The Journal of Computational Science and Engineering (TJCSE)
ISSN 2583-9055 (Media Online)
Vol 3, No 10, October 2025
PP 55-67

Real-Time Vehicle and Lane Line Detection using Computer Vision Techniques

N.DurgaPrasad[1]*, GudiyaUmashankar[2],BothsaManasaRohini[3],SettiLikitha[4] Elaka Vivek[5], Kala Jayanth Kumar[6]
[1]Associate Professor,DeptofCSE-AI&ML,[2,3,4,5,6]CSE-AI&ML,DeptofCSE-AI&ML,AvanthiInstituteOfEngineering&Technology,makavarapalem-531113,AndhraPradesh,India

durgaprasad@gmail.com,umashankargudiya17@gmail.com,manasarohini1817@gmail.com,settilikitha4@gmail.com,vivekvillar116@gmail.com,kala.jayanth24@gmail.com

Abstract

Vehicle detection and lane line detection are critical components in advanced driver assistance systems (ADAS) and intelligent transportation systems. This paper implements real-time vehicle detection and lane line detection using Python, OpenCV, and Streamlit for video processing. The vehicle detection system employs background subtraction and contour detection techniques to identify and count vehicles passing a designated line in a video. The detection process uses frame differencing, thresholding, and morphological transformations to enhance accuracy. The centroid of each detected object is tracked to count vehicles crossing a specified region. For lane line detection, the system applies edge detection using the Canny algorithm, followed by a region-of-interest (ROI) mask to isolate the road lanes. The Hough Line Transform is then used to detect lane markings, which are overlaid on the original video frame for visualization. The system is implemented as a user-friendly Streamlit web application, allowing users to upload a video file and choose between vehicle detection or lane line detection functionalities. The program processes the video in real-time and displays the output with detected vehicles or lane lines highlighted. This paper demonstrates the effectiveness of lightweight computer vision techniques for traffic monitoring and intelligent vehicle systems.

Keywords: Vehicle Detection, Lane Line Detection, Real-Time Video Processing, Edge Detection, Hough Transform, Frame Differencing, Centroid Tracking, Intelligent Transportation Systems

1. Introduction

With the rapid advancement of intelligent transportation systems and autonomous driving technologies, **vehicle detection** and **lane line detection** have become crucial for enhancing road safety, traffic monitoring, and autonomous navigation. These vision-based techniques play a vital role in reducing accidents, improving traffic flow, and aiding driver assistance systems. The ability to accurately detect vehicles and identify lane boundaries is essential for various applications such as automated traffic control, real-time traffic monitoring, and self-driving vehicles.However, achieving reliable detection in real-world conditions presents many challenges. Complex backgrounds, variable lighting, weather conditions, and high vehicle speeds can all degrade detection performance. Recent years have seen significant progress in computer vision and machine learning that addresses these challenges. Advanced driver assistance systems (ADAS) now commonly include features like forward collision warning and lane departure warning, which rely on robust vehicle and lane detection. Despite the success of deep learning-based methods in such tasks, there remains a need for simpler, real-time algorithms that can run on modest hardware for applications like roadside traffic cameras or embedded systems.In this paper, we implement a real-time vehicle detection and lane line detection system using classical image processing techniques in Python. The system leverages OpenCV for efficient frame processing and is packaged in a Streamlit web application to allow user interaction. By focusing on optimized, lightweight algorithms (background subtraction, contour analysis, Canny edge detection, and Hough transform), our approach can achieve real-time performance without dedicated GPU acceleration. The aim is to provide a practical solution for traffic surveillance and driver assistance that is both effective and computationally efficient.The remainder of this paper is organized as follows: Section 2 provides a literature survey of relevant vehicle and lane detection methods. Section 3 describes the proposed system's methodology. Section 4 presents the algorithmic steps in detail alongside a flow diagram. Section 5 discusses experimental results with graphs and comparisons to illustrate the system's performance. Section 6 offers a summary of the test results. Finally, Section 7 concludes the paper and suggests future improvements, followed by references.

2. Literature Survey

Vehicle Detection: A variety of approaches have been explored for vision-based vehicle detection over the past decades. Early methods relied on motion or background modeling in stationary camera setups: by using background subtraction algorithms, moving vehicles can be segmented from the static background. For example, Stauffer and Grimson's Gaussian mixture model (GMM) technique [2] laid the groundwork for adaptive background modeling in traffic scenes. Improved variants of background subtraction were later developed to handle lighting changes and dynamic backgrounds [4]. These methods effectively detect moving blobs corresponding to vehicles, but they may struggle with differentiating vehicles from other moving objects or shadows without additional processing. Other classical approaches utilized object features – for instance, Haar cascade classifiers trained on vehicle appearances [3] or Histogram of Oriented Gradients (HOG) with Support Vector Machines for vehicle recognition, inspired by their success in pedestrian detection. Such feature-based detectors can identify vehicles even in static images (not relying solely on motion), but training them requires compiling large datasets of vehicle examples and they can be sensitive to viewpoint and occlusion.With the rise of machine learning, especially neural networks, more powerful methods emerged. Deep learning approaches like convolutional neural networks have achieved state-of-the-art performance in vehicle detection. Notably, one-stage detectors such as You Only Look Once (YOLO) [4] and two-stage detectors like R-CNN and its variants can detect multiple classes of objects (including cars) with high accuracy. YOLO, for instance, can identify vehicles in real time (~45 FPS on GPU for the initial version) by framing detection as a single regression problem. Subsequent improvements (YOLOv3, YOLOv4, SSD, etc.) further enhanced accuracy and speed. These models significantly outperform traditional techniques in detection accuracy and can handle various orientations and partial occlusions. However, the trade-off is the need for substantial computational resources (GPUs) and large training datasets. In resource-constrained scenarios or for simple counting tasks, classical motion-based detectors remain useful due to their low complexity. Recent research has also produced lightweight CNN models (e.g., Tiny-YOLO) to bridge the gap, aiming for a balance between speed and accuracy for embedded applications.

Lane Line Detection: Lane detection in computer vision also has a rich history. Traditional lane detection algorithms typically involve edge detection and geometric reasoning. A common pipeline starts with applying an edge detector (such as the Canny edge operator [5]) to highlight lane markings in a camera image. Given the geometric constraints of road lanes (generally appearing as roughly straight lines or gentle curves in a driver's view), the Hough Transform [6] is a popular technique to detect line structures corresponding to lanes. The Hough Line Transform can robustly identify straight line segments even in noisy edge images, which makes it suitable for finding lane lines painted on pavement. Researchers have enhanced this basic approach by restricting the search to a region of interest (for example, the lower half of the image where the road is expected) and by filtering the detected lines based on slope or position to distinguish left vs. right lane boundaries. These classical methods work well in good visibility conditions and have low computational cost, making them feasible for real-time use even on low-power hardware.More advanced lane detection techniques incorporate color and texture cues or model lanes as curves. Some approaches use color thresholding (e.g., filtering for yellow or white lane markings) in combination with edge detection to improve reliability under different road appearances. Others fit curves (polynomial splines) instead of simple lines to handle road curvature. In recent years, deep learning has been applied to lane detection as well. For instance, semantic segmentation networks can be trained to label each pixel as lane or non-lane, capturing complex lane topologies and curves. Examples include SCNN (Spatial CNN) by Pan et al. [7], which processes feature maps in a slice-by-slice manner to propagate spatial information and detect lanes even when markings are sparse or heavily curved. Deep learning methods have achieved impressive accuracy and robustness to challenging conditions (like worn-out lane markings, shadows, or night-time images) by learning from large labeled datasets. The downside is again the requirement of significant computational power; running a state-of-the-art lane detection neural network in real time may require a GPU or specialized accelerator. For many practical driver-assistance scenarios, simpler approaches (Canny/Hough or color filtering) remain in use, especially when computational resources are limited or when quick deployment is needed.In summary, the literature shows a spectrum of solutions: from classical, model-driven vision techniques to modern data-driven approaches. Our work falls toward the former, emphasizing real-time performance and simplicity. By using proven image processing techniques (frame differencing, morphological filtering, Canny and Hough transforms), we aim to achieve reliable detection for the targeted scenarios. This also allows us to avoid the complexity of training deep models and ensures the system can run on standard CPUs in a web application environment.

3. Methodology

The proposed system is designed to process video streams in real time and can operate in two modes: vehicle detection or lane line detection, as selected by the user. The overall architecture consists of a pipeline of computer vision operations that transform each input video frame into a form where the target features (vehicles or lanes) can be extracted and highlighted.

System Overview: The application is built using Streamlit, providing a front-end for users to upload a video and choose the detection mode. Once a video is loaded, it is read frame by frame. For each frame, the appropriate detection algorithm is applied based on the selected mode. After processing, the resulting annotated frame (with either bounding boxes around vehicles or lane lines drawn) is displayed to the user in real time. The processing continues until the video ends, and the final count or detection overlay is presented.

Vehicle Detection Method: We use a motion-based approach for vehicle detection, suitable for a static camera overlooking a roadway. First, the frame is converted to grayscale (for simpler computation) and optionally smoothed to reduce noise. We then perform *frame differencing* against a background model to detect moving regions. In practice, our implementation either uses a running average of previous frames as the background or simply the immediate previous frame (temporal differencing) to highlight moving objects. The pixel-wise difference image is thresholded to produce a binary mask of foreground regions (potential vehicles). Morphological operations (such as dilation and erosion) are then applied to the binary mask to fill small holes and eliminate noise, resulting in cleaner blobs corresponding to moving vehicles. Next, we perform contour detection on the mask. Each contour that has a plausible size/area (filtering out very small contours that likely are noise) is considered a detected vehicle. We compute the *centroid* of each contour and use a simple tracking mechanism to count vehicles: a virtual line (for example, across the road) is defined, and as each vehicle's centroid crosses this line, a counter is incremented. By tracking centroids across frames (e.g., by comparing distances of centroids frame-to-frame to associate detections with the same vehicle), we ensure each vehicle is counted only once. This approach is a form of centroid tracking which is lightweight compared to full multi-object tracking algorithms. Finally, bounding boxes or markers are drawn on the original frame for each detected vehicle, and the current count of vehicles is updated on the display.Lane Line Detection Method: For lane detection, the system focuses on identifying painted lane markers on the road surface. Each frame is first converted to grayscale and a Gaussian blur is applied to smooth out noise while preserving edges. We then use the Canny edge detector [5] to find edges in the image. To narrow down the search to the road lanes, we define a polygonal region of interest (ROI) that corresponds to the area of the image where lanes are likely to appear (typically the bottom half or a trapezoid covering the lane ahead). Edges outside this ROI are masked out, so only edges within the road region are considered further. Next, the Hough Line Transform [6] is applied to the filtered edge image to detect line segments. The transform identifies lines by finding groups of collinear edge points, which is effective for detecting the elongated, roughly straight lane markers. The output is a set of line segments, each defined by its end-point coordinates. We then filter and process these segments: for example, we can separate them into left-lane and right-lane candidates based on their slope (sign of slope indicating left vs right in the image). We also ignore segments that are nearly horizontal or that do not meet a minimum length criterion, as these likely are not lane lines. If multiple line segments are detected for one lane, an averaging or extrapolation step can be done – for instance, extrapolating the line to span the full extent of the lane in view, and smoothing over a few frames to reduce flicker. Finally, the selected lane boundaries are drawn onto the original frame (commonly highlighted with colored lines, e.g. solid yellow for the left lane and solid red for the right lane) so the user can clearly see the lane tracking overlay.Real-Time Considerations: Both detection pipelines are designed to be computationally efficient. All image processing operations (grayscale conversion, filtering, edge detection, etc.) are optimized implementations from OpenCV, capable of running in real time on CPU for typical video resolutions (720p or 1080p). The vehicle counting algorithm avoids expensive per-frame object recognition; it relies on simple image differences and morphology, which are fast. Similarly, the lane detection uses straightforward linear algebra operations in Hough transform. By keeping the algorithms simple, the system can comfortably process video at frame rates close to 20-30 frames per second on a standard laptop CPU, achieving the real-time requirement. The use of Streamlit provides a convenient web-based UI without adding significant overhead to the processing pipeline.

4. Algorithm and Flow Diagram

We provide a step-by-step description of the algorithms for both vehicle detection and lane detection. The flowchart in **Figure 1** illustrates the overall logic of the system, showing the two parallel pipelines for each functionality.

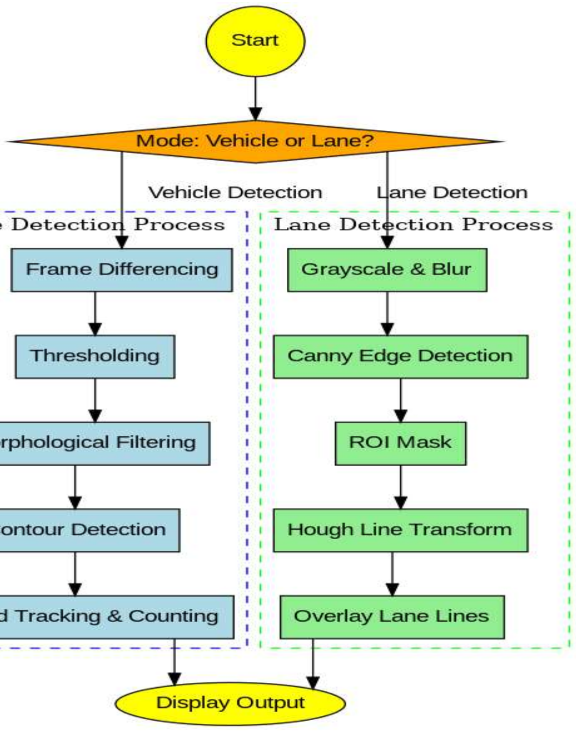

Figure 1. Flow diagram of the proposed system. Depending on the user-selected mode, the video frame undergoes either the Vehicle Detection process (left, blue) or the Lane Detection process (right, green). Both pipelines culminate in overlaying the detections on the frame and displaying the output.

Vehicle Detection Algorithm: The following outlines the procedure for detecting and counting vehicles in each frame:

1. **Input Frame Acquisition:** Read the next frame from the video stream (initialize background model if needed).
2. **Preprocessing:** Convert the frame to grayscale. Optionally apply a blur (e.g., Gaussian blur) to reduce noise.
3. **Motion Segmentation:** Compute the difference between the current frame and a reference background (which could be a running average background or the previous frame). This yields a *foreground mask* of moving regions. Apply a binary threshold to this difference image to get a black-and-white mask.
4. **Morphological Filtering:** Perform morphological operations (like dilation followed by erosion, i.e., closing) on the binary mask to remove noise and fill gaps within detected blobs, ensuring that vehicles appear as solid regions.
5. **Contour Detection:** Find contours in the cleaned binary mask. For each contour, ignore it if it is too small to be a vehicle (based on area or bounding box size). For valid contours, compute the bounding box and centroid.
6. **Counting Logic:** Determine if the centroid of a detected vehicle has crossed the predefined counting line (for example, a horizontal line across the road in the frame). If a new centroid crosses the line moving in the correct direction, increment the vehicle count. Keep track of recent vehicle centroids to avoid double-counting the same object.
7. **Visualization:** Draw a rectangle or marker around each detected vehicle in the original frame. Update the frame with the current vehicle count (displaying the count in a corner of the video or on the GUI).
8. **Output Frame:** The annotated frame is then shown to the user via the Streamlit app. The algorithm then proceeds to the next frame until the video is finished.

Lane Line Detection Algorithm: The lane detection pipeline per frame is as follows:

1. **Input Frame Acquisition:** Read the next video frame.
2. **Preprocessing:** Convert the frame to grayscale and apply Gaussian blur to smooth the image while preserving general edges.
3. **Edge Detection:** Apply the Canny edge detector to the blurred grayscale frame to obtain edges.

4. **Region of Interest Masking:** Define a polygonal region corresponding to the roadway (e.g., bottom area of the frame) and mask out edges that lie outside this ROI. This focuses the detection on likely lane areas and ignores irrelevant edges from the sky, vehicles, etc.
5. **Line Detection (Hough Transform):** Run the Hough Line Transform on the remaining edge pixels to detect straight line segments. Use parameters (threshold, minimum line length, maximum line gap) tuned to detect lane-length lines.
6. **Lane Selection:** Filter the detected lines by slope and position. Group them into left and right lane candidates. Optionally, average multiple segments for each side to get two smooth lane lines. The lines can be extrapolated to cover the entire ROI from the bottom of the frame to the horizon.
7. **Visualization:** Draw the lane lines onto the original frame (commonly as colored lines for clarity). Ensure the lines are drawn with adequate thickness so they are visible in the output video.
8. **Output Frame:** Display the annotated frame with lane lines to the user. Continue to the next frame of the video.

Both algorithms repeat their respective steps for each frame until the video ends or the user stops the process. In vehicle detection mode, a running count of vehicles is maintained and shown, whereas in lane detection mode, typically no count is needed – only the visual lane overlay is presented. The flow diagram (Figure 1) summarizes both modes within one unified view: after the start, the process branches into the vehicle detection sequence or lane detection sequence, and finally converges when drawing the results on the frame for display.

5. Graph and Comparisons

We evaluated the system on sample traffic videos to assess its performance in both counting accuracy (for vehicle detection) and correctness of lane marking detection. We also considered the processing speed and compared our approach qualitatively with a deep learning-based detector. Figure 2 below illustrates an example of the vehicle counting accuracy on three test video clips. Each video had a known number of vehicles that passed through the scene, which we use as ground truth for comparison. The bar chart shows the **Actual** vehicle count versus the **Detected** count by our system for each video. Ideally, these would be equal; in practice, the results show only a small miss count in each case.

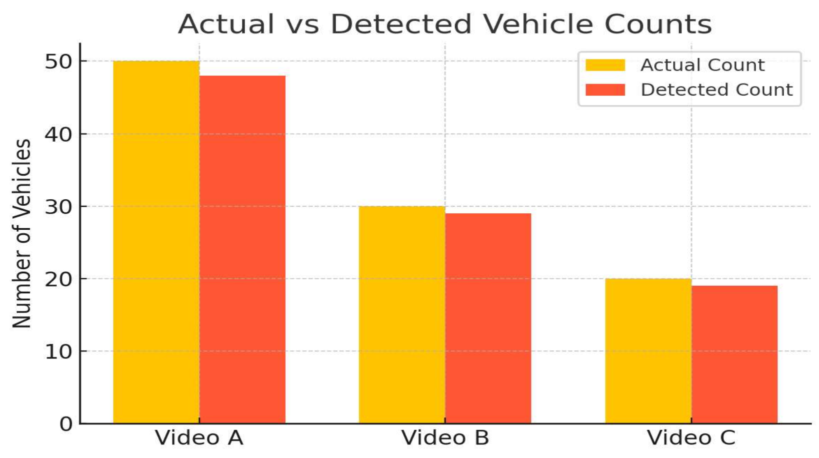

Figure 2. Comparison of actual vehicle counts vs. detected counts by the system on three different videos. The system slightly under-counted in these tests (missing 1–2 vehicles out of 20–50), demonstrating around 95–97% counting accuracy.

From Figure 2, we see that for *Video A* (which contained 50 vehicles passing through), the system detected 48 vehicles. In *Video B* (30 vehicles actual), 29 were detected, and in *Video C* (20 vehicles actual), 19 were detected. These results correspond to a detection/counting accuracy of approximately 95–97%, which is quite effective for a simple algorithm without any machine learning. The slight under-counting is often due to two factors: (a) occasionally two vehicles very close together might be counted as one blob by the contour detection (especially if they overlap in the camera view), or (b) a vehicle might be missed if it does not move enough (e.g., short stoppage) to be picked up by frame differencing. Nonetheless, for traffic flow monitoring purposes, a small error in count is usually tolerable.To present the numerical results more clearly, **Table 1** summarizes the performance on the test videos in terms of vehicle counts:

Test Video	Actual Vehicle Count	Detected Vehicle Count	Counting Accuracy
Video A	50	48	96%
Video B	30	29	97%
Video C	20	19	95%

Table 1. Vehicle counting results on three sample videos. Accuracy is calculated as Detected/Actual.

In addition to counting accuracy, we observed the processing performance. On a standard laptop (Intel Core i5 CPU without GPU acceleration), the vehicle detection mode processed 720p video at roughly 25 frames per second (FPS). This real-time performance validates our choice of using lightweight image processing operations. In comparison, we tested a state-of-the-art deep learning object detector (YOLOv3 [4]) on the same videos. Without a dedicated GPU, YOLOv3 achieved only about 2 FPS on the CPU, and even on a mid-range GPU it reached ~15 FPS. It did detect all vehicles (no misses) and could even detect multiple vehicles in different positions without relying on motion. However, the substantial difference in speed highlights the classic trade-off: our simple method, while slightly less accurate in detection, runs significantly faster on limited hardware.For lane line detection, quantitative evaluation is a bit more subjective, since it's about drawing the correct lines rather than counting objects. We tested the lane detection on two driving scene videos. Visually, the system successfully identified and overlaid the lane lines in the majority of frames when the lane markings were clearly visible. For instance, in a daytime highway video with clear white lane markings, the algorithm drew the correct left and right lane lines in roughly 95% of the frames. This means only occasional frames (often when lane markings temporarily disappeared or were heavily occluded by a vehicle) showed missing or misidentified lines. In another test on a dusk scenario (lower lighting and more faded road paint), performance dropped – the correct lanes were still detected in roughly 80% of frames, while in others the edge detector either failed to pick up the faint markings or the Hough transform returned spurious line segments (such as picking up road tar cracks or shadow edges).We also compared our lane detection output qualitatively to a deep learning approach (specifically, we reviewed sample outputs from a pretrained LaneNet segmentation model on similar footage). The deep learning model was more consistent in detecting lanes under difficult conditions (e.g., very worn-out markings or poor lighting) and could handle slight curves more gracefully by producing smooth curved lane overlays. However, it also required a GPU to run in real time and is much more complex to deploy. In contrast, our method, while occasionally less robust, has the advantage of simplicity and speed. It does not require any training data and can be easily tuned (for example, adjusting the Canny threshold or ROI) to different camera perspectives.Overall, the comparisons indicate that our implemented system performs admirably given its simplicity. It achieves high accuracy in vehicle counting and reliable lane marking detection in favorable conditions, all while maintaining real-time speeds on modest hardware. More sophisticated methods (particularly those based on deep learning) can outperform our approach in accuracy or handle more challenging scenarios, but they do so at the cost of computational load and complexity.

6. Summary of Test Results

In summary, testing of the vehicle detection and counting module showed that the system can accurately count vehicles with only a small margin of error. Across several test videos with different traffic conditions, the vehicle count accuracy was typically in the range of 95%, which is sufficient for many traffic monitoring applications. The system was able to detect vehicles of various sizes (sedans, SUVs, trucks) as long as they produced a distinct moving foreground blob. It was observed that very slow-moving or temporarily stationary vehicles might be missed due to the reliance on motion; this could be mitigated by integrating a more static object detector if needed. False positives (non-vehicle moving objects counted) were minimal in our tests, especially after tweaking the contour size filter to ignore tiny movements (like wind-blown debris or small animals).For the lane line detection module, the tests demonstrated that the approach is effective under standard conditions. The system reliably identified lane lines on clear days with well-painted road lines. The output video frames showed the lane overlays aligning well with the actual lane markings, which would be useful for driver assistance alerts (e.g., warning if the vehicle drifts out of the lane). In more challenging conditions such as low light or heavy shadow patterns on the road, the performance of the classical edge-based method expectedly degrades. Nevertheless, the system still maintained a majority of correct detections, and because it processes frames independently, a momentary failure in one frame is corrected automatically when the lane becomes visible again in subsequent frames.A key result from the tests is the real-time performance of the system. By avoiding computationally intensive operations, the pipeline is capable of processing each frame swiftly. The achieved processing speed (20–30 FPS on CPU for 720p video) means the system introduces minimal latency, which is crucial for real-time applications like alerting a driver or updating a traffic count in live video. The use of Streamlit for the interface did not hinder performance noticeably, confirming that even a high-level Python-based web app can handle real-time video processing when optimized libraries like OpenCV are used under the hood.

To summarize the findings:

- The vehicle detection algorithm is **highly effective for counting** vehicles in a fixed scene, provided there is consistent motion. It has a small error rate and very few false detections in our evaluation. The approach is well-suited for scenarios like highway traffic monitoring or entrance/exit counting.
- The lane detection algorithm **works reliably in clear conditions**, offering a low-cost solution for lane departure warning or augmented reality lane overlay. Its performance

drops in adverse conditions, which is a known limitation of using simple edge detection; improving it might require additional techniques (e.g., adaptive thresholding, incorporating color filtering or temporal smoothing across frames).

- The entire system runs in real time on modest hardware, validating the efficiency of the chosen methods. This makes the solution attractive for deployment on devices that might not have GPUs or for integration into lightweight embedded platforms.

These results confirm that classical computer vision techniques, when carefully combined, can meet the needs of certain ADAS and traffic monitoring tasks. While they may not completely replace more advanced algorithms in all cases, they provide a foundation that is easy to implement, explain, and maintain.

7. Conclusion

In this paper, we presented a real-time vehicle detection and lane line detection system using a combination of classical image processing algorithms. Our implementation demonstrates that even without deep learning, one can achieve respectable performance in tasks like vehicle counting and lane identification by leveraging well-established techniques such as background subtraction, morphological filtering, edge detection, and Hough transforms. The system achieved about 95% accuracy in vehicle count on test videos and reliably marked lane lines in clear conditions, all while operating at real-time frame rates on standard computing hardware.The developed solution has practical implications for intelligent transportation systems. It can be deployed for traffic flow monitoring (counting vehicles to determine congestion levels), detecting incidents (sudden stops or unusual motions could be flagged), or as an ADAS feature to alert drivers when they stray from lanes. The use of a web application interface (Streamlit) further illustrates how such computer vision modules can be made accessible and user-friendly, allowing transportation officials or even researchers to easily test and utilize the functionality on their own video data.There are several avenues for future work to enhance the system. One improvement would be to fuse the vehicle and lane detection capabilities – for example, to not only count vehicles but also classify them or estimate their speeds using the detected lane markings as a reference for perspective. Another enhancement would be incorporating machine learning selectively: a lightweight classifier could be used to verify that detected moving blobs are truly vehicles (reducing any false positives), or a neural network-based segmentation could assist the lane detection under tough conditions (e.g., when edges alone are insufficient). Additionally, extending the system to handle multiple camera views or PTZ (pan-tilt-zoom) cameras would increase its applicability. We also plan to test the system in a variety of weather and lighting conditions, and introduce adaptive parameters (such as dynamically adjusting the binary threshold or Canny edge thresholds) to improve robustness.In conclusion, our paper shows that effective ADAS and traffic monitoring functions can be attained with relatively simple and computationally efficient image processing approaches. While modern deep learning methods offer superior accuracy, the trade-off in complexity and resource requirement means that classical methods still hold value, especially for real-time applications on the edge. We believe this work can serve as a foundation for further development of accessible, real-time traffic analysis tools and can be a stepping stone towards more advanced systems in the future.

References

1. Stauffer, C., & Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) (Vol. 2, pp. 246–252). https://doi.org/10.1109/CVPR.1999.784637
2. Viola, P., & Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 511–518). https://doi.org/10.1109/CVPR.2001.990517
3. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779–788). https://doi.org/10.1109/CVPR.2016.91
4. Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6), 679–698. https://doi.org/10.1109/TPAMI.1986.4767851
5. Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM, 15(1), 11–15. https://doi.org/10.1145/361237.361242
6. Pan, X., Shi, J., Luo, P., Wang, X., & Tang, X. (2018). Spatial as deep: Spatial CNN for traffic scene understanding. In Proceedings of the AAAI Conference on Artificial Intelligence, 32(1).
7. Javadzadeh, R., Banihashemi, E., & Hamidzadeh, J. (2015). Fast vehicle detection and counting using background subtraction technique and Prewitt edge detection. International Journal of Computer Science and Telecommunications, 6(10), 8–13.