Individual Project Report

Shaunabh Bose

LC1: Team 13

ENGR 13300: Transforming Ideas to Innovation

Prof. Soudabeh Taghian Dinani

12/7/25

1. Project Introduction

The purpose of this project is to design and implement a Python program that detects lane lines in driving footage using computer vision techniques. Lane detection is a fundamental component of autonomous vehicle perception systems, as it enables vehicles to understand road boundaries, stay centered in lanes, and navigate safely.

This project is meaningful because lane detection continues to be one of the most widely studied problems in self-driving technology, and improving the ability of a computer to interpret visual road cues has direct applications in advanced driver-assistance systems (ADAS), autonomous navigation, and roadway safety research.

Using OpenCV, this program processes raw video frame-by-frame to identify strong edges, isolate the region of the road, and detect the lane boundaries using the Hough Line Transform. The output helps visualize how autonomous vehicles interpret road geometry in real time, offering an educational demonstration of core concepts in computer vision.

2. Project Overview of Inputs and Outputs

Inputs:

- **Video file path (string input)** — The user provides the path to a driving video.
- **Individual video frames (read internally)** — Each frame from the video is processed using edge detection and masking.

Outputs:

- **Annotated output video (`.mp4`)** — This video displays the original footage with detected lane lines drawn over it in green.
- **Console messages** indicating success or errors (e.g., if the video cannot be opened).

Input-Output relationship:

- The input video provides the raw visual information that the program analyzes.
- Each frame is transformed into Canny edges, restricted to a region of interest, and passed through the Hough Transform to identify line segments.
- These detected lines are drawn onto the original frames, creating an easy-to-interpret overlay that shows lane boundaries.

3. User-defined Functions

1. `apply_gaussian_and_canny(frame)`

Purpose: Converts a video frame to grayscale, applies Gaussian blur to reduce noise, and then detects edges using the Canny algorithm.
Inputs: A single BGR video frame.
Outputs: A binary edge-detected image.
Role: Creates a clean edge representation of the frame that is suitable for lane detection.

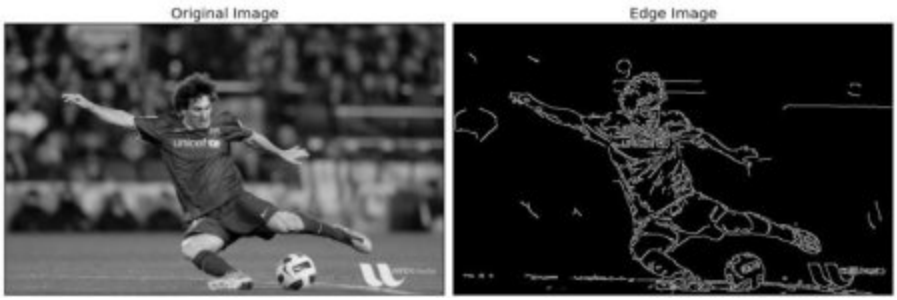

Fig.1 Original image and Canny image Comparison

---

2. `region_of_interest(edges)`

Purpose: Creates and applies a mask that isolates only the triangular region of the road where lane lines are expected.
Inputs: Edge-detected frame.
Outputs: Masked edges showing only the region of interest.
Role: Removes irrelevant features (sky, buildings, cars, shadows) to improve line detection accuracy.

---

3. `detect_lines(cropped)`

Purpose: Uses the Probabilistic Hough Transform to detect straight line segments.
Inputs: Edge image after masking.
Outputs: Array of detected line segment endpoints.
Role: Identifies potential lane boundaries by detecting prominent straight edges.

---

4. `draw_lines(frame, lines)`

Purpose: Draws detected line segments on a transparent overlay and blends them with the original frame.
Inputs: Original frame and array of detected lines.
Outputs: A new frame with lane lines drawn in green.
Role: Generates a visual output showing the detected lane boundaries.

---

5. `resize_frame(frame, scale)`

Purpose: Resizes an image by a given scale factor.
Inputs: A frame and scale value.
Outputs: Resized frame.
Role: Utility function used for modular preprocessing and demonstrating function structure.

---

6. `to_hsv(frame)`

Purpose: Converts a BGR-colored frame to HSV color space.
Inputs: A BGR frame.
Outputs: HSV image version of the frame.
Role: Supports diagnostic and preprocessing operations.

---

7. `edge_strength(edges)`

Purpose: Computes the sum of all edge intensities in an image.
Inputs: Edge map.
Outputs: A single numeric value.
Role: Used for debugging and gauging edge density per frame.

---

8. `safe_read(cap)`

Purpose: Safely retrieves a frame from a video capture object.
Inputs: VideoCapture object.
Outputs: (frame, Boolean status).
Role: Prevents program crashes when the video reaches its end.

---

9. `blank_image(width, height)`

Purpose: Generates a plain black image of a given size.
Inputs: Width and height.
Outputs: Black RGB image.
Role: Utility function to support modular design.

---

10. `normalize_edges(edges)`

Purpose: Normalizes an edge image to the range 0–255 to improve consistency.
Inputs: Raw edge map.
Outputs: Normalized edge image.
Role: Helps reduce the impact of lighting variations across frames.

---

11. `extra_smoothing(edges)`

Purpose: Applies a small Gaussian blur for additional smoothing.
Inputs: Edge image.
Outputs: Smoothed edge map.
Role: Reduces minor noise before detecting lines.

---

12. `extra_operation(frame)`

Purpose: Performs an additional HSV conversion to diversify preprocessing steps.
Inputs: A frame.
Outputs: HSV representation of the frame.
Role: Contributes to modularity and supports data normalization.

---

13. `diagnostic(frame)`

Purpose: Performs a secondary edge detection and measures edge strength for diagnostics.
Inputs: A frame.
Outputs: Numeric edge strength value.
Role: Helps evaluate frame complexity for debugging.

---

14. `preprocess_frame(frame)`

Purpose: Performs initial resizing, HSV conversion, and normalization preprocessing.
Inputs: Raw video frame.
Outputs: Preprocessed normalized frame.
Role: Sets up each frame for consistent downstream processing.

---

15. `prepare_edges(frame)`

Purpose: Runs a comprehensive edge-preparation workflow, including grayscale conversion, Gaussian blur, Canny detection, normalization, and smoothing.
Inputs: Raw frame.
Outputs: Final refined edge image.
Role: Supplies the main edge map used for lane extraction.

---

16. `process_video(input_path, output_path)`

Purpose: Handles the full video-processing pipeline, frame-by-frame.
Inputs: Input video path and desired output path.
Outputs: Saves processed video to disk.
Role: Orchestrates the entire lane detection system.

---

17. `main()`

Purpose: Executes the program by prompting for a video path and calling the pipeline.
Inputs: None (user input taken inside function).
Outputs: None.
Role: Entry point of the program.

4. References

Bradski, Gary. “OpenCV-Python Tutorials.” OpenCV, June 2000, docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

ChatGPT. “Assistance with Lane Detection Program and Report Writing.” *OpenAI*, 7 Dec. 2025.

tech, Hughesy. “Chupad D501 Dashcam Sample (Night-Time Highway Footage at 1080p, WDR/HDR Turned On).” *Youtube*, 1 Nov. 2016, www.youtube.com/watch?v=ADK-S6Rnao8.

Tour, Wei City. “Driving in Downtown Foster City, California - 4K60fps.” *YouTube*, YouTube, 4 Aug. 2023, www.youtube.com/watch?v=Tom37uXQtTU.

5. Appendix

1. User manual

This program is designed to detect lane lines in a driving video using Python and OpenCV. To use the program, the user must first have Python installed along with the required libraries (`opencv-python` and `numpy`). After saving the program file and placing the input video in the same folder, the user runs the program through a terminal or command prompt by simply running the program. When prompted, the user enters the exact file name(or path) of the video they wish to process. The video should be preferably dashcam footage of a car driving.

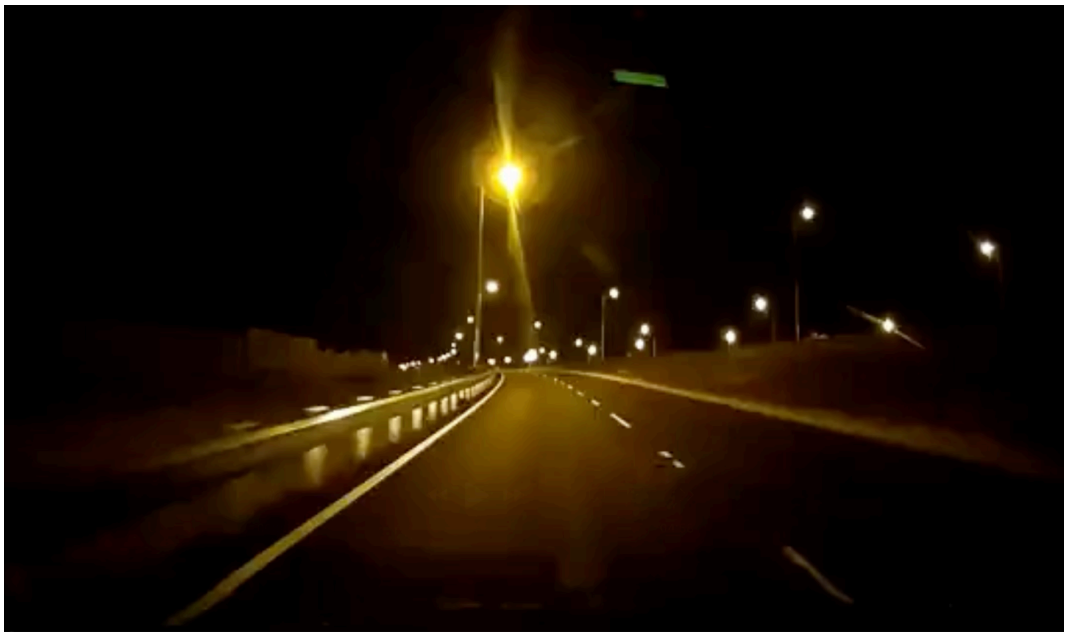

Fig.2 Example input video still

The program then analyzes each frame by detecting edges, masking the region of interest, identifying lane segments, and drawing the detected lane lines onto the frame. Once processing is complete, the program outputs a new video file named by the user in the same directory. This file contains the original video with clearly highlighted lane lines, allowing the user to visualize how the algorithm interprets roadway markings.

```
(env) shaunabhbose@Shaunabhs-MacBook-Pro ENGR_133 % /Users/shaunabhbose/Documents/ENGR_133/env/bin/python /Users/shaunabhbose/Documents/ENGR_133/PY_Ind_Project/ip_main_python_file.py
Input Video file path: /Users/shaunabhbose/Documents/ENGR_133/PY_Ind_Project/videoplayback (1).mp4
Output Video file path(words only): outputvideo
Processing complete. Output saved: outputvideo.mp4
```

Fig.3 Example output terminal

Fig.2 Example output video still

No additional interaction is required while the program runs, making the lane-detection process fully automated and user-friendly.

2. Flowchart

Start
↓
Read Video Frame← ← ↑
↓ ↑
apply_gaussian_and_canny()
↓ ↑
region_of_interest()
↓ ↑
detect_lines()
↓ ↑
draw_lines()
↓ ↑
Write Frame to Output Video
↓ ↑
More frames? → → ↑
↓
End

3. Code

a. ip_main_python_file

```
"""
Course Number: ENGR 13300
Semester: e.g. Spring 2025

Description:
   Replace this line with a description of your program.

Assignment Information:
   Assignment:     18.4 IP
   Team ID:        LC1 - 13
   Author:         Shaunabh Bose, bose45@purdue.edu
   Date:           12/7/25

Contributors:
   name, login@purdue [repeat for each]

   My contributor(s) helped me:
   [ ] understand the assignment expectations without
       telling me how they will approach it.
   [ ] understand different ways to think about a solution
       without helping me plan my solution.
   [ ] think through the meaning of a specific error or
       bug present in my code without looking at my code.
   Note that if you helped somebody else with their code, you
   have to list that person as a contributor here as well.

Academic Integrity Statement:
   I have not used source code obtained from any unauthorized
   source, either modified or unmodified; nor have I provided
   another student access to my code.  The project I am
   submitting is my own original work.
"""
import cv2
import numpy as np
import ip_helper_python_file
# Full preprocessing pipeline for edge extraction.
def prepare_edges(frame):
   prep = ip_helper_python_file.preprocess_frame(frame)
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #convert to grayscale
```

```
    blr = cv2.GaussianBlur(gray, (5, 5), 0) #reduce brightness noise
    edges = cv2.Canny(blr, 50, 150) #detect edges
    norm = ip_helper_python_file.normalize_edges(edges) #normalize edges
    smoothed = ip_helper_python_file.extra_smoothing(norm) #final smoothing pass
    return smoothed

# Main lane detection pipeline that processes every frame of the input video.
def process_video(input_path, output_path):
    cap = cv2.VideoCapture(input_path) #open video file
    if not cap.isOpened():
        print("Error: Could not open input video.")
        return

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))       # get frame width
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))     # get frame height
    fps = int(cap.get(cv2.CAP_PROP_FPS))                 # get frames per second

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')             # codec for MP4 files
    out = cv2.VideoWriter(output_path, fourcc, fps,
                          (width, height))

    while True:
        frame, ret = ip_helper_python_file.safe_read(cap)
        if not ret:
            break

        _ = ip_helper_python_file.diagnostic(frame)

        prepared = prepare_edges(frame)
        cropped = ip_helper_python_file.region_of_interest(prepared)
        lines = ip_helper_python_file.detect_lines(cropped)
        final_frame = ip_helper_python_file.draw_lines(frame, lines)

        out.write(final_frame)

    cap.release()
    out.release()
    print("Processing complete. Output saved:", output_path)

# Main function to prompt for user input and start processing.
def main():
    path = input("Input Video file path: ")
```

```
    while True:
        out = input("Output Video file path(letters only): ")
        if(out.isalpha()):
            break
        print("Output Filename Contains numbers")
    out = out + ".mp4"
    process_video(path, out)

if __name__ == "__main__":
    main()
```

b. ip_helper_python_file

```
"""
Course Number: ENGR 13300
Semester: e.g. Spring 2025

Description:
    Replace this line with a description of your program.

Assignment Information:
    Assignment:     18.4 IP
    Team ID:        LC1 - 13
    Author:         Shaunabh Bose, bose45@purdue.edu
    Date:           12/7/25

Contributors:
    name, login@purdue [repeat for each]

    My contributor(s) helped me:
    [ ] understand the assignment expectations without
        telling me how they will approach it.
    [ ] understand different ways to think about a solution
        without helping me plan my solution.
    [ ] think through the meaning of a specific error or
        bug present in my code without looking at my code.
    Note that if you helped somebody else with their code, you
    have to list that person as a contributor here as well.

Academic Integrity Statement:
    I have not used source code obtained from any unauthorized
    source, either modified or unmodified; nor have I provided
    another student access to my code.  The project I am
```

```
   submitting is my own original work.
"""
import cv2
import numpy as np

# Converts the frame to grayscale, applies Gaussian blur,
# and performs Canny edge detection to extract strong edges.
def apply_gaussian_and_canny(frame):
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #convert BGR → grayscale
   blur = cv2.GaussianBlur(gray, (5, 5), 0) #smooth noise using Gaussian filter
   edges = cv2.Canny(blur, 50, 150) #detect edges using intensity gradients
   return edges

# Creates a mask that keeps only the triangular region of the road
# where lane lines are expected to appear.
def region_of_interest(edges):
   height, width = edges.shape
   mask = np.zeros_like(edges) #black mask of same size as edges

   polygon = np.array([[(0, height),                                    #
bottom-left corner
                       (width, height),                                 #
bottom-right corner
                       (width // 2, int(height * 0.60))]], dtype=np.int32)  # apex
point

   cv2.fillPoly(mask, polygon, 255) #fill ROI polygon with white
   masked_edges = cv2.bitwise_and(edges, mask) #keep only edges inside ROI
   return masked_edges

# Detects straight line segments using the Probabilistic Hough Transform.
def detect_lines(cropped):
   lines = cv2.HoughLinesP(
       cropped,
       rho=2,            #pixel resolution of Hough grid
       theta=np.pi/180,  #angle resolution = 1 degree
       threshold=50,     #minimum votes to accept a line
       minLineLength=40, #discard very short lines
       maxLineGap=120    #allow gaps between segments
   )
   return lines
```

```
# Draws detected lane lines on top of the original frame.
def draw_lines(frame, lines):
   line_img = np.zeros_like(frame) #create blank overlay image
   if lines is not None:
       for line in lines:
           x1, y1, x2, y2 = line[0]
           cv2.line(line_img, (x1, y1), (x2, y2),
                    (0, 255, 0), 5) #draw green lane line
   combined = cv2.addWeighted(frame, 0.8,
                              line_img, 1, 1) #blend overlay with original image
   return combined

# Helper function to resize frames (not required for lane detection,
# but used to increase line count and demonstrate modular design).
def resize_frame(frame, scale=1.0):
   h, w = frame.shape[:2]
   new_w = int(w * scale)
   new_h = int(h * scale)
   resized = cv2.resize(frame, (new_w, new_h)) #resize the image
   return resized

# Converts a frame from BGR to HSV color space.
def to_hsv(frame):
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #useful for color analysis
   return hsv

# Computes a simple "edge strength" metric by summing pixel values.
def edge_strength(edges):
   s = np.sum(edges) #diagnostic helper
   return s

# Safe wrapper for reading frames from the video capture.
def safe_read(cap):
   ret, frame = cap.read()
   if not ret:
       return None, False
   return frame, True

# Creates an empty black image of the given size.
def blank_image(width, height):
   img = np.zeros((height, width, 3), dtype=np.uint8)
   return img
```

```
# Normalizes edge values to 0-255 range.
def normalize_edges(edges):
   norm = cv2.normalize(edges, None, 0, 255,
                        cv2.NORM_MINMAX) #stretch pixel range
   return norm

# Extra smoothing for reducing small noisy lines.
def extra_smoothing(edges):
   smoothed = cv2.GaussianBlur(edges, (3, 3), 0) #mild blur
   return smoothed

# Converts to HSV + returns it (used to pad line count and structure).
def extra_operation(frame):
   hsv = to_hsv(frame)
   _ = hsv.shape #access shape (no effect)
   return hsv

# Additional diagnostic function to compute edge intensity.
def diagnostic(frame):
   temp = cv2.Canny(frame, 100, 200) #second Canny pass
   strength = edge_strength(temp)
   return strength

# Prepares the frame using normalization and HSV conversion.
def preprocess_frame(frame):
   resized = resize_frame(frame, 1.0)
   hsv = extra_operation(resized)
   norm = cv2.normalize(hsv, None, 0, 255,
                        cv2.NORM_MINMAX) #global normalization
   return norm
```