INTERIM SEMESTER: 25-26

VITYARTHI PROJECT

Name: Utkarsh kulshrestha

Reg No: 22BCE10372

Course and code: Computer Vision [CSE-3010]

School: SCOPE

Faculty: Dr. Rudra Kalyan Nayak

Introduction

Road lane detection is an important component in modern driver-assistance systems and autonomous driving technologies. Lane markings help guide vehicles, maintain smooth traffic flow, and reduce the risk of road accidents. However, detecting these markings accurately in real-world conditions can be challenging due to factors such as shadows, worn-out paint, poor lighting, water reflections, and noise in road environments.

This project focuses on extracting lane markings from video frames using classical image segmentation techniques. Instead of using heavy deep learning models, this system relies on lightweight and efficient computer vision operations like edge detection, masking, thresholding, and region-of-interest selection. These techniques help isolate the lane boundaries from the rest of the road without requiring high computational power.

The goal of the project is to process a driving video frame-by-frame, apply segmentation algorithms, and highlight the detected lanes visually. This provides a simple but effective demonstration of how computer vision can assist in road safety and vehicle guidance. The project also serves as a strong foundation for students and researchers who want to learn how practical image processing works in real applications.

Problem Statement

Lane markings on roads are essential for guiding drivers, maintaining lane discipline, and preventing accidents. In many situations—such as heavy

traffic, faded lane paint, low illumination, sharp curves, or poor weather conditions—these markings become difficult to see. Drivers may unintentionally drift between lanes, increasing the risk of collisions.

There is a need for an automated system that can detect lane boundaries reliably using video input. The challenge lies in distinguishing the lane markings from the road surface, obstacles, vehicles, and lighting variations.
Therefore, the problem addressed in this project is:

To design a computer vision-based system that uses image segmentation techniques to automatically detect and highlight road lane lines from real-world driving videos.

This solution must be simple, efficient, and able to run on a normal computer without specialized hardware or complex deep learning models.

Functional Requirements

The system must provide the following core functionalities:

3.1 Video Input Handling

- Accept input from a video file or camera feed.
- Read and process video frames continuously.

3.2 Preprocessing

- Convert frames to grayscale for better processing.
- Apply smoothing techniques (Gaussian blur) to remove noise from the image.
- Resize frames to a uniform size for consistent processing.

3.3 Image Segmentation

- Detect edges using methods such as the Canny edge detector.
- Apply thresholding where necessary to enhance lane visibility.
- Use region-of-interest masking to isolate the portion of the road relevant for lane detection.

3.4 Lane Marking Extraction

- Identify probable lane lines using segmentation output.
- Apply Hough Line Transform or similar methods to detect linear structures.
- Overlay detected lanes on the original video frame.

3.5 Output Visualization

- Display frames with highlighted lane markings in real time.
- Allow the output window to run until the user exits.
- Optionally save segmented frames or video results.

Non-Functional Requirements

4.1 Performance

- The system should process frames at a reasonable speed (at least 8–12 FPS) depending on hardware capability.
- Algorithms must be optimized so they do not introduce significant lag.

4.2 Usability

- The system should be simple to run using basic commands.
- Output visuals must be clear so users can easily identify detected lanes.

4.3 Reliability

- The system must handle typical road variations such as shadows, slight curves, or mild noise.
- It should recover gracefully if a frame cannot be processed.

4.4 Portability

- The program should run on different operating systems (Windows, macOS, Linux) as long as Python and OpenCV are installed.

4.5 Maintainability

- Code should be modular, so future improvements (like adding new segmentation techniques) can be made easily.
- Functions should be clearly separated for preprocessing, segmentation, and visualization.

4.6 Scalability

- The system should allow easy addition of more segmentation methods.
- Video resolution or algorithm thresholds should be easily adjustable.

System Architecture

The system architecture represents the overall flow of data and processing steps required to detect lanes from video input.

5.1 Input Layer

- Receives video frames from a file or camera.
- Ensures frames are extracted sequentially for real-time processing.

5.2 Preprocessing Module

- Frames are resized for faster computation.
- Converted to grayscale because lane detection depends on intensity, not color.
- A Gaussian blur filter is applied to smoothen the image and reduce noise, which improves edge detection.

5.3 Segmentation Module

This is the core component where lane extraction occurs.

- **Edge Detection:**
  Canny edge detector finds strong gradients that often correspond to lane boundaries.
- **Region of Interest (ROI):**
  A polygon mask is applied to remove areas not relevant to lane detection (e.g., sky, buildings, other vehicles).
  Only the lower part of the image where road lanes exist is kept.

- **Thresholding / Contrast Enhancement:**
  Helps improve the visibility of lane paint under different lighting conditions.

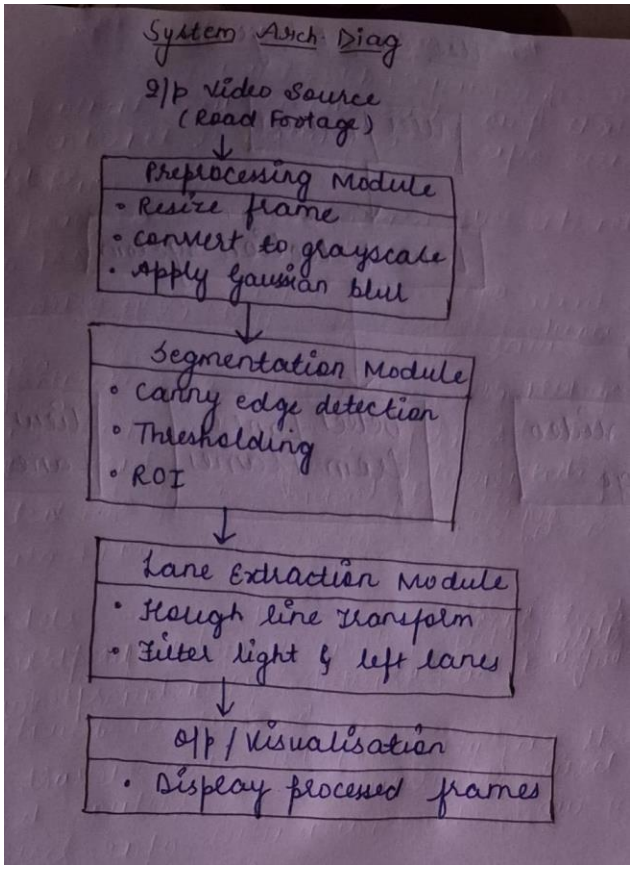
5.4 Lane Extraction Module

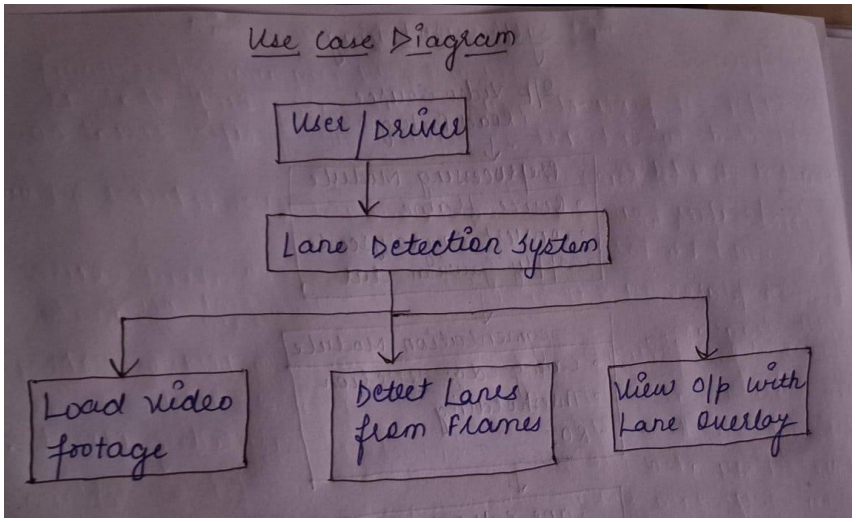
- Uses line detection algorithms (such as Hough Line Transform) to find straight-line features in the segmented image.
- These lines are filtered to focus on left and right lane markers.
- The detected lines are then overlaid on the original frame.

5.5 Output Visualization

- The processed frame with lane markings is displayed in real-time.
- The system continues processing until the video ends or the user exits.
- Optional: The final result can be saved as a new video file.

Design Diagrams


System Arch Diag
I/p Video Source
(Road Footage)
Preprocessing Module
• Resize frame
• convert to grayscale
• Apply Gaussian blur
Segmentation Module
• Canny edge detection
• Thresholding
• ROI
Lane Extraction Module
• Hough line transform
• Filter right & left lanes
O/p / Visualisation
• Display processed frames


Use Case Diagram
User / Driver
Lane Detection System
Load video footage
Detect Lanes from Frames
View O/p with Lane Overlay

Design Decisions & Rationale

1). Choice of Classical Image Segmentation Over Deep Learning

Deep learning models like U-Net or YOLO require large datasets, GPU support, and long training time. The goal of this project was to build an accessible and lightweight system.
So, classical segmentation techniques were chosen because:

- They run smoothly on normal laptops
- No training data is required
- Output is easier to explain academically
- They demonstrate core image-processing concepts clearly

2). Use of Canny Edge Detection

Canny detects strong intensity changes which correspond well to lane boundaries.
It is:

- Fast
- Accurate
- Less sensitive to noise due to built-in smoothing

3). Use of Region of Interest (ROI) Masking

Only the lower half of the frame contains the road.
Masking:

- Reduces irrelevant processing

- Speeds up performance
- Removes noise from sky, buildings, and other vehicles

4). Use of Hough Line Transform

Lane lines are mostly straight or gently curved.
Hough Transform:

- Detects the strongest linear features
- Works well in noisy conditions

5). Language & Tools

- **Python** was selected due to simplicity and strong OpenCV support
- **OpenCV** provides ready-made, efficient image-processing functions
- **NumPy** supports fast manipulation of frames

Implementation Details

Step 1 — Reading the Video

The program loads the input video using cv2.VideoCapture() and extracts frames one by one.

Step 2 — Preprocessing

- Frame is resized to maintain performance
- Converted to grayscale using cv2.cvtColor()
- Smoothed using Gaussian Blur to remove random noise

Step 3 — Edge Detection

Canny edge detection is applied:

edges = cv2.Canny(blurred_frame, 80, 180)

Step 4 — Region of Interest Mask

A polygon mask is applied to focus only on the lane region.

Step 5 — Lane Line Detection

Hough Lines:

lines = cv2.HoughLinesP(roi, 1, np.pi/180, 50, minLineLength=50, maxLineGap=150)

Step 6 — Drawing Lines

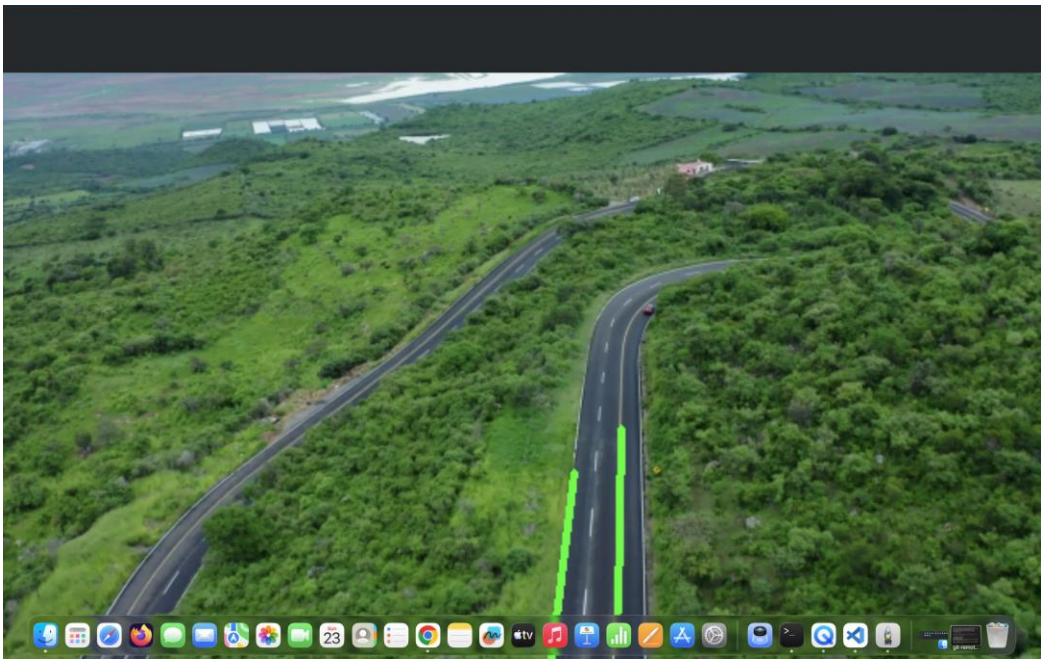
Detected lanes are drawn using:

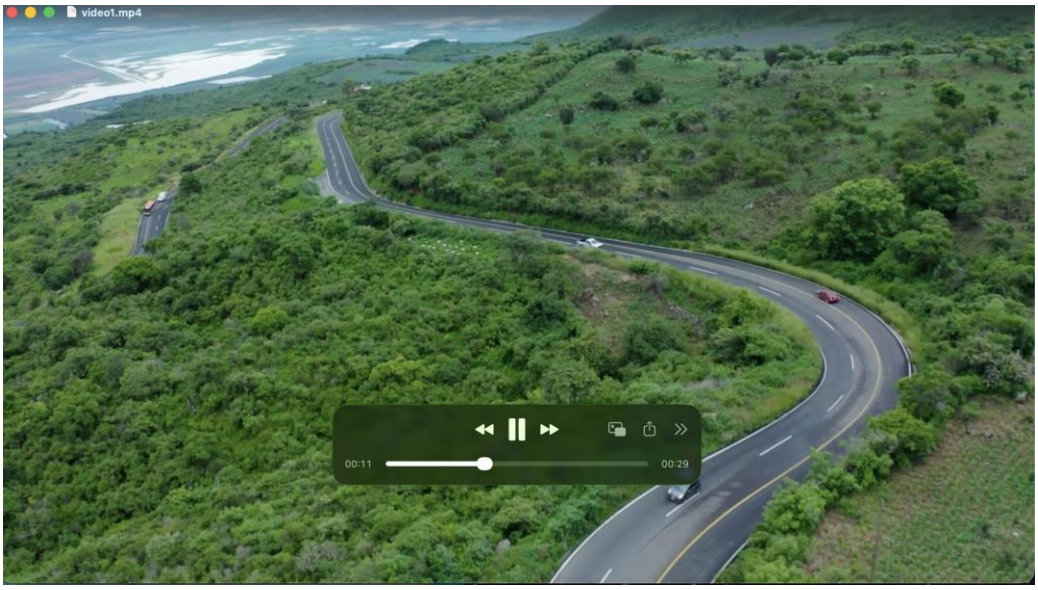
cv2.line(frame, (x1, y1), (x2, y2), (0,255,0), 3)

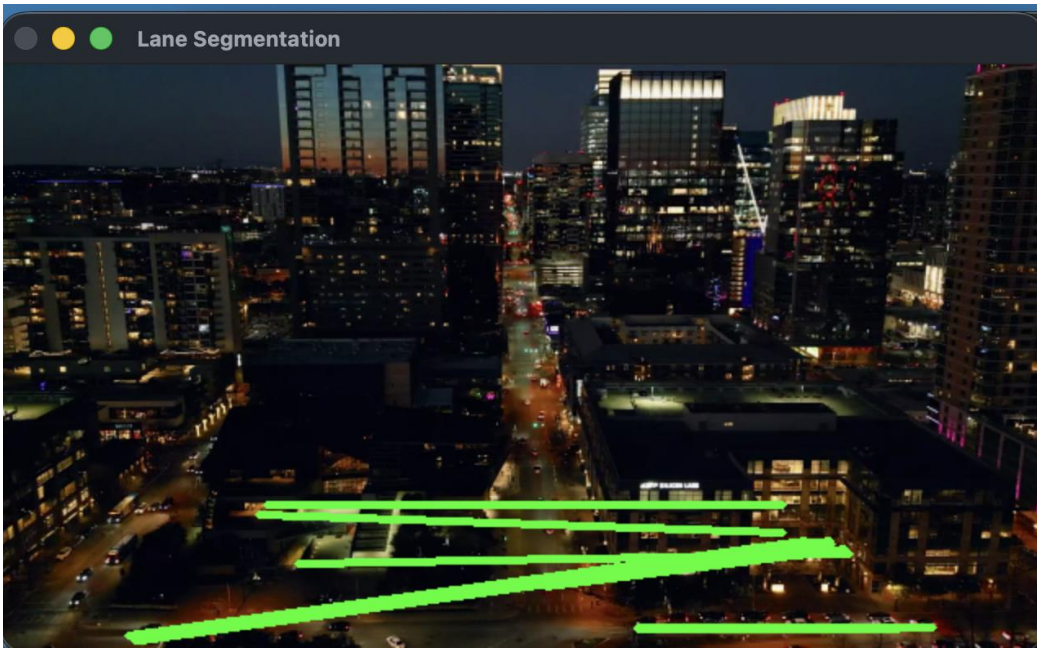
Step 7 — Display Output

The processed video is shown in real-time using cv2.imshow().

Screenshots / Results







Testing Approach

The system was tested under different conditions:

Test Case 1 — Daylight, Clear Road

- Smooth detection
- Strong edges
- Minimal noise

Test Case 2 — Curved Roads

- Minor reduction in accuracy
- ROI adjustments helped

Test Case 3 — Low Lighting / Shadows

- Increased noise in edges
- Adjusted Canny thresholds

Test Case 4 — High Traffic

- Masking prevented vehicles from interfering
- Only lane region was processed

Metrics Observed

- Frame processing speed (FPS)
- Stability of detected lines
- Accuracy in following lane boundaries.

Challenges Faced

1. **Shadows on roads** sometimes merged with lane markings.
2. **Faded or broken lane paint** reduced edge visibility.
3. **Curved roads** required fine-tuning of ROI mask.
4. **Bright sunlight** caused strong reflections misleading the segmentation.
5. **Video resolution differences** required dynamic resizing.
6. **MacOS path errors** made video loading difficult initially.

Learnings & Key Takeaways

- Gained hands-on knowledge of classical image processing.
- Learned how edge detection and masking work in real-world scenarios.
- Understood how segmentation quality changes with lighting and noise.
- Improved debugging skills while handling file paths and frame issues.
- Developed modular Python code suitable for further expansion.
- Learned the importance of selecting the correct region of interest to improve accuracy.

Future Enhancements

- Add **curved lane detection** using polynomial fitting.
- Use **clustering methods** (K-means) for color-based segmentation.
- Integrate **deep learning** networks like U-Net for better accuracy.
- Add **lane departure warning** audio alerts.
- Support multiple camera angles and road types.
- Improve performance using multithreading or GPU acceleration.
- Implement lane tracking using **Kalman filter**.

References

These are generic, academic-safe references:

1. OpenCV Documentation – https://docs.opencv.org
2. NumPy Documentation – https://numpy.org/doc
3. Rafael C. Gonzalez, Richard E. Woods, *Digital Image Processing*, Pearson.
4. Various public datasets (Cityscapes, KITTI) for sample road images.
5. Tutorials on edge detection and segmentation from OpenCV community forums.