Computer Vision & Image Understanding

Anik Barury
CSE/22017/871

August 30, 2025

Project Question

Implement Hough transform-based line detection for the following application:

Lane Departure Warning Systems
Hough line detection can be used to identify and track lane markings on a road. By detecting the lines, the system can determine if a vehicle is drifting out of its lane, triggering a warning for the driver.
Search for/prepare the dataset of appropriate images (or video and work on the frames) and apply.
Apply Canny edge detection before applying Hough transform.

Introduction

This mini-project implements a classical computer-vision pipeline that detects lane markings in images/video frames using Canny edge detection followed by the Hough transform to extract lines. The pipeline is lightweight and interpretable, suitable for demos and low-cost real-time systems.

1. Grayscale Conversion

A colored image has three channels (R,G,B). For detecting lane boundaries we mainly need brightness/contrast information, so we reduce the image to a single intensity channel (grayscale). This simplifies processing and reduces compute.

Formula:

$$I(x, y) = 0.299\,R + 0.587\,G + 0.114\,B$$

(Weights follow human-perceived brightness.)

2. Gaussian Blur

Images contain local noise (small texture, shadows, camera sensor noise). Gaussian blur smooths intensity variations so small noisy edges do not trigger false positives.

2D Gaussian kernel:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\Big(- \frac{x^2 + y^2}{2\sigma^2} \Big)$$

Convolving the image with this kernel performs the smoothing. The parameter σ controls blur strength.

3. Canny Edge Detection

Edges are places where intensity changes quickly (lane paint vs. asphalt). Canny finds such edges robustly while suppressing noise.

Key steps and formulas:

- Compute gradients using Sobel filters:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y}$$

- Gradient magnitude:

$$G = \sqrt{G_x^2 + G_y^2}$$

- Gradient direction:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

- Then non-maximum suppression, double-thresholding and edge tracking by hysteresis are applied to produce a clean edge map.

4. Hough Line Transform

Canny produces edge pixels (points). Hough transform maps every edge point into line parameters space and accumulates votes to detect straight lines.

Line equation in Hough space:

$$\rho = x \cos\theta + y \sin\theta$$

Here ρ is the perpendicular distance from the origin to the line, and θ is the angle of the normal. Peaks in (ρ, θ) space indicate lines.

Algorithm

Algorithm 1 Lane detection + lane departure warning (per-frame)

Input: RGB frame
Convert frame to grayscale: $I \leftarrow \text{Grayscale}(frame)$
Smooth: $I_s \leftarrow I * G(x, y; \sigma)$ ▷ Gaussian blur
Edge map: $E \leftarrow \text{Canny}(I_s)$
Apply ROI mask: $E_r \leftarrow E \wedge \text{ROI\_mask}$
$L \leftarrow \text{HoughLinesP}(E_r)$ ▷ Probabilistic Hough
$L_{left}, L_{right} \leftarrow \text{average\_and\_extrapolate}(L)$
if L_{left} and L_{right} exist **then**
    compute lane bottom intersections $\Rightarrow x_{left}, x_{right}$
    lane_center $= (x_{left} + x_{right})/2$
    veh_center $=$ image_width$/2$
    deviation $= veh\_center - lane\_center$
    if $|deviation| > threshold$ **then**
        Output: WARN
    else
        Output: OK
    end if
else
    Output: `Lane detection unreliable`
end if

Steps of the Algorithm

1. Convert to Grayscale

Why: Lane detection depends on intensity edges not color. Reducing the image to one channel reduces computations and simplifies gradient computations.
How it helps: Easier edge detection; less data to process.
Alternatives: Use HLS and inspect the L channel (works better under varied lighting), or adaptive brightness normalization.

2. Gaussian Blur

Why: Smooths sensor noise and small texture; reduces spurious edges.
How it helps: Canny will produce fewer false edges; edges from lane paint remain since they are larger-scale features.
Alternatives: Bilateral filter (preserve edges), median filter (salt-and-pepper noise).

3. Canny Edge Detection

Why: Extracts strong, thin edges; robust hysteresis connects broken edges.
How it helps: Leaves line-like structures for Hough to operate on.
Alternatives: Sobel (simpler, noisier), deep-learning edge detectors (accurate but heavy).

4. Apply ROI Mask

Why: Only part of the image (lower center) contains useful lane lines. Masking discards irrelevant background.
How it helps: Fewer false positives and faster running time.
Alternatives: Dynamic ROI via horizon detection or semantic segmentation to isolate road.

5. Hough Line Transform

Why: Convert noisy edge points to explicit line parameters that represent lane markings.
How it helps: Detects straight segments and returns line segments even with dashed lines.
Alternatives: Hough probabilistic (used here), RANSAC line fitting, polynomial curve fitting for curved lanes.

6. Average & Extrapolate Lines

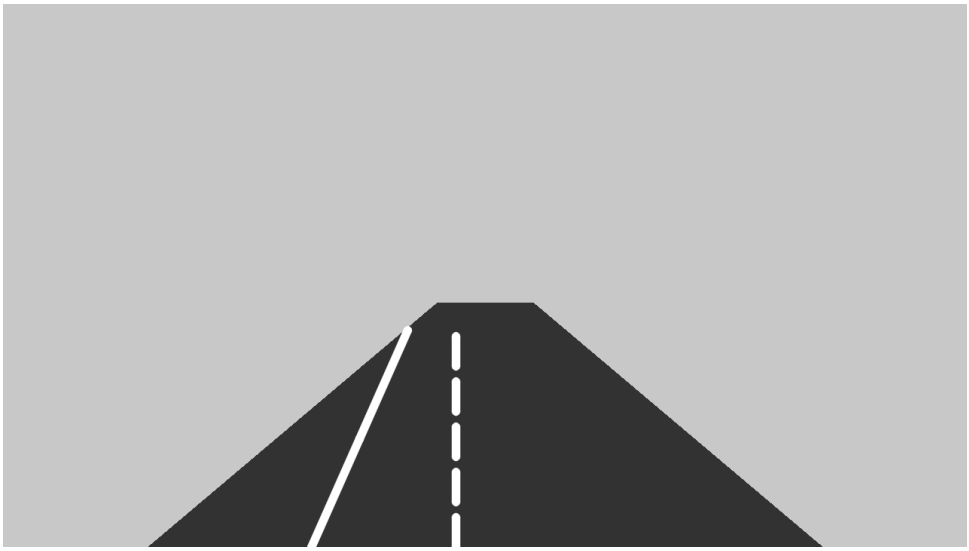
Why: Hough returns many short segments; average them to form stable left/right lane lines and extrapolate to full visible extents.
How it helps: Reduces jitter across frames and provides a single, usable lane boundary for each side.
Alternatives: Kalman filter for temporal smoothing; polynomial fit for curved highways.

7. Lane Center vs Vehicle Center

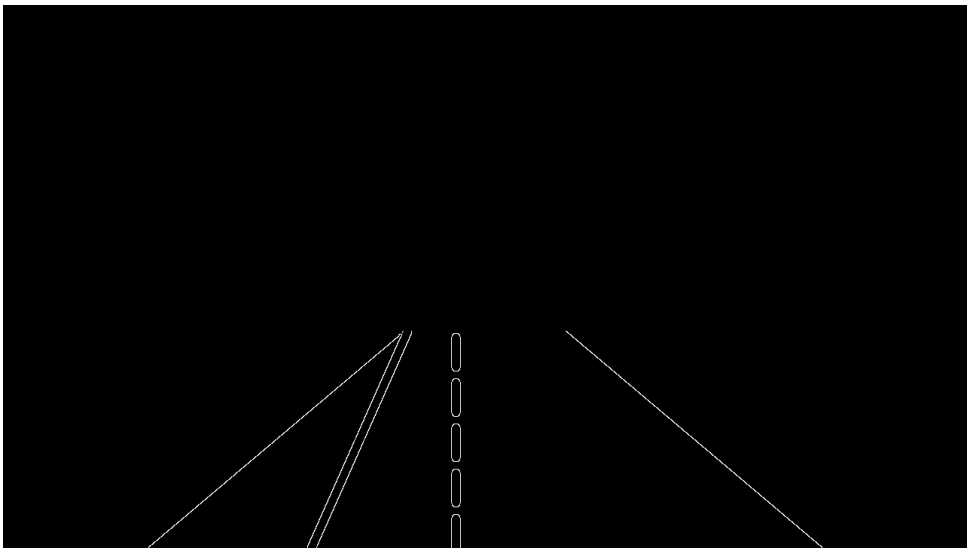
Why: Simple geometric check to detect lane departure.
How it helps: Produces a clear OK/WARN decision based on pixel deviation threshold.
Alternatives: Normalize deviation by lane width, use vehicle odometry (speed/steer) for trajectory prediction.

Detection Images

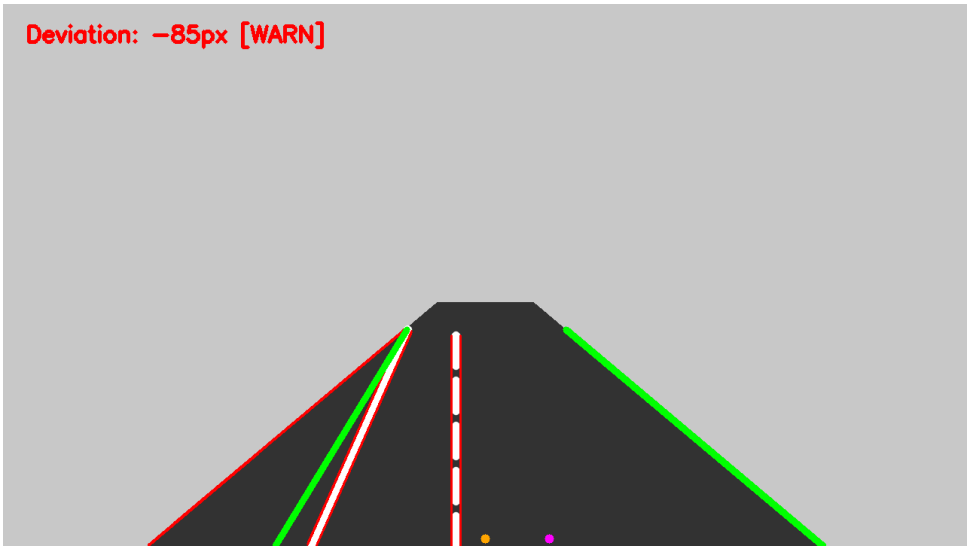
(Input Image)



(Edge map / ROI)



(Annotated Output)



Limitations

- Struggles in heavy rain, strong shadows, night-time, or when lane markings are faded.
- Detects mostly straight-ish lanes; curves require polynomial modeling (2nd order) or perspective transform.
- Occlusion by other vehicles may mask lane markings causing unreliable detection.

More Advanced Alternatives

- **Deep Learning Lane Detectors:** LaneNet, SCNN, Ultra-Fast-Lane-Detection — segment lane pixels directly. Pros: robust to curves and degraded markings; Cons: require GPU and labeled data.
- **Bird's-eye View (Perspective Transform):** Warp the road to top-down and fit lane polynomials for better curvature estimation.
- **Sensor Fusion:** Combine camera with IMU/GPS/LiDAR for more robust lane/vehicle pose estimation.

Conclusion

This project used the following modules and techniques:

- **OpenCV** (image processing, Canny, HoughLinesP), **NumPy** (numerical computations).

- Pipeline: Grayscale → Gaussian Blur → Canny → ROI → Hough → averaging/extrapolation → lane-departure check.

This project explored how classical computer vision techniques can be combined into a working pipeline for lane detection and lane departure warning. By carefully applying grayscale conversion, Gaussian smoothing, Canny edge detection, and the Hough line transform, I was able to extract stable lane boundaries from road images and videos. The approach works well in clear daylight conditions and shows that traditional methods, even without deep learning, can provide a practical solution for real-time applications.

While implementing the system, I noticed that preprocessing choices such as the size of the Gaussian kernel or Canny thresholds strongly influence performance. Similarly, defining a proper region of interest was critical to avoid false detections. These observations reinforced the importance of tuning and experimentation.

Overall, this project demonstrated that even simple, interpretable methods can form the basis of intelligent driver-assistance features, while also highlighting why more advanced methods are needed in complex environments.

References

[1] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[2] J. F. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

Appendix: Implementation (Python)

Below is the Python script used.

```
import cv2
import numpy as np

def roi_mask(img):
    h, w = img.shape[:2]
    mask = np.zeros_like(img)
    pts = np.array([[
        (int(0.1*w), h),
        (int(0.4*w), int(0.6*h)),
        (int(0.6*w), int(0.6*h)),
        (int(0.9*w), h)
    ]], dtype=np.int32)
    cv2.fillPoly(mask, pts, (255,)*img.shape[2])
    return cv2.bitwise_and(img, mask)

def average_and_extrapolate(lines, img_shape, min_slope=0.3):
    if lines is None: return None, None
    left, right = [], []
    for l in lines:
        x1,y1,x2,y2 = l[0]
        if x2 == x1: continue
```

```
        m = (y2-y1)/(x2-x1)
        if abs(m) < min_slope: continue
        b = y1 - m*x1
        if m < 0:
            left.append((m,b))
        else:
            right.append((m,b))
    h = img_shape[0]
    y1 = h
    y2 = int(h*0.6)
    def make_line(arr):
        if not arr: return None
        m = np.mean([a for a,_ in arr])
        b = np.mean([b for _,b in arr])
        if abs(m) < 1e-6: return None
        x1 = int((y1 - b) / m)
        x2 = int((y2 - b) / m)
        return (x1,y1,x2,y2)
    return make_line(left), make_line(right)

def process_frame(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5,5), 0)
    edges = cv2.Canny(blur, 50, 150)
    edges_roi = roi_mask(cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR))
    edges_roi_gray = cv2.cvtColor(edges_roi, cv2.COLOR_BGR2GRAY)

    lines = cv2.HoughLinesP(edges_roi_gray, 1, np.pi/180, threshold=20,
                            minLineLength=40, maxLineGap=20)

    left_line, right_line = average_and_extrapolate(lines, frame.shape)

    out = frame.copy()
    if lines is not None:
        for l in lines:
            x1,y1,x2,y2 = l[0]
            cv2.line(out, (x1,y1), (x2,y2), (0,0,255), 2)

    if left_line:
        cv2.line(out, (left_line[0], left_line[1]), (left_line[2],
           left_line[3]), (0,255,0), 8)
    if right_line:
        cv2.line(out, (right_line[0], right_line[1]), (right_line[2],
           right_line[3]), (0,255,0), 8)

    if left_line and right_line:
        ml = (left_line[3]-left_line[1])/(left_line[2]-left_line[0])
        bl = left_line[1] - ml*left_line[0]
        mr = (right_line[3]-right_line[1])/(right_line[2]-right_line
           [0])
        br = right_line[1] - mr*right_line[0]
        bottom_y = frame.shape[0]
        if abs(ml) < 1e-6 or abs(mr) < 1e-6:
            cv2.putText(out, "Lane detection unreliable", (30,50),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)
            return out, edges_roi_gray
        left_x = int((bottom_y - bl)/ml)
        right_x = int((bottom_y - br)/mr)
```

```
        lane_center = int((left_x + right_x)/2)
        veh_center = frame.shape[1]//2
        cv2.circle(out,(lane_center,bottom_y-10),6,(255,0,255),-1)
        cv2.circle(out,(veh_center,bottom_y-10),6,(0,165,255),-1)
        dev_pix = veh_center - lane_center
        thresh = int(0.05 * frame.shape[1])
        state = "WARN" if abs(dev_pix) > thresh else "OK"
        cv2.putText(out, f"Deviation: {dev_pix}px [{state}]", (30,50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,0,255) if state==
                       "WARN" else (0,255,0), 3)
    else:
        cv2.putText(out, "Lane detection unreliable", (30,50),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)

    return out, edges_roi_gray

if __name__ == "__main__":
    import sys, os
    source = sys.argv[1] if len(sys.argv)>1 else "project_video.mp4"
    ext = os.path.splitext(source)[1].lower()
    if ext in [".jpg", ".jpeg", ".png", ".bmp"]:
        frame = cv2.imread(source)
        out_frame, edges = process_frame(frame)
        cv2.imwrite("out_image.png", out_frame)
    else:
        cap = cv2.VideoCapture(source)
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out_vid = cv2.VideoWriter('out.mp4', fourcc, 20.0,
                                  (int(cap.get(3)), int(cap.get(4))))
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret: break
            out_frame, edges = process_frame(frame)
            out_vid.write(out_frame)
        cap.release(); out_vid.release()
```

Listing 1: hough.py

I have also included two demonstration videos showcasing the model's detection performance on real driving footage.