جامعة أبوظبي
ABU DHABI UNIVERSITY
College of Engineering
Electrical and Computer Engineering Department

Project Report: Self-Driving Car

Amer Barhoush (1064958), Yazeed Eldigair (1074120)
Dr. Mohammed Ghazal

I. Introduction

In this project, we were required to design and implement a self-driving car using the Raspberry Pi 4 as an alternative to the NVIDIA Xavier kit. There are many limitations to this approach since the RPi 4 is not as powerful as the Xavier to run AI and machine learning code/algorithms. The main approach for applying self-driving car using the RPi 4 and RPi camera module relies on lane detection using openCV and PID control. In this report, we will discuss how we applied computer vision and PID control using python on the RPi 4 to develop a fully self-driving car. We will also discuss the major design of our robot which includes the system overview, component design and process. Moreover, we will apply different tests which contain different tracks to evaluate the performance of our design.

II. Hardware Components

The main components we have used in this project are:

- Raspberry Pi 4: A complex microprocessor that can be used as a computer or to build embedded based systems.
- Raspberry Pi Camera Module: This camera module was specially designed for the Raspberry Pi.
- Xplore 7000 mAh Power bank: This 5V DC and 2 A output power bank will be the main source to power the raspberry pi through Type C to USB cable.
- Dual H-bridge Motor Driver (L298N): This motor driver will allow us to control up to 2 DC motors.
- 6V DC motors: These DC motors will be used to move the car (We used 2).
- AA Batteries: these batteries will be used to power the H-bridge which will support our motors. (We used 4 AA batteries)
- microSD card: This will be the memory card to store the OS of the raspberry pi (The OS we chose is Raspbian)

III. System Overview

The system overview of our design can be shown in Figure 1.

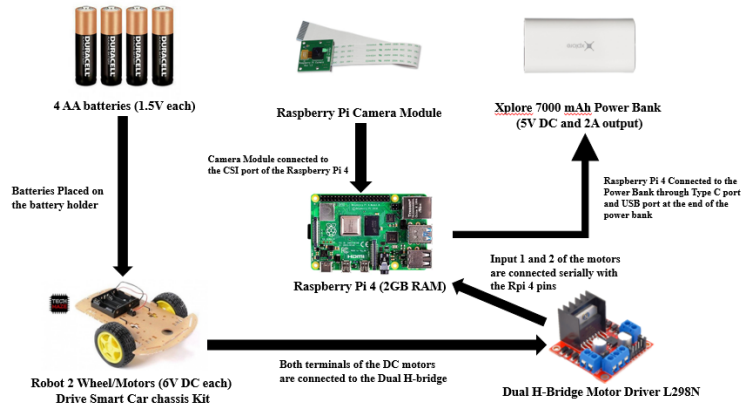

Fig. 1. System overview of Self-Driving Car controlled by Raspberry Pi 4.

IV. Design Process

In this section, we will discuss the main design process followed to reach the final design of our self-driving car.

A. Setting up the hardware

1) Format the microSD and burn/flash the Raspbian OS on it.
2) Insert the microSD into the Raspberry Pi 4
3) Boot the RPI 4 on the laptop using VNC viewer by finding the IPv4 address of the RPI 4 (You can use ipconfig /all command in the cmd to get the RPI 4 ip address which will be connected to the Router through ethernet connection).
4) Enable the picamera on the Rpi 4; using the (sudo raspi-config) command on the bash terminal or openning it from the Rpi 4 settings.
5) Connect the RPI camera module using the CSI port found on the RPI4.
6) Install Python 3 and openCV; by using the command (sudo apt install python3 idle3) you can install the latest version of python3; and by using the command (sudo apt-get install python3-opencv) you can install openCV on python3.
7) Create new python file using the bash or from the Rpi home folder; use the command (nano self_driving.py) to create a python file called self_driving)
8) Then, write the lane detection code on the python file created (The code will be discussed in the upcoming sections)
9) Connect the positive and negative sides of each DC motor to the output pins of the dual h-bridge (4 output pins in total where 2 sides of each motor will be connected to 2 out pins)
10) Connect two input pins from the dual h-bridge to any GPIO pins on the RPi 4 (We chose pin number 20 and 21 since they are the closest to the h-bridge side). The reason we only connected 1 input for each motor because we do not need to change the direction of the DC motors.
11) Connect the dual h-bridge to the 4 AA batteries holder to run the DC motors.
12) Connect the Rpi 4 with the power bank to supply it with the right voltage and amperage (5V and 2A)
13) We used other material to place the Rpi camera module high enough, so it can capture better view of the track; we have also used A0 paper + black tape to design the track that the robot will drive on.

The final design of our self-driving car can be shown in Figure 2.

Fig. 2. Final design of the self-driving car controlled by Rpi 4.

The track used for this project can be shown in Figure 3; the reason it is curved is to test how well will the lane detection be given a curve instead of a straight line.

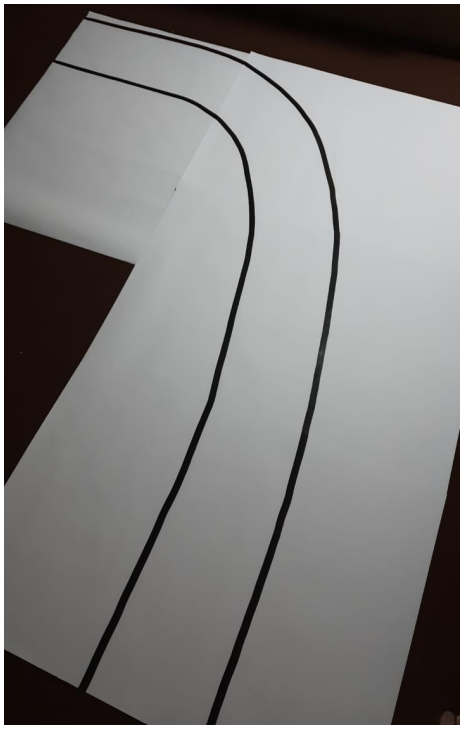

Fig. 3. Simple curved track designed using A0 paper and black tape.

B. Software implementation

The first step of implementing the software of the self-driving car is to be able to detect the lane boundaries. Determining the midpoint between the two lane boundaries allows us to calculate the desired position of our car. Using that we can get a value of the current error. The full code can be seen in Appendix A.

We first import all the necessary libraries. We instantiate our camera object using **PiCamera()** and set values for the frame rate and resolution parameters.

```
camera = PiCamera()
camera.resolution = (width, height)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(width, height))
```

We then initialize our control variables for the two DC motors. PWM signals are used to control the speed of the motors by adjusting the duty cycles of the signals. By controlling the speeds of the two motors, we are able to control the steering angle of the car allowing for lane keeping.

```
GPIO.setmode(GPIO.BCM)
leftDC = 21
rightDC = 20
GPIO.setup(leftDC, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(rightDC, GPIO.OUT, initial=GPIO.LOW)
pwmleft = GPIO.PWM(leftDC, 100)
pwmright = GPIO.PWM(rightDC, 100)
```

The propotional (**Kp**), dereivative (**Kd**) and integral (**Ki**) constants are set and tuned after several test runs of the car. The current time is saved using the function **time.time()**. All other parameters used for the PID control calculations are set to 0.

```
Kp = 0.2; Kd = 1; Ki = 0.01
currentTime = time.time()
currentError = 0
previousError = 0
derivative = 0
integral = 0
```

The current position, **currentX**, is set to be half of the width of the frame or the center of the camera. The desired position, **desiredX**, is initially set to be the current position.

```
currentX = width / 2
desiredX = currentX
```

We then begin our main loop to this code. An image is read from the frame of the camera, seen in Figure 4. Before trying to detect the new desired position of the car, we preform pre-processing of the frame. Firstly, we use a 5x5 Gaussian filter to reduce the presence of noise in the frame. We then turn the RGB image to a gray scale image. We perform Otsu thresholding to obtain a binarized image of the frame. However, this image needs to be inverted so that the lane pixels appear as white pixels, as in Figure 5. We then perform opening on our image as our morphological operation. This performs erosion followed by the dilation of white pixels. This combats the presence of false positives in our binarized frame as observed in Figure 6.

```
    img = frame.array
    img = cv2.GaussianBlur(img, (5, 5), 0)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret3, img1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY_INV +
        cv2.THRESH_OTSU)
    kernel = np.ones((8, 8), np.uint8)
    img = cv2.morphologyEx(img1, cv2.MORPH_OPEN, kernel)
```

Fig. 4. Original frame obtained from RPi camera.

Fig. 5. Using Otsu thresholding followed by inversion.

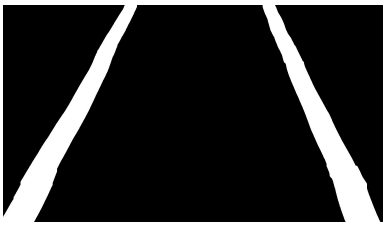

Fig. 6. After performing opening on the binarized image.

Next we begin extracting points for the left and right lane boundaries. This is done by scanning the image from left to right and then right to left at a certain height of the image to detect white pixels. The process is repeated at different heights. When encountering white pixels, their coordinates are appended to the corresponding array. The code segment below illustrated this process.

```
while y > 0:
    for x in range(width):
        if img[y, x] > 200:
            leftPoints = np.append( leftPoints , [y, x])
            break
    for x in range(width - 1, 0, -1):
        if img[y, x] > 200:
            rightPoints = np.append( rightPoints , [y, x])
            break
    y = int(y - 0.15 * height)
```

Finally, using the points obtained, we plot lines for the left and right lane boundaries using the **cv2.line()** function. The lines are plotted on our original frame which is then displayed on the screen. The code segment is shown below. Figure 7 shows the detected lane boundaries.

```
if len( leftPoints ) > 0:
    for i in range(0, len( leftPoints ) - 2, 2):
        cv2.line(img_original , ( leftPoints [i + 1], leftPoints [i]),
                 ( leftPoints [i + 3], leftPoints [i + 2]),
                    (255, 0, 0), 3)
        cv2.line(img_original , ( rightPoints [i + 1], rightPoints [i]),
                 ( rightPoints [i + 3], rightPoints [i + 2]),
                    (0, 255, 255), 3)
cv2.imshow('Camera Stream', img_original)
```

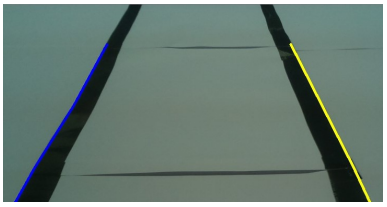

Fig. 7. Lane detection.

Moving on, we being to write the code for the PID control. We first calculate our new desired position, **desiredX**, and update our PID parameters. The time difference, **dT**, is calculated by subtracting the new current time from the previous time. The current error is calculated by the subtraction of the desired position from the current position. We also calculate the derivative and integral errors. With all the required values being updated, we can now calculate the new speed difference of our motors.

```
desiredX = ( leftPoints [3] + rightPoints [3]) / 2
dT = (currentTime - time.time()) * 10
currentTime = time.time()
currentError = currentX - desiredX
derivative = ( currentError - previousError ) / dT
integral = integral + currentError *dT

speed = Kp * currentError + Kd * derivative + Ki * integral
```

The new duty cycles of the PWM signals of the motors are fed to each one. Conditional statements are used to make sure that the PWM values do not go above or below upper and lower thresholds.

```
leftPwm = 50 + speed
rightPwm = 65 - speed
if leftPwm > 70:
    leftPwm = 70
if rightPwm > 80:
    rightPwm = 80
if leftPwm < 0:
    leftPwm = 0
if rightPwm < 0:
    rightPwm = 0

pwmleft. start (leftPwm)
pwmright. start (rightPwm)
```

The process is then repeated for the next frame of the video stream.

V. Results and Discussion

A. Test 1: Straight track

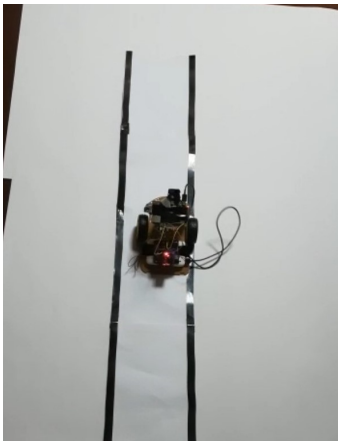

Fig. 8. Straight track test.

Figure 8 shows an instance of testing our system on a straight track. The car successfully managed to reach the end of the path while not going of track. Some jittering of the car was observed but a good performance overall.

B. Test 2: Curved track

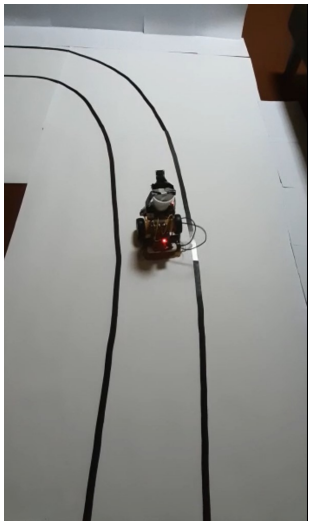

Fig. 9. Curved track test.

```
rightPwm is:  65.33093301400715
==============================
dT is:  -1.3164472579956055
current error is:  17.5
derivative is:  -13.293354438403501
speed is:  -9.793354438403501
leftPwm is:  30.2066455615965
rightPwm is:  64.7933544384035
==============================
```

Fig. 10. Monitoring parameters during training.

Figure 9 shows an instance of testing the car on a curved track. The car manages to complete the turn and reach the end of the track. However, alot of jittering was observed in this test. We monitored our system values, as shown in Figure 10, in order to tune the values of the PID constants and obtain better results.

VI. Conclusion

To conclude with, we were able to successfully apply pre-processing techniques, lane detection and PID control to navigate a car autonomously. This was implemented on the Raspberry Pi 4 and with the help of the RPi camera module; our design was able to drive on both tracks (straight and curved) while being between the lanes we have made out of black tapes. We made sure that we use white background and black tapes as lanes, so the RPi camera can detect those lanes without any trouble. Although in the real world, the input frames are subject to a lot more noise. so we have made sure that the camera is live streaming with negligible latency. As future improvements, we want to enhance the design of our robot to position the camera better. We also want to use a higher fps camera for smoother driving of the car.

Appendix A
Self-driving car code

```
import cv2
import numpy as np
import time
from picamera.array import PiRGBArray
from picamera import PiCamera
import RPi.GPIO as GPIO
from matplotlib import pyplot as plt

width = 480
height = 480

camera = PiCamera()
camera.resolution = (width, height)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(width, height))

time.sleep(0.1)

GPIO.setmode(GPIO.BCM)
leftDC = 21
rightDC = 20
GPIO.setup(leftDC, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(rightDC, GPIO.OUT, initial=GPIO.LOW)
pwmleft = GPIO.PWM(leftDC, 100)
pwmright = GPIO.PWM(rightDC, 100)

Kp = 0.2; Kd = 1; Ki = 0.01
previousTime = 0
currentTime = time.time()
currentError = 0
previousError = 0
derivative = 0
integral = 0

currentX = width / 2
desiredX = currentX

for frame in camera.capture_continuous(rawCapture, format="bgr",
    use_video_port=True):
    img = frame.array
    img_original = img

    img = cv2.GaussianBlur(img, (5, 5), 0)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret3, img1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY_INV +
        cv2.THRESH_OTSU)
    kernel = np.ones((8, 8), np.uint8)
    img = cv2.morphologyEx(img1, cv2.MORPH_OPEN, kernel)

    y = int(0.85 * height)

    leftPoints = np.array([])
    rightPoints = np.array([])

    while y > 0:
        for x in range(width):
            if img[y, x] > 200:
                leftPoints = np.append(leftPoints, [y, x])
                break
        for x in range(width - 1, 0, -5):
            if img[y, x] > 200:
                rightPoints = np.append(rightPoints, [y, x])
                break
        y = int(y - 0.15 * height)

    leftPoints = leftPoints.astype('int32')
    rightPoints = rightPoints.astype('int32')

    if len(leftPoints) > 0:
        for i in range(0, len(leftPoints) - 2, 2):
            cv2.line(img_original, (leftPoints[i + 1], leftPoints[i]),
                (leftPoints[i + 3], leftPoints[i + 2]),
                    (255, 0, 0), 3)
            cv2.line(img_original, (rightPoints[i + 1], rightPoints[i]),
                (rightPoints[i + 3], rightPoints[i + 2]),
                    (0, 255, 255), 3)
    cv2.imshow('Camera Stream', img_original)

    if len(leftPoints) >= 4:
        desiredX = (leftPoints[3] + rightPoints[3]) / 2
    dT = (currentTime - time.time()) * 10
    currentTime = time.time()
    currentError = currentX - desiredX
    derivative = (currentError - previousError) / dT
    integral = integral + currentError*dT

    speed = Kp * currentError + Kd * derivative + Ki * integral

    leftPwm = 50 + speed
    rightPwm = 65 - speed
    if leftPwm > 70:
```

```
88              leftPwm = 70
89          if rightPwm > 80:
90              rightPwm = 80
91          if leftPwm < 0:
92              leftPwm = 0
93          if rightPwm < 0:
94              rightPwm = 0
95          print ("speed is: ", speed)
96          print ("leftPwm is: ", leftPwm)
97          print ("rightPwm is: ", rightPwm)
98
99          pwmleft.start (leftPwm)
100         pwmright.start (rightPwm)
101
102         rawCapture.truncate (0)
103         print ("================================")
104         key = cv2.waitKey(1) & 0xFF
105         if key == ord("q"):
106             pwmleft.start (0)
107             pwmright.start (0)
108             break
```