Project Report

RPi-based Lane-Assist Autonomous Car

Mohammad Fares Aljamous 1088672

Omar Mohammad Zeineh 1088546

Hadi Albanna 1088677

Supervised By: Eng. Gasm and Dr. Sajid

Submitted: February 20, 2025

Contents

List of Figures

List of Tables

Abstract

This project creates an RPi-based lane-assist autonomous vehicle that incorporates real-time lane detection, obstacle identification, and adaptive motion control via computer vision and PID controllers. The system uses the Picamera2 module to capture live video, OpenCV to detect edges and identify lane markings, and the YOLO model to recognize objects. Two PID controllers govern vehicle movement: one for lane centering based on identified lane borders and another for obstacle stopping, resulting in smooth and adaptable navigation. The car dynamically adjusts motor speeds to maintain its position and comes to a stop when an object is spotted within a crucial distance. The system is tested on a two-lane track with various obstacle locations, displaying accurate lane tracking and stopping behavior. The results demonstrate the balance between hardware efficiency and software complexity, with the PID-based control capable of managing real-time adjustments. Future enhancements may include improved lane curvature prediction and adaptive speed control to optimize real-world deployment.

1 Introduction

Autonomous vehicles rely on real-time lane identification and obstacle avoidance to navigate safely, making these features critical in self-driving technology. This project aims to create an RPi-based lane-assist automobile that recognizes lanes, identifies impediments, and adjusts movement via computer vision and PID controllers. The system uses a Raspberry Pi and a camera to process live video to follow lane lines, while a YOLO-based object identification model detects impediments. Two PID controllers regulate position and stopping one maintains the car centered, while the other guarantees it comes to a stop when an obstacle is too close. On a two-lane track with different obstacles, the system demonstrated seamless lane tracking and prompt stopping. This project helps to make autonomous navigation smarter and more dependable by combining vision-based control and real-time motor modifications.

1.1 Motivation

With the development of self-driving vehicles, safe and efficient navigation is more crucial than ever. Lane-keeping and obstacle avoidance are crucial for minimizing accidents and enhancing traffic flow, but many current systems are prohibitively expensive or unduly complicated. This project intends to develop a cheap, real-time lane-assist system based on Raspberry Pi, computer vision, and PID control, making autonomous navigation more accessible. This project helps to provide safer roads, less human error, and more efficient transportation systems by building a smart vehicle that can detect lanes, identify barriers, and modify movement dynamically. Self-driving cars, smart delivery robots, and industrial automation can all benefit from the technologies created here, paving the way for a more reliable and widespread future of intelligent mobility.

1.2 Problem Statement

Autonomous vehicles must move safely while preserving lane position and avoiding obstructions, but real-time accuracy in both jobs is still a barrier. Many present systems fail to balance precise lane tracking with responsive obstacle avoidance, particularly in dynamic situations. This project addresses the requirement for a dependable and cost-effective system that combines

computer vision and adaptive motor control to assure smooth navigation. We hope to make autonomous navigation more efficient and accessible by creating an RPi-based lane-assist system that recognizes lanes, identifies obstructions, and dynamically adjusts movement using PID controllers. This method increases safety, minimizes the need for human intervention, and improves the overall performance of self-driving technology.

1.3 Literature Review

Autonomous lane-following and obstacle avoidance systems have been extensively researched in both academic and commercial settings. Traditional lane-assist solutions use image processing techniques including edge recognition, Hough line transformation, and histogram-based lane tracking. These techniques enable vehicles to detect and follow lane markings by assessing contrast differences in road photographs. While computationally efficient, these algorithms frequently fail in low-light circumstances, with faded lane markings, or in complicated road environments where markings are ambiguous [1].

For lane recognition and obstacle identification, more advanced methods combine deep learning models like Convolutional Neural Networks (CNNs) with object detection algorithms like YOLO (You Only Look Once). These models can learn road details and adapt to varied driving situations, resulting in greater accuracy. Deep learning methods, on the other hand, necessitate significant computational capacity, which can be a constraint for low-cost embedded devices such as Raspberry Pi. Furthermore, training these models requires enormous datasets and significant fine-tuning, which may not be feasible for smaller-scale applications [2].

A hybrid technique that combines computer vision with PID-based motor control provides a useful balance of efficiency and precision. While deep learning can improve object recognition, classical edge detection approaches are still effective for lane tracking in well-marked situations. Our technology, which combines vision-based lane recognition with adaptive PID-controlled movement, seeks to solve the drawbacks of existing technologies while remaining cost-effective and computationally practical for embedded systems.

2 Design

We were given a bunch of requirements to work with, so while considering these requirements we designed and wrote, self driving algorithm, and built the car that would apply the algorithm.

2.1 Requirements Constraints, and Considerations

- Detect lane markings (white, yellow, dashed) on a two-lane track.
- Identify obstacles:
  - A far obstacle in the lane
  - A close obstacle in the lane
  - A closer obstacle out of the lane (to be ignored)
- Keep the robot in the center of the lane using Ackerman or differential control.
- Stop the robot before the closest in-lane obstacle (the close obstacle).
- After stopping, once the close obstacle is removed:
  - The robot must continue
  - Stop before the far obstacle
- Highlight the closest in-lane obstacle in yellow on the robot's display.
- Display other objects and lane markings in white.
- The live screen must indicate the lane type (e.g., white, yellow, dashed).

2.2 Design Process

In total there were 11 steps taken to design and apply the algorithm, and they are:

1. Importing the necessary libraries.
2. Initializing the camera and taking frames.

3. Preprocessing the frames.
4. Applying edge detection on preprocessed frames.
5. Defining the ROI (region of interest).
6. Line detection.
7. Finding the right boundary and the left boundary of the lane.
8. Controlling motors based on right boundary and left boundary.
9. Apply Object Detection.
10. Find MIO and control motors based on it.
11. Starting the motors.

2.2.1 Importing the necessary libraries.

At the start we had to import the libraries that we would need to use.

- cv2 (open computer vision 2) is the library we used for almost all of the computer vision related processes we had (excluding object detection), like filtering, finding edges, finding lines, displaying the frames, etc..

```
1 import cv2
```

- numpy is a library that helps the most when doing math, it has many mathematical constants such as pi or e, it also has many mathematical functions like sin(X), cos(x), tan(x), arctan(x) and many more.

```
1 import numpy as np
```

- time is a library that has a function we used called time(), it gives us the current time.

```
1 import time
```

- picamera2 is the library you would use when you want to use your pi camera module in your python code, it allows you to configure the camera settings and capture frames from the camera.

```
from picamera2 import Picamera2
```

- ultralytics is the library which includes YOLO, which will be used as our object detection model.

```
from ultralytics import YOLO
```

- gpiozero is the library that allows us to control the gpio of the raspberry pi.

```
from gpiozero import Motor
```

2.2.2 Initializing the camera and taking frames.

In this part, we initialized the camera with our needed configurations and we started taking frames with it.

1. Step one, we chose the height and width of the frame in pixels, and created a Picamera2() object for the camera that we would pass the configurations to, the configurations being the width and height of the frame, and color format of the frame, we chose "XRBG8888", where the eights represent the number of bits that each color gets, and the X is not used, and then we called the method start() on the object.

```
height =480
width=640
middle =((width//2),(height//2))
cam = Picamera2()
cam.configure(cam.create_video_configuration(main={"format":
    'XRGB8888',"size": (width, height)}))
cam.start()
```

2. Step two, we created the main program loop that will capture a frame, flip the frame, and then show the frame, and at any point you can

break the loop by pressing 'q'. The motor.stop() is for stopping the motors once we stop the loop.

```
while True:
    frame = cam.capture_array()[:,:,:3]
    frame = cv2.flip(frame, -1)
    cv2.imshow('Original', frame)
    key = cv2.waitKey(1) & 0xff
    if key == ord('q'):
        motor1.stop()
        motor2.stop()
        break

cam.stop()
cv2.destroyAllWindows()
```

2.2.3 Preprocessing the frames.

In this part, we took the frames captured by the camera and applied preprocessing on them, the preprocessing includes converting from RGB scheme to greyscale

1. Step one, we took the frame and applied the Gaussian filter on it to remove any noise, the size of the filter should be NxN where N is an odd number, and here we chose it to be 11x11.

```
frame = cv2.GaussianBlur(frame, (11, 11), 0)
```

2. Step two, we took the frame and changed the color scheme from RGB to greyscale, greyscale uses only one 8-bit channel to describe color instead of the usual three 8-bit channels that RGB uses.

```
gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
```

2.2.4 Applying edge detection on preprocessed frames.

In this part, we took the preprocessed frames and applied canny edge detection on them, with the lower threshold being 60 and upper threshold being

120. These threshold are not really constant, we kept changing them until we found what worked best.

```
edges = cv2.Canny(gray, 60, 120)
```

2.2.5 Defining the ROI (region of interest).

In this part, we defined the ROI, we used the variable pt to change how much of the frame to be removed (from the top of the frame), so if we choose 200, then 200 pixels would be removed from the top, the value of pt depends on the camera position and the track itself (you have to test to get a satisfactory one). We created the ROI of the preprocessed image, and the ROI of the edges image.

```
pt = 200 #200
.
.
.
...in while loop
    ROI = frame[pt:height, 0:width].copy()
    ROIedge = edges[pt:height, 0:width].copy()
```

2.2.6 Line detection.

In this part, we detected all lines that are in ROI of the edges image. We used the hough transform to find the lines, using the function HoughLines(image, r, theta, threshold), the r parameter tells us how much r should be quantized, theta parameter tells us how much theta should be quantized, and threshold tells us how many votes are needed for each line to be considered a line (in pixels).

```
lines = cv2.HoughLines(ROIedge, 1, np.pi / 180, 40 )
   lin = frame
   lin2 = ROIedge
```

2.2.7 Finding the right boundary and the left boundary of the lane.

In this part, we found the left and right boundary of the lane we are in, these boundaries are nothing but lines that are on the left part and right part of the screen.

1. Step one, we reset the left boundary and the right boundary, then we check if we got any lines from the hough transform, then we set the variables "prevRightDistanceFromMid" and "prevLeftDistanceFromMid" to 0, these variables will be used later to find the farthest line from the left of the middle of the screen and the farthest line from the right of the middle of the screen.

2. Step two, we iterate over all lines while getting their r and theta and then finding two points that are faraway from each other on each of the lines, using these two points we can find the angle that each line makes with the x axis and then we are able to remove lines that do not fit the angle threshold that we set, in this example we set the thresholds to ($\theta < \pi/2$ and $\theta > \pi/30$) or ($\theta > -\pi/2$ and $\theta < -\pi/30$).

3. Step three, if a line fits the angle threshold, we check whither the line is to the left of the screen or to the right of the screen, we check at the point at the bottom middle of the screen (x=width/2, y=height-pt), the reason we have to take pt into account is because we decreased the height of the frame by pt when finding the ROI.

4. Step four, for every left line we check the farthest one from the middle, and we do the same for the right lines, we do this to find the left boundary and the right boundary, the boundaries are the lines which are farthest from (x=width/2, y=height-pt) from the left and right respectively.

```
    rightLane = None
    leftLane = None
    if lines is not None:
        prevRightDistanceFromMid = 0
        prevLeftDistanceFromMid = 0
        for line in lines:
```

```
            r, th = line[0]
            a = np.cos(th)
            b = np.sin(th)
            x0 = a*r
            y0 = b*r
            x1 = int(x0 + 1000*(-b))
            y1 = int(y0 + 1000*a)
            x2 = int(x0 + 1000*(b))
            y2 = int(y0 + 1000*(-a))

            if x2-x1 != 0:
                angle = np.arctan((y2 - y1) / (x2 - x1))
                if (angle < np.pi/2 and angle > np.pi/30) or (angle
                    > -np.pi/2 and angle < -np.pi/30):
                    m = (y2 - y1) / (x2 - x1)
                    inter = -m*x1 + y1
                    midx = (height-pt-inter)/m
                    distanceFromMid = midx - width/2
                    if distanceFromMid > 0:
                        if distanceFromMid > prevRightDistanceFromMid:
                            rightLane = line[0]
                            prevRightDistanceFromMid = distanceFromMid
                    else:
                        if abs(distanceFromMid) >
                            prevLeftDistanceFromMid:
                            leftLane = line[0]
                            prevLeftDistanceFromMid =
                                abs(distanceFromMid)
```

2.2.8 Controlling motors based on right boundary and left boundary.

In this part, we controlled the motors based on whither the right and left lane boundaries are there or not, and based on the distance of each boundary from the middle of the bottom middle of the screen.

1. Step one, at the start of the python file, we first created the motor objects and we defined on which pins they will be, then we chose some values of kp, ki, kd for the PID controller that will keep the car in the

middle of the lane when both lines can be seen, we got these values by testing, and we ended up not using ki or kd since we did not have time to fine tune them. We also defined the base speed of the motor, and initialized the previous error, and the integral value.

```
motor1 = Motor(23, 14)
motor2 = Motor(24, 27)




Kp = 0.15 #0.25
Ki = 0
Kd = 0
Previous = time.time()
errorPrevious = 0
Integral = 0

base = 124
```

2. Step two, if any of the two boundaries exist we find their slopes and intersects so we can create two different points from each line that are far away from each other, we would then take these points and use them in the function line(image, start point, end point, color, thickness) which takes and image and draws a line on it, since the lines are generated from the ROI image, we had to offset the lines by pt so they are in the right place when we draw them on the original image.

```
        if rightLane is not None:
            r, th = rightLane
            a = np.cos(th)
            b = np.sin(th)
            xr0 = a*r
            yr0 = b*r
            xr1 = int(xr0 + 1500*(-b))
            yr1 = int(yr0 + 1500*a)
            xr2 = int(xr0 + 1500*(b))
            yr2 = int(yr0 + 1500*(-a))
            mr = (yr2 - yr1) / (xr2 - xr1)
```

```
            interr = -mr * xr1 + yr1
            lin = cv2.line(lin, (xr1, yr1+pt), (xr2, yr2+pt),
                (255, 255, 255), 2)
            lin2 = cv2.line(lin2, (xr1, yr1), (xr2, yr2),
                (255, 255, 255), 2)
        if leftLane is not None:
            r, th = leftLane
            a = np.cos(th)
            b = np.sin(th)
            xl0 = a*r
            yl0 = b*r
            xl1 = int(xl0 + 1500*(-b))
            yl1 = int(yl0 + 1500*a)
            xl2 = int(xl0 + 1500*(b))
            yl2 = int(yl0 + 1500*(-a))
            ml = (yl2 - yl1) / (xl2 - xl1)
            interl = -ml * xl1 + yl1
            lin = cv2.line(lin, (xl1, yl1+pt), (xl2, yl2+pt),
                (255, 255, 255), 2)
            lin2 = cv2.line(lin2, (xl1, yl1), (xl2, yl2),
                (255, 255, 255), 2)
```

3. Step three

   - if both boundaries exist, then we find the distance of each boundary from x=width/2 at y=height, the error that will be used for the PID is the sum of both distances (we do not take the absolute value of either distance so the distance on the left gives a negative value and the one on the right gives a positive value). after calculating the error we find the control value and we use it control the speed of the motors.

```
        if leftLane is not None and rightLane is not None:
            if (ml-mr) != 0:
                x = (-interl+interr)/(ml-mr)
                y = mr*x + interr
                midxl = (height - pt - interl) / ml
                midxr = (height - pt - interr) / mr
                distR = midxr-width/2
                distL = midxl-width/2
```

```
9                 error = distR + distL
10                Current = time.time()
11                dt = Current - Previous
12                Derivative = (error - errorPrevious) / dt
13                Integral = Integral + error * dt
14                Control = Kp * error + Ki * Integral + Kd
                      * Derivative
15                Previous = Current
16                errorPrevious = error
17                rightmotor = base - Control
18                leftmotor = base + Control
```

- if one of the boundaries exists but the other one does not, then we start turning towards the boundary that does not exist, this helps keep the car in lane, since many times when the car faces a turn, the camera would only see the lane which is the one opposite to the turn.

```
1         if leftLane is not None and rightLane is None:
2             rightmotor = 0
3             leftmotor = base
4         if leftLane is None and rightLane is not None:
5             rightmotor = base
6             leftmotor = 0
```

4. Step four, we just make sure that the motor speed is between 0 and 255.

```
1             if(rightmotor > 255):
2                 rightmotor = 255
3             elif(rightmotor < 0) :
4                 rightmotor = 0
5             if(leftmotor > 255):
6                 leftmotor = 255
7             elif(leftmotor < 0):
8                 leftmotor = 0
```

2.2.9 Apply Object Detection.

In this part, we used yolov8-nano to detect objects which are in the frame.

1. Step one, export the model as an openVino model and reduce the image size given to the model to make the object detection fast enough to be practical, openVino models are one of the best types of models to run on edge devices, by exporting the model we were able to the inference time down from 400ms to 30-50 ms. Originally the inference was being done on a 640x640 image, we brought that down to 256x256.

```
model = YOLO("yolov8n.pt")

model.export(format="openvino", imgsz=256)

model = YOLO('yolov8n_openvino_model/')
```

2. Step two, after exporting the model, using it was very straight forward, we only had to call the function predict(image, image size), which uses inference on the image, then we used the function plot() to put the rectangles generated by the inference on the image.

```
results = model.predict(lin, imgsz=256)
lin = results[0].plot()
   obstacles = []
   for r in results:
       for obj in r.boxes:
           x, y, w, h = obj.xywh.tolist()[0]
           label = obj.cls
           obstacles.append((x, y, w, h, label))
```

2.2.10 Find MIO and control motors based on it.

In this part, we found the MIO and used the distance from the bottom of the screen to the MIO as the error for the object PID

1. Step one, as we did with the PID of the lane keeping, we initialized the values of Kp, Ki ,Kd for the object PID, we put Kp as high value to immediately stop if an MIO is found.

```
KpObj = 10
KiObj = 0
KdObj = 0
errorPreviousObj = 0
IntegralObj = 0
distanceObject = 0
```

2. Step two, in this step we iterate over each obstacle that was detected previously, and we find ones that are in the same lane as the car, the lane is defined by the two boundaries of the lane and the intersection between them defined by x and y in the code. We do not look at the center of the object, but we look at the bottom center of the object defined by the point (x, y+h/2) where (x,y) is the middle point of the box (not defined by x and y in the code since x and y belong to the intersection).

3. Step three, when finding the objects that are in lane, we also run an algorithm which finds the closest one to the bottom of the screen (MIO), based on the bottom point of the box, after finding the closest one, we find the distance from the bottom of the screen, and the error is based on this distance from the MIO, after finding the error, we calculate the control, and we add it to the speed of the motors.

```
if obstacles:
   closest = 0
   index = 0
   for i in range(len(obstacles)):

       if(obstacles[i][1] + obstacles[i][3]/2 > y):
           rx = (obstacles[i][1]+obstacles[i][3]/2-interr)/mr
           lx = (obstacles[i][1]+obstacles[i][3]/2-interl)/ml
           print("x: ", x, "y: ", y)
           if(obstacles[i][0] < rx and obstacles[i][0] > lx):
               if obstacles[i][1]+obstacles[i][3]/2 > closest:
                   index = i
                   closest = obstacles[i][1]+obstacles[i][3]/2
                   distanceObject = 480-closest


```

```
    errorObj = -distanceObject
    print(errorObj)
    CurrentObj = time.time()
    dtObj = CurrentObj - PreviousObj
    DerivativeObj = (errorObj - errorPreviousObj) / dtObj
    IntegralObj = IntegralObj + errorObj * dtObj
    ControlObj = KpObj * errorObj + KiObj * IntegralObj +
        KdObj * DerivativeObj
    errorPreviousObj = errorObj
    rightmotor = rightmotor + ControlObj
    leftmotor = leftmotor + ControlObj
```

2.2.11 Starting the motors

In this part, we put all of the code together and gave the calculated motor speeds to the motors, and showed the view of the camera.

```
motor1.forward(rightmotor/255)
motor2.forward(leftmotor/255)

print("Left Motor: " + str(leftmotor) + " Right Motor: " +
    str(rightmotor))
cv2.imshow('Lines and Objects', lin)
```

2.3 System Overview

The lane-assist autonomous vehicle is a multi-component system that combines computer vision, real-time processing, and motor control to enable autonomous navigation. At its core, a Raspberry Pi 5 processes live video from the Pi Camera using OpenCV for lane detection and YOLO for obstacle recognition. The lane detection technique uses edge detection and the Hough Transform to identify lane borders, whereas the obstacle detection model classifies and tracks objects in real time. The PID controllers change the speed and direction of the DC motors, which are controlled by an H-bridge motor driver, to keep the vehicle centered and stop at detected impediments. An additional power supply delivers consistent power to the Raspberry Pi and its motors, preventing performance drops.An additional power supply delivers consistent power to the Raspberry Pi and its motors,

preventing performance drops. The system continuously interprets camera input, making real-time judgments to ensure safe navigation, with all components working together to accomplish smooth lane-following and responsive obstacle avoidance.Th work flow can be seen in the following Figure 1.

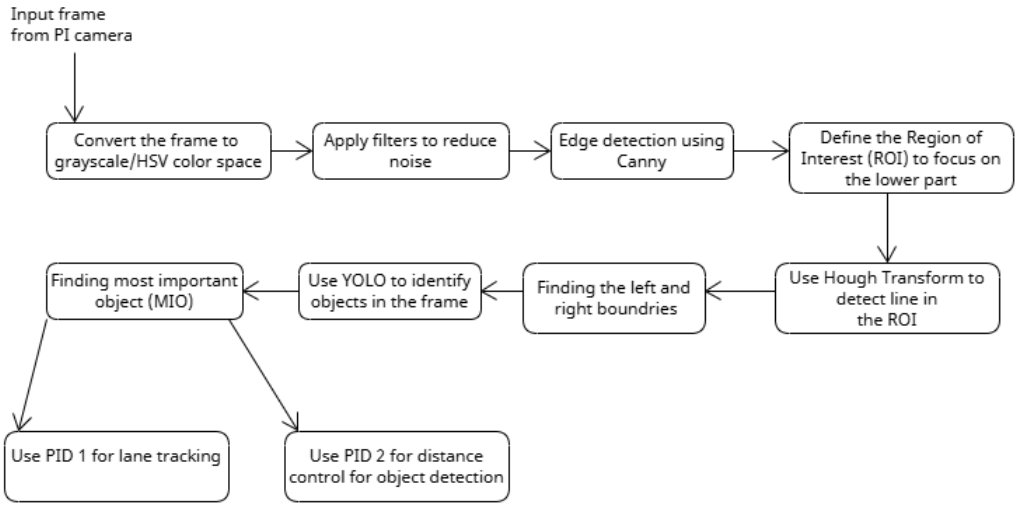
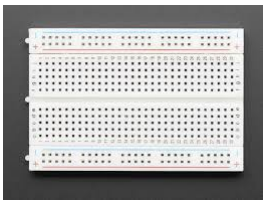


Figure 1: WORK FLOW.

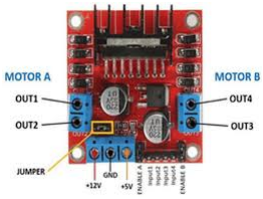
2.4 Component Design

The autonomous lane-assist vehicle is composed of multiple interconnected components that work together to provide real-time navigation, lane sensing, and obstacle avoidance. A Raspberry Pi 5 serves as the central processing unit, processing images, detecting objects, and controlling motors. A Pi Camera is linked to the Raspberry Pi, recording live video frames that are then processed with OpenCV for lane detection and YOLO for obstacle recognition. The DC motors that propel the car are controlled by an H-Bridge motor driver, which receives PWM signals from the Raspberry Pi and dynamically adjusts the speed and direction. To ensure stable functioning, an additional power supply powers both the Raspberry Pi and the motors.All of these components are put on a car kit that serves as the the base, allowing the system to be small and movable. This seamless connection enables the vehicle to autonomously detect lanes, avoid obstructions, and move effectively.



(a) Breadboard

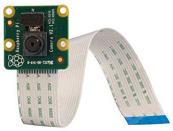
(b) Car Kit



(c) L293D Motor Driver

(d) DC motors

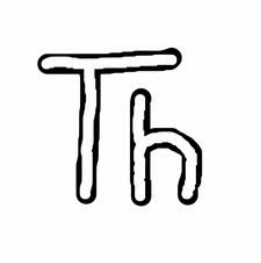
(e) Raspberry Pi 5



(f) OI camera

(g) Power Bank

(h) Jumper wires



(i) Python on Thonny IDE

Figure 2: Components

3 Experimental Testing and Results

3.1 Testing Plan and Acceptance Criteria

This project was primarily implemented in Python, so we just uploaded the code to RASPI 5 and began testing depending on the outputs displayed on the screen. After verifying the logic of the code, we began testing a rudimentary lanes that we constructed to see how the car would react to them. If the car began to move smoothly, we would increase the sharpness or number of the turns. If the automobile fails, we will alter the speed or any thresholds we set and attempt again. This guaranteed that our vehicle could respond to any alteration.

3.2 Results

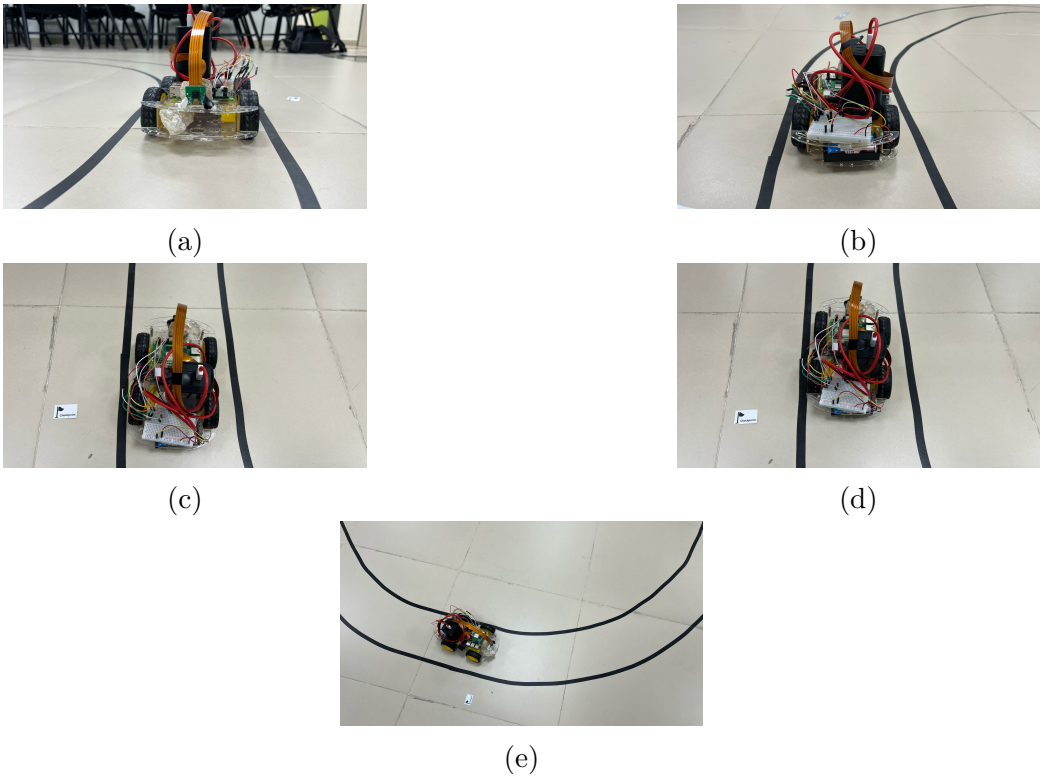
The project was successful in implementing real-time lane detection and obstacle avoidance utilizing computer vision and PID control. Under ideal conditions, the system accurately tracked lanes and spotted impediments; however, performance varied in low-light and complex environments. Motor control was effective, with only minimal over corrections in sharp corners. Processing delays on the Raspberry Pi 5 indicate possible hardware optimizations for future upgrades. Overall, the system achieved its objectives by demonstrating a workable and cost-effective solution to autonomous navigation.

3.2.1 Hardware implementation

Our lane-assist autonomous vehicle's hardware arrangement successfully integrated crucial components, allowing for real-time lane identification and obstacle avoidance. The automobile kit provided a sturdy foundation for mobility, and the DC motors, controlled by an H-bridge motor driver, allowed for smooth and precise speed adjustments. The Raspberry Pi 5 processed video input from the Pi Camera, successfully detecting lanes and obstacles with OpenCV and YOLO. The external power supply provided steady performance, minimizing power fluctuations that could impair motor control or picture processing. During testing, the system displayed consistent lane tracking and obstacle response, with the motors dynamically altering speed based on lane positions and halting as needed. Overall, the hard-

ware implementation established a strong and efficient platform for real-time autonomous driving, resulting in accurate lane-following and quick obstacle avoidance. As can be seen in the images

Figure 3 shows the resulting signals.



(a) (b) (c) (d) (e)

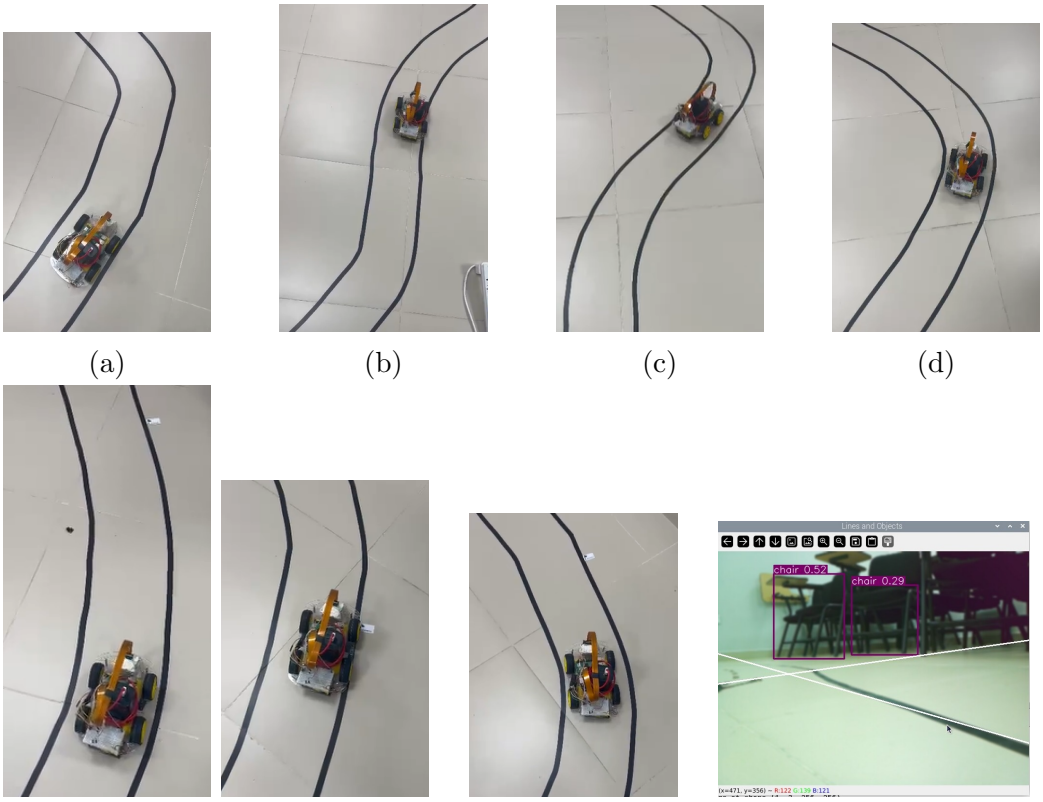
Figure 3: Hardware Implementation

3.2.2 Software implementation

Our lane-following autonomous vehicle's software successfully interpreted real-time video input to detect lanes and avoid obstacles. Using OpenCV, the system effectively detected lane markings via edge detection and Hough Transform, allowing for exact lane centering. The YOLO object detection model successfully spotted obstacles allowing the car to stop at the right distance. PID controllers helped to stabilize movement through adjusting motor speeds in response to lane location and object closeness. The integration of

these components enabled smooth navigation with little lane changing and rapid obstacle response. Despite initial problems such as noisy lane detection and real-time processing delays, improvements in image filtering and processing speed improved overall performance. The software ran smoothly on the Raspberry Pi 5, demonstrating the power of embedded systems to handle real-time vision-based movement. The results proved our algorithm's usefulness in producing accurate, responsive, and efficient autonomous driving behavior.

Figure 4 shows the outcome.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4: Software implementation

3.3 Analysis and Interpretation of Data

The project successfully showed real-time lane detection and obstacle avoidance on an embedded system by combining computer vision with PID control. OpenCV's edge detection and Hough Transform correctly identified lane markings in ideal settings, but struggled in low-light or faded lane scenarios, indicating the necessity for adaptive thresholding or sensor fusion. YOLO-based obstacle detection performed well for bigger objects but had inconsistencies with smaller or remote obstacles, indicating future improvements through dataset expansion or depth estimates. The PID controllers effectively adjusted motor speeds, but modest overcorrections in steep corners indicate that dynamic tuning could improve stability. Processing delays on the Raspberry Pi 5, particularly during high-resolution video analysis, demonstrate the need for hardware acceleration in future versions. Considering these constraints, the system reached its goals, demonstrating that cost-effective embedded solutions can achieve reliable autonomous navigation, however additional improvements are required to improve environmental adaptation and real-time performance.

4 Conclusion

4.1 Summary

This project successfully designed and implemented an RPi-based lane-assist autonomous vehicle that can track lanes in real time and avoid obstacles. By combining OpenCV for lane detection, YOLO for object recognition, and PID control for adaptive motor changes, the system displayed precise navigation and prompt stopping. The findings demonstrate the efficacy of integrating classical edge detection with deep learning for balancing accuracy and processing economy in embedded systems. While the system performed admirably under controlled conditions, issues such as fluctuating lighting, lane irregularities, and processing limits highlight opportunities for improvement. To boost robustness, future developments may incorporate adaptive thresholding, sensor fusion, and dynamic PID tuning. Despite these restrictions, this experiment produced useful information about real-time vision-based navigation and embedded robots.

4.2 Future Improvements and Takeaways

This project showed the difficulties and trade-offs of real-time lane detection and obstacle avoidance while balancing accuracy, efficiency, and cost. We discovered that combining classical edge detection with deep learning enhances dependability while maintaining acceptable computing needs. Future enhancements may include adaptive thresholding for improved lane detection, sensor fusion with LiDAR or ultrasonic sensors for more precise obstacle detection, and dynamic PID tuning for smoother control. Overall, this project provided important hands-on experience with autonomous navigation, as well as the integration of hardware and software for real-world robotics and transportation applications.

4.3 Lessons Learned

This project expanded our knowledge of computer vision, real-time object detection, and motion control with Raspberry Pi. Implementing and tuning PID controllers for lane-keeping and obstacle detection taught us how to fine-tune motor settings for smoother navigation. We also learned how to optimize YOLO-based object detection for embedded systems and process live video feeds with Picamera2 and OpenCV. Early challenges with noisy lane detection compelled us to improve edge detection and investigate adaptive thresholding. Debugging motor control increased the complexities of autonomous navigation. Overall, this project provided invaluable practical experience in real-time vision-based control and embedded robotics.

4.4 Team Dynamics

- Who was the team leader? What evidence of good leadership can you provide? Omar was the team leader, and he efficiently delegated tasks, fulfilled deadlines, and addressed difficulties cooperatively. His advice helped achieve project milestones, including installing fully autonomous vehicles.

- How did you create a collaborative and inclusive environment? Collaboration was achieved through regular meetings in the lab and library, with communication maintained via WhatsApp and MS Teams.All members participated actively, contributing to discussions and task execution. Progress was tracked through updates and discussions

- What goals did you set for your team? the team aimed to: 1. To develop and test a fully autonomous car . 2. Implement and validate the model on a RASPI 5. 3. Implement lane tracking and object detection algorithms

- How did you plan the tasks? Did you use a Gantt Chart. Did you track your progress and update your tasks? Yes a Gantt chart was created to ensure a manageable schedule , as seen in Figure 5

- Did you meet your objectives Yes, we were able to develop a fully autonomous vehicle that detect obstacles in it's way and stop before colliding with it.In addition,lane following was also implemented to keep the car in it's track.

A Gantt Chart: a figure indicating the tasks, their dependencies, their required efforts, and who's responsible for them.

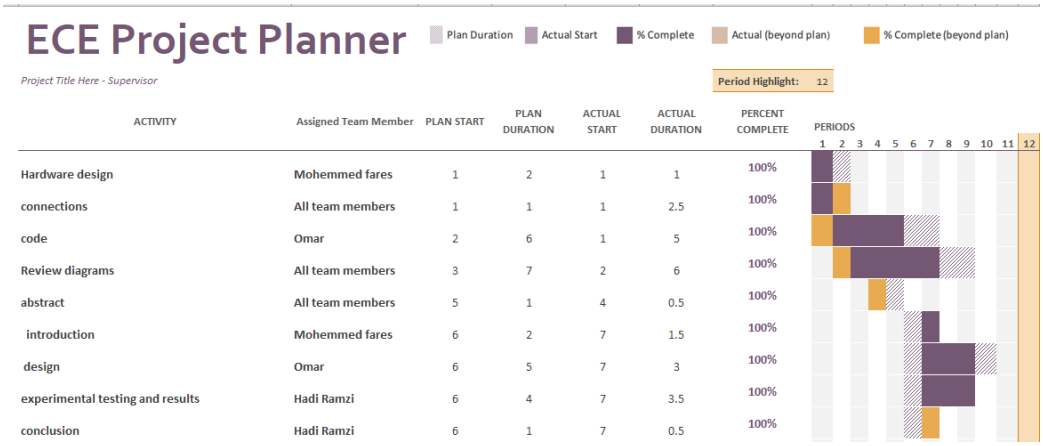


Figure 5: Gantt Chart.

Environmental Impact Analysis								
Impact of your project	**Nature**	**Extent**	**Timing**	**Severity**	**Duration**	**Reversibility**	**Uncertainty**	**Significance**
The climate Example: *Does the project affect the emission of greenhouse gases into the atmosphere?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Use of Energy Example: *Does your project affect the energy consumption of the economy? How?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Air quality Example: *Does the project have an effect on emissions of harmful air pollutants that might affect human health, damage crops or buildings or lead to deterioration in the environment (soil or rivers)?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Biodiversity, flora, fauna and landscapes Example: *Does it affect endangered species, their habitats or ecologically-sensitive areas?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							

Water quality and resources Example: *Does the project decrease or increase the quality or quantity of freshwater and groundwater?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Renewable or non-renewable resources Example: *Does the project reduce or increase use of non-renewable resources?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Sustainability Example: Does the option lead to more sustainable production and consumption? How?	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Waste production/generation/recycling Example: *Does the project affect waste production (solid, urban, agricultural, industrial, mining, radioactive or toxic waste) or how waste is treated, disposed of or recycled?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							

Economic Impact Analysis								
Impact of your project	**Nature**	**Extent**	**Timing**	**Severity**	**Duration**	**Reversibility**	**Uncertainty**	**Significance**
Economic Prosperity Example: *Does the project affect the GDP/capita, employment rate, household savings?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Investment Flows Example: *Does your project affect the flow of investment from outside the country? Does it encourage local investment in it?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, because of the nature of this project, many people who are interested in building a self driving car,this project might useful							
Public Budgets or Services Example: *Does the project affect the budgets of hospitals, community services, older people services, transport services, service quality, schools, policing, municipality services..etc?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, many car companies that are interseted in self driving cars might want to invest in this project and build it in a larger scale.							
Market Mechanisms Example: *Does it affect the private sector business opportunities? Help companies reach more costumers? Change how business is done?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, car dealerships might want to add these types of cars in their collection.							

Innovation, Research and Development Example: *Does the project have commercialization potential, lead to a potential patent? Does it allow others to innovate/research through it?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, many engineers might develop it and make it more efficient,by improving the range and the quality of the components.							
Sustainable Consumption and Production Example: Does the project produce a sustainably consumed product or service? Can it be produced sustainably?	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes,this car can be implemented as electric car , so it can reduce pollution							

Social Impact Analysis								
Impact of your project	**Nature**	**Extent**	**Timing**	**Severity**	**Duration**	**Reversibility**	**Uncertainty**	**Significance**
Health and Longevity Example: *Does the project impact health and longevity? Does it affect physical activity, nutrition, chronic diseases, accidental injuries, independent living, mental wellbeing?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes,it can be produced to be very precise and accurate so it may avoid human errors and save lives.							
Safety Example: *Does your project affect safety of social environment, protection of older people against abuse, protection against risks, response to emergency cases, feelings of safety, physical safety?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: Yes, it can be made to be extremely exact and accurate, perhaps avoiding human error and saving lives..							
Productive and Valued Activities Example: *Does the project increase leisure time, reduce stress, lead to positive behavior, increase productivity?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, if the system is reliable enough the driver can do other tasks while the car is moving.							

Standard of Living Example: *Does it affect the quality of life? Make lives easier? Reduce poverty and deprivation? Increase life choices and opportunities?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: yes, it can make the quality of life better by helping people who don't know how to or scared of driving have a better and easy life.							
Education/Life-long Learning Example: *Does the project affect literacy, use of ICT, chances of higher education, quality of education, life-long learning? Improve attainment of learning outcomes?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Quality of Social Interaction Example: *Does the project affect social connectedness, social participation, volunteering?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							

Privacy and Personal Data Example: *Does the project reveal the user identities? Create potential private data leaks or identity theft?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							
Social Reasonability Example: *Does the project affect access to products and services for people of determination? Does it affect their integration into society? Does it affect their participation in the economy? Does it address their needs?*	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Unimportant
	Justification/Explanation: NO,our project focused on building fully autonomous car that stay in its lane and stop if there is any object in its way							

References

[1] M. Pagale, R. Sharma, and A. Thakare, "A review for autonomous vehicles technologies¡? show [aq id= gq6]?¿," *Multiagent and Grid Systems*, p. 15741702241296428, 2025.

[2] M. Dika, M. Owayjan, and R. Achkar, "Steering control of self-driving car using super-twisting sliding mode control," in *2023 Fifth international conference on advances in computational tools for engineering applications (ACTEA)*. IEEE, 2023, pp. 190–196.