EC312 Digital Image Processing

Real-time Image Analysis for Self Driving Capabilities

Submitted To

Dr. Usman Akram

Dr. Asad Mansoor Khan

Submitted By

Shaheer Mukhtiar

Ulfat Zahra

Ali Waris

Eesha Arjumand

Department of Computer and Software Engineering

College of Electrical and Mechanical Engineering (CEME)

National University of Sciences and Technology (NUST)

Abstract

Advancements in artificial intelligence have greatly accelerated the development of self-driving cars, especially through the use of deep learning. However, classical image processing techniques can still provide effective and reliable solutions, particularly in constrained hardware environments. This project demonstrates a rule-based self-driving car algorithm implemented on a Raspberry Pi 5 using a Pi Camera. The system uses traditional computer vision methods such as Canny edge detection, Hough line transforms, and contour analysis to detect lanes, identify obstacles, and determine directional commands. The design divides each video frame into modular regions to perform specific tasks: lane detection, obstacle avoidance, and immediate stopping. Without relying on machine learning or external hardware control, the project achieves pseudo real-time performance, showcasing that cost-effective and efficient autonomous navigation is possible using conventional techniques on low-power platforms.

Keywords: Self-driving car, Raspberry Pi 5, Pi Camera, OpenCV, lane detection, object detection, image processing, real-time simulation

Contents

1 Introduction **v**

1.1 Background v

1.2 Purpose and Objectives v

1.3 Scope v

2 System Requirements **vi**

2.1 Functional Requirements vi

2.2 Non-Functional Requirements vi

3 System Design **vii**

3.1 Architecture Overview vii

3.2 Flow Diagram Placeholder vii

3.3 Direction Logic vii

4 Implementation **ix**

4.1 Technology Stack ix

4.2 Code Structure ix

4.3 Outputs ix

5 Testing and Results **xi**

5.1 Test Setup xi

5.2 Performance Summary xii

5.3 Links to Resources xii

6 Conclusion and Future Work **xiii**

List of Figures

3.1 flow diagram viii

4.1 Road x

4.2 Segmented x

4.3 obstacle x

4.4 stopping at detection x

5.1 Hardware Setup for Testing xi

Chapter 1

Introduction

1.1 Background

Autonomous vehicles use sensors and computer vision to understand their environment. In this beginner project, a simulated self-driving system is built using only OpenCV to mimic a car's decisions by analyzing a video feed and printing movement commands.

1.2 Purpose and Objectives

- Detect lane lines from a video frame.
- Simulate navigation direction using visual feedback.
- Identify and avoid objects using simple logic.
- Trigger emergency stop if an obstacle is too close.

1.3 Scope

The project simulates decision-making using only camera input. It does not control motors or use machine learning. It is implemented in Python using OpenCV on a Raspberry Pi 5 platform with Pi Camera.

Chapter 2

System Requirements

2.1 Functional Requirements

- Lane detection using Hough Transform.
- Object detection using contour analysis.
- Direction simulation logic (left/right/straight/stop).
- Real-time display of processed frames.

2.2 Non-Functional Requirements

- **Performance:** Should process frames in real-time (~20-30 FPS).
- **Portability:** Runs on Raspberry Pi or any OpenCV-supported system.
- **Usability:** Beginner-friendly and modular design.

Chapter 3

System Design

3.1 Architecture Overview

The system divides the input frame into three regions:

1. Bottom horizontal window – detects lane lines.
2. Middle square window – detects objects on left or right.
3. Top center window – detects immediate obstacles.

Each window uses tailored logic to perform its specific task.

3.2 Flow Diagram Placeholder

3.3 Direction Logic

If both lane lines are visible, the car steers based on their slope. If only one line is detected, extrapolation is used. Object detection overrides this logic to turn away or stop if needed.

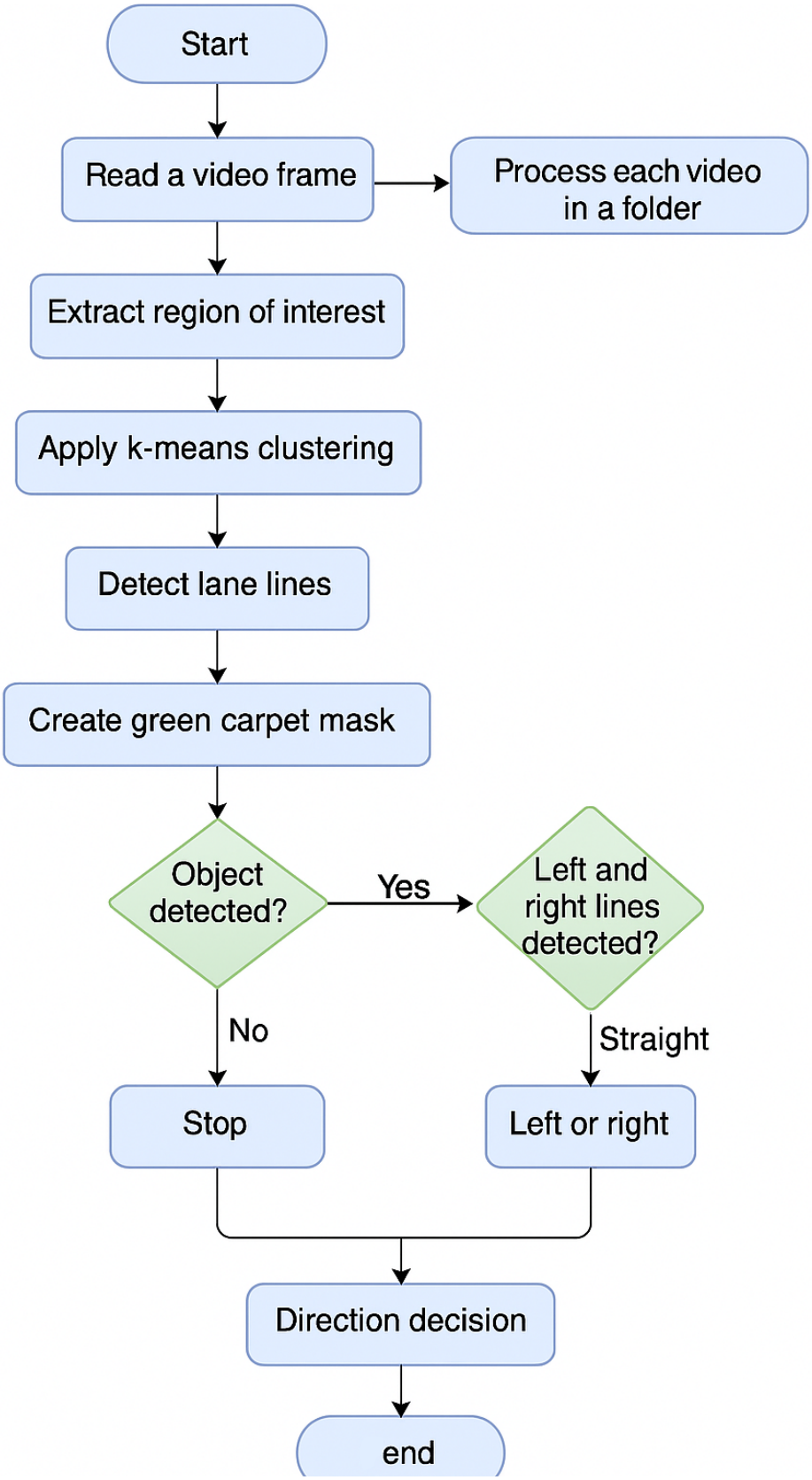

Figure 3.1: flow diagram

Chapter 4

Implementation

4.1 Technology Stack

- **Language:** Python 3
- **Libraries:** OpenCV, NumPy, scikit-learn (for k-means)
- **Platform:** Raspberry Pi 5 (development)

4.2 Code Structure

- `process_frame()`: Core processing of each frame.
- `average_line()`, `average_history()`: Smooth lane estimates.
- `smooth_kmeans_labels()`: Stabilizes object detection.

4.3 Outputs

Figure 4.1: Road

Figure 4.2: Segmented

Figure 4.3: obstacle

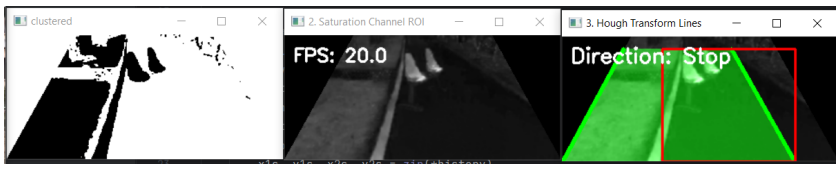

Figure 4.4: stopping at detection

Chapter 5

Testing and Results

5.1 Test Setup

Video files of roads with synthetic obstacles were processed using the proposed algorithm on both a Raspberry Pi and a laptop. Additionally, real-time lane and object detection, along with decision-making logic, was successfully implemented on a Raspberry Pi 5 using a Pi Camera.

Figure 5.1: Hardware Setup for Testing

5.2 Performance Summary

Test	Result
Lane Detection	Reliable under consistent lighting conditions
Object Detection	Effective for large black or colored blobs
Stop Mechanism	Responsive within 1–2 frames
Frame Rate	Approximately 25 FPS on Raspberry Pi 5

Table 5.1: System Performance Summary

5.3 Links to Resources

Video demonstrations and dataset resources are available online:

- **Dataset Videos:** `https://nustedupk0-my.sharepoint.com/:f:/g/personal/smukhtar_ce44ceme_student_nust_edu_pk/Ei-juiA7s2BAnnW3M0xwTHkBURRCChKtVwngUZdNQ0`
  `e=RRC3s9`
- **Demo Videos:** `https://nustedupk0-my.sharepoint.com/:f:/g/personal/awaris_ce44ceme_student_nust_edu_pk/EmAo1ZuLlklGpmXfs9oeFrkBrpc6AWI9MCA-_fRB3fU7Yg?e=8wvAM1`
- **GitHub Repository:** `https://github.com/aliwariskhoso123/Real-time-Image-Analysis-`

Chapter 6

Conclusion and Future Work

This project introduces the fundamentals of vision-based autonomous navigation using classical image processing techniques. A rule-based system was implemented to handle lane detection, direction control, and obstacle handling in real-time using a Raspberry Pi 5 and Pi Camera. The system performed reliably in controlled conditions, validating the feasibility of classical methods for embedded autonomous systems.

Future improvements include:

- Integration with motor control for physical robot actuation.
- Implementation of AI models (e.g., YOLO) for improved object classification.
- Adaptive ROI scaling and lighting compensation for robustness in variable environments.

References

[1] Bradski, G., Kaehler, A. *Learning OpenCV 3*, O'Reilly Media, 2016.

[2] Szeliski, R. *Computer Vision: Algorithms and Applications*, Springer, 2010.

[3] OpenCV Documentation: `https://docs.opencv.org`

[4] Raspberry Pi Foundation: `https://www.raspberrypi.org`