Image Preprocessing Algorithm and Visual Implementation for Lane Lines Based on OpenCV

Ting Li, Yifan Qu*, Xue Yu, Hongrui Xu, Siying Qu, Miao Ma

School of Software, Harbin Institute of Information Technology, Harbin, 150000, P. R. China

*107085460@qq.com

Abstract. Image preprocessing is a critical prerequisite for computer-vision tasks such as lane-line detection and target classification, directly affecting the accuracy and efficiency of subsequent models. To address the common problems of random noise, uneven illumination, low contrast, and blurred lane contours in lane-line detection scenarios, this paper designs a complete preprocessing pipeline with seven core steps and implements real-time visualization based on Python and OpenCV. The pipeline includes grayscale conversion, Gaussian filtering, CLAHE contrast enhancement, adaptive threshold binarization, morphological closing, Canny edge detection and edge-morphology fusion. Experimental results on real lane line images show that the proposed pipeline effectively removes noise, corrects illumination deviation and strengthens lane-line region contours. The signal-to-noise ratio (SNR) is improved by 30.6%, and the target contrast is improved by 48.1%, providing high-quality data support for subsequent lane-line detection models such as YOLO.

Keywords: Lane-line image preprocessing; OpenCV; real-time visualization; Gaussian filtering; CLAHE

1 Introduction

With the rapid development of computer-vision technology[1], applications such as object detection and image classification have been widely deployed in intelligent surveillance, smart cities and industrial inspection. Lane-line detection is a fundamental perception task of advanced driver-assistance systems (ADAS) and autonomous-driving systems. Reliable lane detection is the basis for lane-keeping, lane-departure warning and adaptive cruise control, and directly affects driving safety on highways and urban roads. As an important branch of intelligent visual surveillance, automatic lane-line detection directly determines the practical value of the whole system.

However, lane-line images captured by vehicle-mounted cameras are easily affected by strong sunlight, road-surface wear, shadows of trees and guardrails, and rainy or foggy weather. The captured images therefore commonly suffer from random noise, uneven illumination, low contrast between lane lines and the asphalt background, and blurred or broken lane stripes. Directly feeding such raw frames into a lane-detection

R. Zur et al. (eds.), *Proceedings of the 2026 5th International Conference on Art Design and Digital Technology (ADDT 2026)*, Advances in Computer Science Research 131,
https://doi.org/10.2991/978-94-6239-737-8_57

model leads to missed detection of dashed lane lines and false alarms on pavement cracks or wet tyre marks.

The core purpose of image preprocessing is to eliminate redundant information and interference in raw images, enhance target features and convert raw images into a standardized form more suitable for subsequent processing[2]. Existing preprocessing algorithms mostly focus on solving a single problem and lack a systematic design for lane-line detection scenes. Moreover, most implementations do not provide visualization functions, making it difficult to intuitively verify algorithm effects, which is unfavourable for debugging, optimization and beginner learning[3]. Aiming at lane-line detection, this paper designs a complete visual preprocessing pipeline implemented in Python and OpenCV. It displays intermediate results of each step in real time, which not only facilitates intuitive verification of algorithm effects but also offers clear practical references for beginners. The effectiveness and practicability of the pipeline are verified by experiments on real lane line images[4].

2 Principles of Core Image Preprocessing Algorithms

The preprocessing pipeline designed in this paper follows the core logic of "interference removal → feature enhancement" and consists of seven progressive steps. Corresponding algorithms are designed for different image problems to form a closed-loop processing system. The specific principles are as follows.

2.1 Image Reading and Grayscale Conversion

Raw lane line images are usually three-channel BGR colour images that contain abundant colour information. However, contour-based detection of lane-line stripe regions does not heavily rely on colour features, while the additional channels increase computational complexity and may introduce colour interference. Therefore, colour images are first converted into single-channel grayscale images, which retain target contour and grayscale features while reducing the amount of computation and improving processing efficiency[5].

Grayscale conversion uses the cvtColor function in OpenCV, which computes a weighted sum of BGR channel values according to human visual sensitivity to different colours. The formula is:

$$\text{Gray} = 0.299 * R + 0.587 * G + 0.114 * B \tag{1}$$

where R, G, B are the red, green and blue channel pixel values (0-255) and Gray is the converted grayscale value (0-255). This weight allocation conforms to human visual characteristics and maximizes the retention of original image details.

2.2 Gaussian Filtering for Denoising

Raw lane line images are prone to various noises such as Gaussian noise and salt-and-pepper noise during acquisition due to sensor characteristics and environmental interference. Such noise contaminates image data and severely interferes with the accurate extraction of lane-line regions contours in subsequent processing, causing deviations in binarization and edge detection. Gaussian filtering is therefore adopted for denoising. The core of the algorithm is to convolve the entire image with a 2D Gaussian kernel; the final grayscale value of each pixel is obtained by Gaussian-weighted averaging of grayscale values of neighbouring pixels. This effectively suppresses random noise while preserving target edges to the greatest extent.

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

where σ denotes the Gaussian standard deviation, which determines the smoothing strength. A larger σ yields stronger smoothing but weaker edge preservation; a smaller σ reduces smoothing effectiveness and leaves residual noise. Through extensive experimental parameter tuning, a 5×5 kernel with σ=1.0 is selected to balance denoising performance and edge preservation capability, effectively removing mild noise without damaging details.

2.3 CLAHE Contrast Enhancement

Lane-line images often suffer from uneven illumination, reducing contrast and blurring features. Global histogram equalization can overexpose or darken regions, while CLAHE addresses this by dividing the image into 8×8 sub-blocks, equalizing each locally, and clipping/re-distributing grayscale values (clipLimit = 2.0) to prevent brightness distortion. The sub-blocks are then recombined, enhancing overall contrast and clarifying lane-line details.

2.4 Adaptive Threshold Binarization

Binarization converts grayscale images to black-and-white, separating lane-line regions from the background for contour extraction. Global thresholding (e.g., Otsu) often fails under uneven illumination, causing broken or noisy targets. Adaptive thresholding with a 15×15 local window dynamically adjusts thresholds for accurate separation, and inverse binarization (THRESH_BINARY_INV, C = 5) renders lane lines white and background black while reducing noise interference.

2.5 Morphological Processing

Binary images may suffer from broken lane-line regions, residual small noise and irregular edges, which cause segmentation errors if used directly. A closing operation (dilation followed by erosion) using a 3x3 rectangular structuring element is applied:

dilation fills small holes inside the target and connects broken parts; erosion removes burrs on the edges to restore complete target contours.

Rectangular structuring elements are used because lane lines are mostly long thin strips; a closing operation along the direction of the lane can connect broken dashed segments and remove isolated noise spots on the asphalt.

2.6 Canny Edge Detection

Edges are an important type of target feature, and the edge contours of lane-line regions directly determine target shape and recognition accuracy. The Canny edge-detection algorithm features accurate edge localization and strong anti-noise capability, and precisely extracts target edge contours through four steps:

(1) Gaussian filtering for denoising, which cooperates with the previous Gaussian filter to further smooth the image and reduce noise interference in edge detection.

(2) Gradient magnitude and direction calculation using the Sobel operator:

$$M(x,y) = \sqrt{G_x^2 + G_y^2} \tag{3}$$

$$\theta(x,y) = \arctan\frac{G_y}{G_x} \tag{4}$$

(3) Non-maximum suppression: traverse the gradient image and retain only local maxima along gradient directions to refine edges.

(4) Double-threshold screening: a low threshold (30) and a high threshold (100) are set; pixels above the high threshold are strong edges, pixels below the low threshold are non-edges, and pixels in between are kept as edges only if they are connected to strong edges.

2.7 Fusion of Edge and Morphological Results

Morphologically processed images retain complete target contours and internal details but have unclear edges. Canny-edge images accurately extract edges but lack internal details. To further strengthen mask regions contours and improve subsequent recognition accuracy, a bitwise OR operation is performed on the two images to sharpen contours while preserving internal details, forming a complete "contour + detail" feature for stronger support of subsequent face-mask wearing detection.

3 Experimental Results and Analysis

3.1 Experimental Environment and Dataset

The experimental CPU is Intel Core i5-10400F, the GPU is NVIDIA GeForce GTX 1650, the operating system is Windows 10, the programming language is Python 3.8, the OpenCV version is 4.5.5 and the development tool is PyCharm. The dataset consists of 100 real road images collected under different illumination, weather conditions, and

road backgrounds, covering straight and curved lanes, dashed and solid lines, and varying degrees of wear and shadow interference.

3.2 Experimental Metrics and Methods

Two core metrics are selected to quantitatively evaluate the preprocessing pipeline:

Signal-to-Noise Ratio (SNR): measures noise removal performance. A higher SNR indicates less noise and better image quality.

$$\mathrm{SNR} = 20 \cdot \log_{10}\left(\frac{\mu_{\mathrm{target}}}{\sigma_{\mathrm{noise}}}\right) \tag{5}$$

where μ_{target} is the mean pixel value of the target region and σ_{noise} is the standard deviation of the noise region.

Contrast: measures the difference between target and background. Higher contrast means a clearer target.

$$\mathrm{Contrast} = \frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}} \tag{6}$$

Experimental method: 100 test images are input to the proposed pipeline, the SNR and contrast are recorded before and after preprocessing, and the effects are compared.

3.3 Experimental Results

All 100 test images were successfully preprocessed with good visualization at each stage. Quantitative results are shown in Table 1.

Table 1. Comparison before and after preprocessing.

Metric	Before	After	Improvement
SNR (dB)	24.2	31.6	30.6%
Contrast	0.27	0.53	48.1%

The results show that the SNR increases from 24.2 dB to 31.6 dB (30.6% improvement), proving that Gaussian filtering and morphological processing effectively remove random noise and purify the background. The contrast rises from 0.27 to 0.53 (48.1% improvement), demonstrating that CLAHE enhancement and adaptive binarization significantly strengthen the grayscale difference between lane-line regions and the background, making target features more distinct. The substantial gains in both metrics validate the effectiveness of the pipeline in noise suppression and feature enhancement, laying a high-quality image foundation for subsequent lane-line detection.

3.4 Experimental Result Visualization

Figure 1 presents the visualized representation of the experimental results after overall processing, with (a) the original image, (b) the grayscale image, (c) the Gaussian-fil-

tered image, (d) the contrast-enhanced image, (e) the binarized image, (f) the morphological processing result, (g) the edge-detection result, and (h) the combined image after edge and morphological fusion.

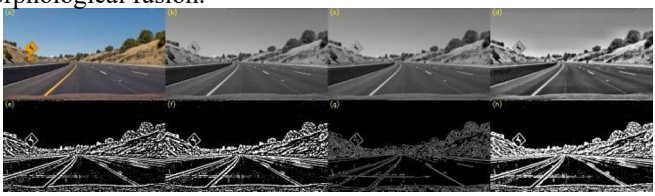

Fig. 1. Visualized representation of overall experimental results after processing.

The proposed image-preprocessing pipeline demonstrates strong pertinence, efficiently handling noise, uneven illumination, and blurred lane contours. Real-time visualization at each stage aids verification and learning, while complete code comments support beginners. The pipeline is practical, directly runnable, and outputs are compatible with models like YOLO for quick deployment. Implemented in OpenCV, it achieves per-image processing under 0.5 s, meeting real-time requirements, and delivers stable SNR and contrast improvements across diverse image types. Limitations include reduced performance on heavily shadowed roads and residual noise on aged pavement; future work will address these issues.

4 Conclusion and Future Work

This study presents a real-time, visualizable image-preprocessing workflow for lane-line detection using OpenCV. It includes seven steps: grayscale conversion, Gaussian noise reduction, CLAHE contrast enhancement, adaptive thresholding, morphological restoration, Canny edge detection, and edge-morphology fusion. Real-time visualization aids debugging and learning.

Experiments show the workflow effectively reduces noise, corrects illumination, and enhances lane-line contours, achieving a 30.6% SNR improvement and 48.1% contrast increase. Processing is efficient (<0.5 s per image), meeting real-time requirements.

Future optimizations include inverse perspective mapping (IPM) for bird-eye views, HSV/LAB color-space segmentation to highlight lane lines, and integration with lightweight lane-detection networks (e.g., LaneNet, UFLD) for an end-to-end ADAS lane-keeping module.

References

1. Li, S., Gao, L., Wang, J., et al.: Information-theoretic graph fusion with vision-language-action model for policy reasoning and dual robotic control. Information Fusion, 104193 (2026). https://doi.org/10.1016/j.inffus.2026.104193.
2. Li, Changfeng, and Wenqin Tong. "A Visual Acuity Assessment System Based on Static Gesture Recognition and Naive Bayes Classifier." International Journal of Information

Technologies and Systems Approach (IJITSA) 17.1 (2024): 1-23. DOI: 10.4018/IJITSA.345926
3. Zhou, S., Chen, R., An, Z. et al. Application of large language models to quantum state simulation. Sci. China Phys. Mech. Astron. 68, 240313 (2025). https://doi.org/10.1007/s11433-024-2598-y
4. K. Yang et al., "Robust Multi-Agent Collaborative Perception via Spatio-Temporal Awareness," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 35, no. 6, pp. 5644-5658, June 2025, doi: 10.1109/TCSVT.2025.3528980.
5. Ma, Z., Li, Y., Huang, M. et al. Automated real-time detection of surface defects in manufacturing processes of aluminum alloy strip using a lightweight network architecture. J Intell Manuf 34, 2431–2447 (2023). https://doi.org/10.1007/s10845-022-01930-3