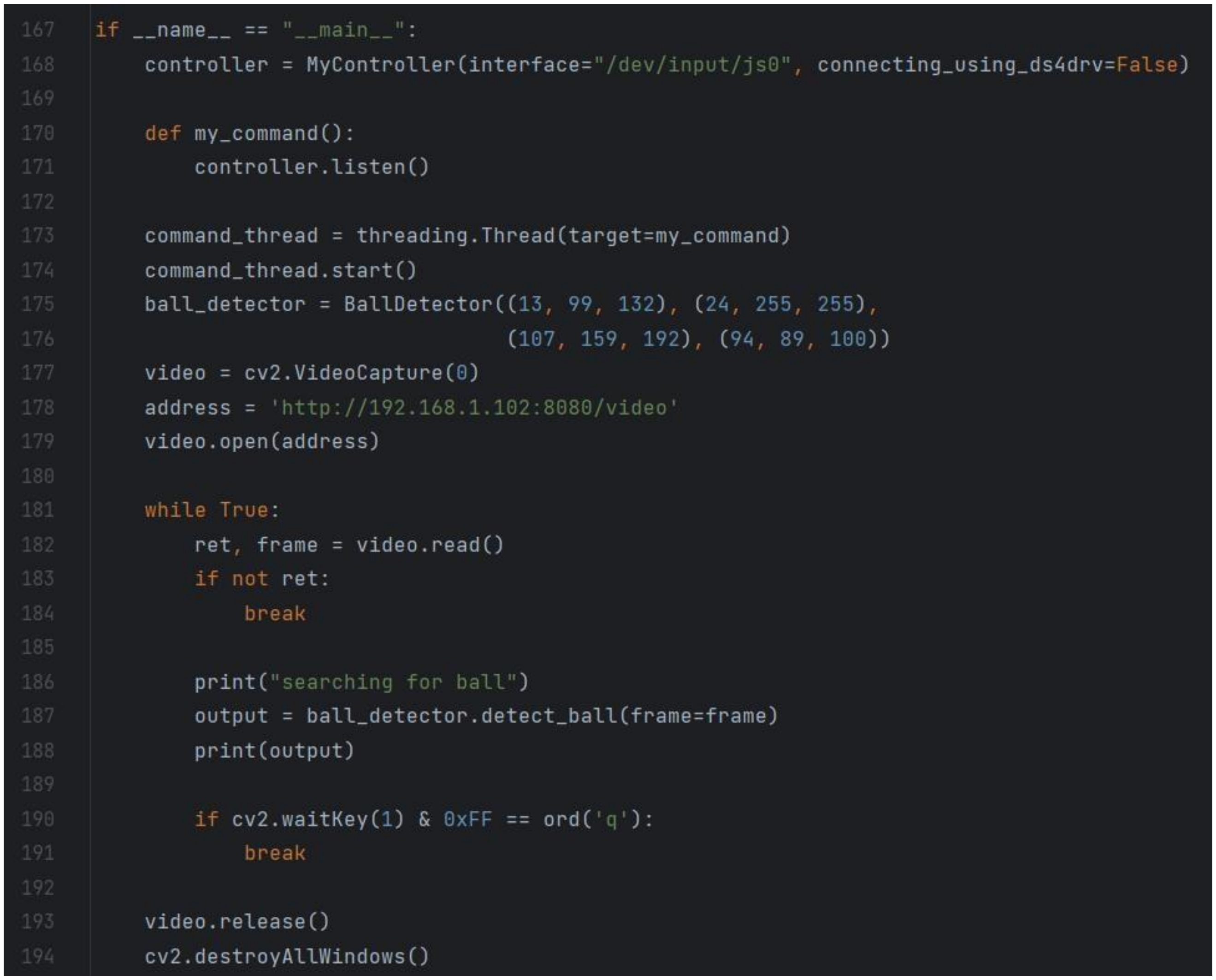
Gate Detection:

The detect_gate function works similarly and if a rectangular shape is detected, its center is marked and shown in the output image:



```python
    def detect_gate(self, frame):
        output = None

        frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (17, 17), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, self.LOWER_HSV_GATE, self.UPPER_HSV_GATE)
        mask = cv2.erode(mask, None, iterations=0)
        mask = cv2.dilate(mask, None, iterations=0)
        cnts = cv2.findContours(mask.copy(),
                                cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        if len(cnts) > 0:
            c = max(cnts, key=cv2.contourArea)
            rect = cv2.minAreaRect(c)
            box = cv2.boxPoints(rect)
            box = np.int0(box)
            cv2.drawContours(frame, [box], 0, (0, 255, 0), 2)  # Draw rectangle around the gate
            output = rect[0]  # Return the center of the gate
        cv2.imshow("Frame", frame)
        return output
```

The main function can be seen in the image below:



```python
if __name__ == "__main__":
    controller = MyController(interface="/dev/input/js0", connecting_using_ds4drv=False)

    def my_command():
        controller.listen()

    command_thread = threading.Thread(target=my_command)
    command_thread.start()
    ball_detector = BallDetector((13, 99, 132), (24, 255, 255),
                                 (107, 159, 192), (94, 89, 100))
    video = cv2.VideoCapture(0)
    address = 'http://192.168.1.102:8080/video'
    video.open(address)

    while True:
        ret, frame = video.read()
        if not ret:
            break

        print("searching for ball")
        output = ball_detector.detect_ball(frame=frame)
        print(output)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video.release()
    cv2.destroyAllWindows()
```

Here, in order to have simultaneous control of the robot and process the image alongside it, we use the my_command function and pass it to Command_thread to read instructions from the PlayStation controller input. This allows the robot to start moving and execute subsequent instructions in parallel.

Using the IP Webcam app on the mobile phone connected to the robot, we can view it on our display screen. We just need to give the IP address to the program to have a window of the video captured from the front of the robot.