Real-Time Vision-Based Lane Detection for ADAS

Technical Report & Implementation Guide

Mohammed Amansour
Faculty of Sciences and Technology, Fes
Master in Electrical Engineering and Embedded Systems

February 2026

Abstract

This report details the mathematical foundations, system architecture, and usage of the ADAS Lane Detection system. Developed in MATLAB R2025b, the system implements a pipeline for real-time lane tracking using illimunation-invariant HSV segmentation, Canny edge detection, and temporal ROI tracking. The report serves as a complete reference for the algorithmic design and future embedded deployment.

Contents

1 Introduction

Lane detection is a critical perception task for Advanced Driver Assistance Systems (ADAS). This project provides a modular algorithm capable of detecting lane boundaries under varying lighting conditions (shadows, tunnels) and predicting vehicle turn direction (Left/Right/Straight).

The system prioritizes deterministic Computer Vision techniques over "black box" Deep Learning approaches to ensure explainability and efficiency on embedded hardware.

2 System Architecture

The codebase is organized into three logical layers: Perception, Estimation, and Visualization.

2.1 File Structure & Roles

- **Core Logic:**
  - `src/main_process_video.m`: The entry point. Loops through video frames and orchestrates the pipeline.
- **Perception Layer:**
  - `getAdaptiveThresholds.m`: Computes dynamic luminance stats and HSV color thresholds.
  - `buildRoiEdges.m`: Performs Gaussian smoothing, HSV masking, Canny detection, and ROI cropping.
  - `detectHoughLines.m`: Extracts linear features from the edge map using the Hough Transform.
- **Estimation Layer:**
  - `collectLanePoints.m`: Selects left/right candidates using geometry and tracking corridors.
  - `updateLaneState.m`: Fits polynomials and applies Exponential Moving Average (EMA) smoothing.
  - `computeVanishingPoint.m`: Calculates intersection of lane models for turn prediction.
- **Visualization:**
  - `drawOverlay.m`: Renders lane polygons and HUD text.
  - `generateLaneCurves.m`: Generates smooth visualization points with horizon fading.

3 Mathematical Foundations

This section details the algorithmic operations performed at each stage of the pipeline, explaining not just the equations but the data transformations (Input $\rightarrow$ Output) and the engineering rationale behind them.

3.1 Preprocessing: Gaussian Smoothing

Raw video frames from automotive cameras contain high-frequency noise caused by sensor thermodynamics and compression artifacts. If left untreated, these single-pixel intensity spikes create false edges.

Operation: We apply a Gaussian filter, which is a mathematical convolution of the image I with a kernel G. For a pixel at (x, y), the new intensity is a weighted average of its neighbors, where weights fall off according to a bell curve:

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

- **Input:** Raw $H \times W \times 3$ RGB frame.
- **Output:** Smoothed RGB frame with preserved edges but suppressed noise.
- **Effect:** The standard deviation σ controls the blur amount. We use 3D filtering to smooth channels independently, preparing the image for gradient detection.

3.2 Perception: HSV Color Segmentation

Standard RGB thresholding is sensitive to illumination changes because a shadow reduces R, G, and B values simultaneously. To detect lanes consistently, we transform the image into HSV (Hue, Saturation, Value) space.

- **Hue (H):** The "color" angle (e.g., Yellow is $\sim 60°$). Invariant to lighting.
- **Saturation (S):** The purity of color vs. white/gray.
- **Value (V):** The brightness intensity.

Logic for Yellow Lanes: We define yellow not by its RGB mix, but by its position on the color wheel. A pixel is classified as yellow lane marking if: 1. **Hue Check:** $0.08 \leq H \leq 0.17$ (Correponds to yellow-orange spectrum). 2. **Saturation Check:** $S \geq 0.4$ (Must be vivid, not washed out). 3. **Value Check:** $V \geq V_{\text{adaptive}}$ (Must be brighter than the road shadow).

Logic for White Lanes: White is achromatic. A pixel is white lane marking if: 1. **Saturation Check:** $S \leq 0.25$ (Low color content). 2. **Value Check:** $V \geq V_{\text{adaptive}}$ (High brightness).

$$Input : HSV Image \rightarrow Output : Binary Mask(1 = Lane, 0 = Background) \tag{2}$$

3.3 Region of Interest (ROI) & Horizon Clamp

Processing the entire image includes the sky, trees, and other vehicles, which generates false edges. We apply a spatial mask to isolate the road surface.

Geometric Definition: The ROI is a trapezoid defined by four vertices $\mathbf{V}_{\text{ROI}}$: bottom-left, bottom-right, top-right, and top-left. To avoid detecting the horizon or sky (which creates strong horizontal edges), we enforce a "Horizon Clamp":

$$y_{\text{top}} = \max(0.5 \times \text{ImageHeight}, \text{EstimatedHorizon}) \tag{3}$$

Any pixel above the midline ($y < 0.5H$) is strictly ignored.

- **Input:** Edge map from Canny detector.
- **Output:** Masked edge map where only road features remain.

3.4 Estimation: ROI Tracking with Temporal Priors

Searching for lanes across the entire image in every frame is inefficient and prone to errors (e.g., locking onto a guardrail). Once a lane is found, we assume it will not move significantly in the next frame ($\sim$ 33ms).

Tracking Logic: Let $\hat{x}_{t-1}(y)$ be the polynomial curve of the lane from the previous frame. For a new candidate point (x_i, y_i) detected in the current frame, we calculate its horizontal distance to the expected lane position:

$$d = |x_i - \hat{x}_{t-1}(y_i)| \tag{4}$$

We impose a strict condition:

$$\text{Keep Point if } d < 50 \text{ pixels} \tag{5}$$

This creates a "Search Corridor." Points outside this corridor are discarded as noise.

- **Input:** Set of all candidate line points.
- **Output:** Filtered subset of points belonging to the actual lane.
- **Result:** Prevents the tracker from jumping to adjacent vehicles or shadows.

4 Usage Guide

4.1 Prerequisites

- MATLAB R2024b/2025b
- Computer Vision Toolbox
- Automated Driving Toolbox

4.2 Running the Algorithm

1. Place input video in `data/test_drive.mp4`. 2. Open MATLAB and navigate to the `src/` directory. 3. Execute the main script:

```
>> main_process_video
```

4. The output will be visualized in real-time windows and saved to `data/output_annotated.avi`.

5 Conclusion

This project successfully demonstrates a real-time lane detection system for ADAS applications using classical computer vision techniques. By leveraging HSV color space segmentation and temporal ROI tracking, the algorithm achieves consistent performance under varying illumination conditions without relying on deep learning models. The modular architecture in MATLAB facilitates understanding of the complete perception-to-estimation pipeline, serving as a foundational reference for embedded automotive systems.