Hindawi
Advances in Multimedia
Volume 2022, Article ID 3425295, 18 pages
https://doi.org/10.1155/2022/3425295

Research Article

Proposing Lane and Obstacle Detection Algorithm Using YOLO to Control Self-Driving Cars on Advanced Networks

Phat Nguyen Huu, Quyen Pham Thi, and Phuong Tong Thi Quynh

School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Hanoi, Vietnam

Correspondence should be addressed to Phat Nguyen Huu; phat.nguyenhuu@hust.edu.vn

Received 19 September 2021; Revised 27 April 2022; Accepted 1 May 2022; Published 30 May 2022

Academic Editor: Zhongxu Hu

Developing self-driving cars is an important foundation for the development of intelligent transportation systems with advanced telecommunications network infrastructure such as 6G networks. The paper mentions two main problems, namely, lane detection and obstacle detection (road signs, traffic lights, vehicles ahead, etc.) through image processing algorithms. To solve problems such as low detection accuracy of traditional image processing methods and poor real-time performance of methods based on deep learning methods, lane and object detection algorithm barriers for smart traffic are proposed. We first convert the distorting image caused by the camera and use a threshold algorithm for the lane detection algorithm. The image with a top-down view is then determined through the extraction of a region of interest and inverse perspective transform. Finally, we implement the sliding window method to determine pixels belonging to each lane and adapt it to a quadratic equation. YOLO algorithm is suitable for identifying many types of obstacles for identification problems. Finally, we use real-time videos and the TuSimple dataset to perform simulations for the proposed algorithm. The simulation results show that the accuracy of the proposal for detecting lanes is 97.91% and the processing time is 0.0021 seconds. The accuracy of the proposal for detecting obstacles is 81.90%, and the processing time is 0.022 seconds. Compared with the traditional image processing method, the average accuracy and execution time of the proposed method are 89.90% and 0.024 seconds, which is a strong antinoise ability. The results prove that the proposed algorithm can be deployed for self-driving car systems with a high processing speed of the advanced network.

1. Introduction

Transportation represents the prosperity and progress of a country. However, it also creates several serious problems such as accidents and traffic congestion. There were 3,206 traffic accidents nationwide killing 1672 people in the first quarter of 2021. Subjective causes related to vehicle drivers are drunkenness, fatigue, and inaccurate vehicle control. To reduce negative impacts and improve the efficiency of transportation, countries are developing smart systems, including infrastructure and vehicles, based on the basis of advanced networks such as 5G and 6G networks.

For smart cars, one of the most indispensable tasks is lane recognition and obstacle detection. They directly affect driving behavior. A driver can effectively steer a smart vehicle that provides its precise position on a road surface based on a lane. Obstacle recognition such as location and distance from other smart vehicles or animals on the road and recognition of objects such as signs or traffic lights significantly improve driving efficiency and safety. These jobs are performed through radio, sound, and light sensors such as RADAR and LiDAR. The authors [1] propose a lane detection and warning technique that uses a global positioning system (GPS) in combination with an inertial sensor and a highly accurate map. The authors [2] use a method to combine the LiDAR sensor and camera. The authors [3] present a LiDAR sensor that enables real-time car detection based on distance and light ray intensity information. These sensors measure parameters directly

with a small processing time. However, the resolution of GPS ranges from 10 to 15 meters, and the cost is high. In several outdoor environments, generating signals may interfere with each other, causing a reduction in system reliability.

Currently, software drives self-driving cars by tracking dozens of sensors and collecting data from a vehicle, radar, high-resolution cameras, GPS, and cloud services. The huge amount of data will then be transmitted to the center quickly using advanced 5G and 6G network infrastructure. The 5G network is currently in limited performance in a few parts of the world and promises many improvements such as faster speeds, higher connection density, lower latency, and energy savings. Theoretical speeds of 5G networks reach up to 20 Gbps, and 6G networks move towards terabit (Tbps) speeds that are several hundred to several thousand times faster than 5G networks. However, the goal of the 6G network is not only speed but also the remaining problems of the 5G networks. Four main directions of connectivity are being studied: smart connectivity (intelligent connectivity), deep connectivity, and holographic and ubiquitous connectivity. Currently, there are a lot of potential technologies that are considered for 6G networks such as optical wireless, quantum communications, unmanned aerial vehicles, and low-level satellites. Other technologies such as artificial intelligence (AI) and big data analytics are also included to support 6G networks to ensure network quality (QoS) goals.

Therefore, we propose an algorithm that uses a single front camera of a vehicle instead of using multiple cameras. The technique in [4] achieves stable results on the Caltech lane dataset. They introduce three techniques for lane detection, namely, using inverse perspective transformation to remove effects, setting HLS thresholds to filter color thresholds, and using the sliding window method to search for lanes.

However, proposed techniques face several difficulties in detecting lanes where they are not fully visible, resulting in changing color values or shadows. Several systems are developed based on an edge detection algorithm [5] to solve the cases that the color threshold is not able to handle. However, it causes a lot of noise. YOLO is popularly used due to many reasons with legitimate neural networks. One of the basic reasons is the speed of recognition in real time. The authors [6] have detected many objects based on a deep learning algorithm (YOLO). The results are ideal for obstacle detection problems. Therefore, we will use YOLO for the proposed algorithm in the paper.

Our paper has two key points as follows:

(i) Firstly, we propose to convert distorting images caused by the camera and use a threshold for edge detection for the lane algorithm. Images with a top-down perspective are determined through a region of interest (ROI) extraction and inverse perspective transformation. Finally, we implement the sliding window method to determine pixels belonging to each lane.

(ii) Secondly, we propose to use the YOLO algorithm for the obstacle identification problem. This is an advanced algorithm that is suitable for recognizing many types of obstacles as analyzed above.

The rest of the paper is presented as follows. In Section 2, we will present related work. In Sections 3 and 4, we present and evaluate the effectiveness of the proposed model, respectively. Finally, we give a conclusion in Section 5.

2. Related Work

Traffic congestion and accidents are very important problems, especially in urban areas and commercial centers. It causes not only a waste of time and money but also air pollution and health problems.

Smart cars have been a distant dream for a few decades. However, many experts and scholars or engineering corporations have been working on systems that solve these problems with the rapid increase in computer speed. Self-driving vehicles will revolutionize current society and reduce road deaths.

In particular, computer vision plays a central role in object recognition and tracking systems in future intelligent transportation systems.

The benefits of smart cars include the following:

(i) Smart cars can reduce traffic accidents caused by human-caused errors such as mental incompetence, drunkenness, or fatigue.

(ii) Thanks to smart cars, the disabled, elderly, or those without licenses can safely travel long distances.

(iii) Industry is subject to change. Goods can be delivered automatically, quickly, accurately, and cheaply, especially for long-distance orders.

(iv) Public transport is also planned such as taxis or buses to be replaced by smart cars that carry many passengers almost all the time. It will improve performance by minimizing receiving time and customer distance. It helps to improve urban spaces such as parking, traffic density, and congestion, thus reducing the impact on the environment.

Many computer vision studies have been done recently regarding lane detection based on features and models. Several traditional image processing models for road line detection are based on color and edge features [7], Hough transforms [8], and Kalman filter [9]. Identifying ROI has low processing time. However, it is not good since it is affected by many factors, including brightness, weather, and traffic conditions. The authors [10] introduce a new color space to replace RGB color space and analyze the distribution of road markings and pavements using Gaussian distribution to determine its color. The authors [11] focused on the effect of light changes on lane detection. The authors [12] used the Canny edge filter to find their borders. However, road surface conditions and obstacles have inadvertently created additional noise resulting in the performance of

the algorithm being much lower than a color threshold. Therefore, we need a suitable method of combining color threshold and edge.

In recent years, there have been many studies on unstructured lanes and curbs. The authors [13, 14] presented a vanishing point-based lane detection technique and achieved good results for detecting unstructured lanes. The authors [15] present an algorithm after obtaining pixels from extracting features that use direction, gray intensity, and starting and ending point positions of road markings to determine lanes. The authors [16] introduce the B-snake method using B-spline to describe lane shapes. All algorithms perform Hough transform [17] to compute probability and find the lane containing the largest number of pixels. This method has low reliability for certain curvilinear lane areas. Therefore, the sliding window method [4] is used on perspective transformed images that can search for quadratic lane equations with an accuracy of up to 84% on the KITTI dataset [18–21]. However, the sliding window method is highly dependent on conditions such as the environment and road surface that lead to obstructions or trees being misrepresented as lanes.

The authors [22] presented a system that can detect and classify lanes using a complex neural network with manually labeled ROIs from KITTI dataset for left and right lanes. The detection accuracy is high. However, image processing speed is slow since the algorithm takes too long to preprocess data. The authors [23] propose SegNet (a pixel classifier network) that not only segments and detects lanes but also classifies and receives pedestrians, trees, and buildings. However, the disadvantages of the network are complex network structure, large computational work, and poor real-time performance. The authors [24] present geometry-based spatial CNN (SCNN) that solves congesting lane problems. The results indicate that the research method has improved the lane recognition rate. However, the disadvantage of an algorithm is limited by different conditions. Color and feature-based detection only apply to scenes with clear roads and clear lanes under simple conditions. When the lane is damaged and complicated, accuracy drops significantly. Model-based detection is only suitable for preset situations, and algorithms have high complexity and large computation. It improves efficiency inaccuracy but requires high hardware and a more complex model.

Vehicle detection methods fall into three categories, namely, motion-based, feature-based, and template-based matching. The authors [17, 23] used background subtraction and mean background difference methods, respectively. However, this motion-based method is not suitable. When a vehicle moves, the background also changes, causing very serious errors. Tehrani Niknejad et al. combined Haar and AdaBoost to detect vehicles on highways [24]. The authors [4] proposed a vehicle detection method based on a histogram of oriented gradients (HOG) and support vector machines (SVMs) features for urban environments. Two methods have improved accuracy significantly. However, the traditional machine learning method only supports training for a small amount of data since it has bad results for obstacle detection, especially when detecting many layers. The sliding window technique is also slow compared to more recent modern algorithms. Currently, deep learning methods are mainly two- and single-stage. Two-stage networks such as Fast R-CNN [25], Faster R-CNN, and Mask R-CNN usually have high accuracy. However, they have high complexity and long computation time. Single-stage networks such as YOLO and SSD speeds have been greatly improved. However, the results are not reliable. Therefore, deep learning using legitimate neural networks [6] outperforms other techniques. It can learn specific shapes (like cars, pedestrians, traffic lights, animals, etc.) and use more data. It is also harder to train. However, the results are better, and the processing time is also shorter than that of using SVM. It can strike a balance between accuracy and real-time performance.

Controlling self-driving cars on advanced networks such as 4G, 5G, and 6G networks has been studied in [26, 27]. In [26], the author studied the 6G-assisted cooperative driving mode, which is an advanced driving mode through information sharing and driving coordination using the V2V method. Simulation results show that proposed algorithms significantly improve road safety, capacity, and efficiency. The authors [28] use a typical video data stream to control car features through 5G video enabling a remote driving system. The results show that it is a possible direction for future research. In [29], the authors simulate cars driven by a mobile edge cloud placed in a Nokia simulator on top of a 5G network. The setup accurately simulates the ecosystem of future connecting traffic scenarios based on Xbox Kinect, in-vehicle infrared, and ultrasonic sensors. In [27], the authors present a test of connecting cars on the basis of 5G networks. They address latency in classically connecting network-physical systems by using a mobile edge cloud in a 5G interface that is fully emulated in cmWave. In [30], the authors present the design of customized 5G network sections to improve mobility, traffic safety, and road comfort. The proposed solutions involve zoning of core network and radio access network resources as well as the functional configuration of terminals for self-driving cars.

Based on the above analysis, the object feature-based image processing method to classify lane pixels shows efficiency in many types of environments and fast processing time. However, it needs to select and combine color and morphological characteristics to optimize and limit the generation of additional noise. The sliding window algorithm performs to find around the previous frame lane that can reduce processing speed and increase reliability. Deep learning is outperforming a lot of techniques such as computer vision and traditional machine learning for obstacle recognition problems especially when it consists of many types. Therefore, we use the sliding window method for the proposed system when finding obstacles. More details will be presented in Section 3. To the best of our knowledge, a system of lane determination using YOLO and obstacle detection using a sliding window has not been published in any previous literature.

3. Proposed System

3.1. Overview of System. The diagram of the system is shown in Figure 1.

The system consists of two blocks that perform two functions, namely, lane detection and obstacle recognition.

(i) Obstacle detection: objects, including vehicles ahead, animals, or traffic signs, are common obstacles on the road when participating in traffic and directly affect the accuracy of lane detection such as crossing the line street or obstructing the camera view. The proposed purpose of this module is to detect and classify the obstacles ahead obtained by the camera. Therefore, we come up with methods of handling obstacles such as removing obstacles to improve lane detection or providing warnings of obstacles ahead for smart self-driving cars.

(ii) Lane detection: the reason why we recommend the lane detection module is to issue warnings of violations while participating in highway traffic for self-driving cars such as encroaching on the lane, going in the wrong lane, or crossing the line street.

The results of the obstacle recognition algorithm are used to improve lane problems. The vehicle detection block uses the pretraining model YOLO to recognize a variety of obstacles. Line detection block performs chromatic feature extraction primarily and incorporates edge detection only when color threshold gives bad results. Performing to find neighborhood sliding windows is to reduce the false-positive rate and increase the accuracy of the lane.

3.2. Lane Detection. Lane boundary detection is important for determining the left and right positions of the real-world driveway. In this lane determination, our algorithm is divided into three stages of implementation including the following.

3.2.1. Stage 1: Removing Image Distortion. Input image before processing is recorded by the front camera on the vehicle. Although the process of recording images is fast, distortions still appear. This distortion alters several parameters such as actual size, the shape of the lane, vehicles, or the actual environment. These changes cause errors in judgment of direction and position or lane curvature. Therefore, distortion correction is necessary. In this section, image distortion removal is performed in two steps, namely, camera correction and image distortion removal.

The model of lane detection algorithm is shown in Figure 2. Lane detection algorithm that we develop includes the following steps:

(i) *Step 1.1.* Calibrating camera and correcting image distortion.

(ii) *Step 1.2.* Creating a binary image using the lane feature.

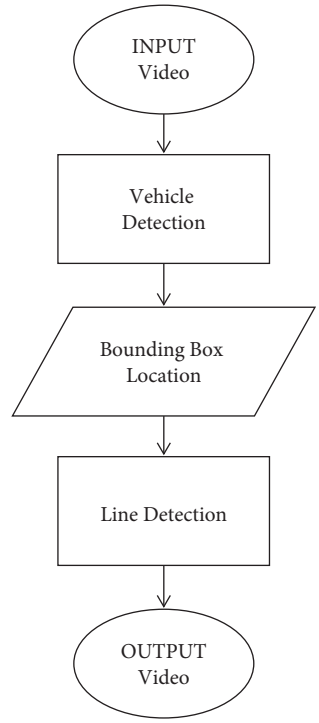


Figure 1: An overview of the proposed system model.

(iii) *Step 1.3.* Perspective conversion (for a top-down view of the road).

(iv) *Step 1.4.* Locating lane.

(v) *Step 1.5.* Drawing showing the lane position on the original image.

In this step, the system takes video from the front camera of the vehicle. Output data is a video that contains a bounded area between two lanes. Details of stages are presented as follows.

The calibration process requires a series of chessboard images taken from multiple directions, as shown in Figure 3.

In this paper, we use OpenCV function findChessboardCorners () to automatically find and chart chessboard. We next use functions cv2.calibrateCamera () and cv2.undistort () to calculate and correct.

From the resulting camera correction, we remove image distortion using the cv2 function. The results are shown in Figure 4.

3.2.2. Phase 2: Analyzing Lane Area. In the phase, we perform the following steps:

(1) Step 2.1. Edge Detection. In this step, we have an image without distortion. We next perform filtering to identify lines on the road through their characteristics. We use edge detection algorithms, including methods such as Canny, Sobel, and Prewitt. However, traditional edge detection algorithms have many problems such as wide detection range, high noise, and long computation time.

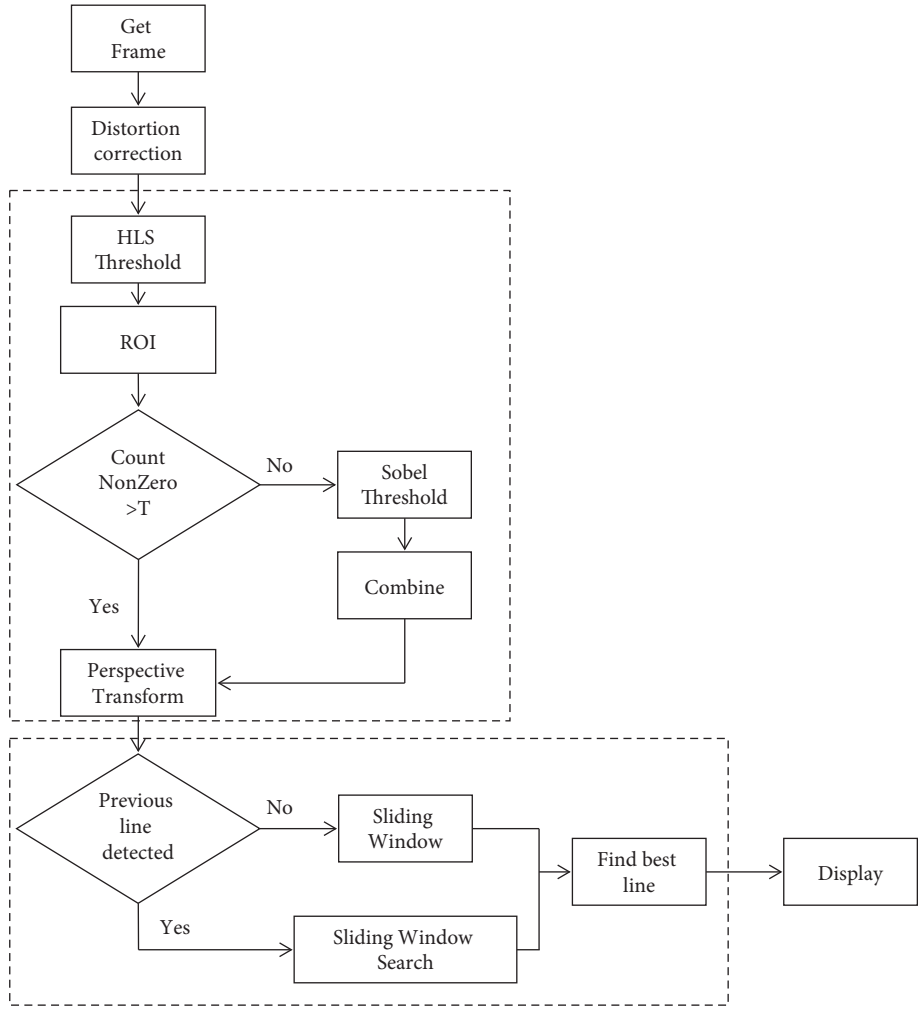


Figure 2: Diagram of the lane detection algorithm.

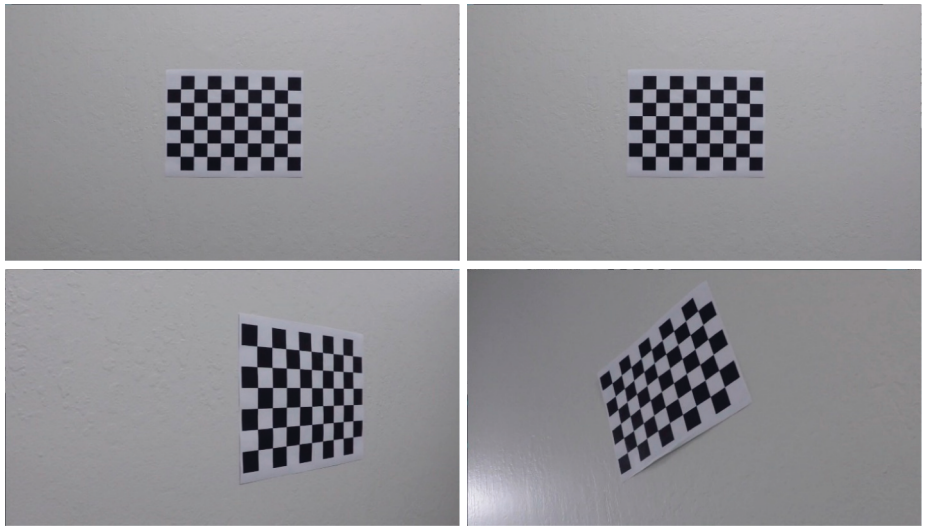


Figure 3: Illustrating image of chessboard.

Therefore, we choose to use Sobel and color operators. The idea of the algorithm is to make lanes separate by color and enhance accuracy by edge detection.

We try to combine different algorithms of color threshold and gradient to create a binary image where lanes can be segregated. There are many ways that good results

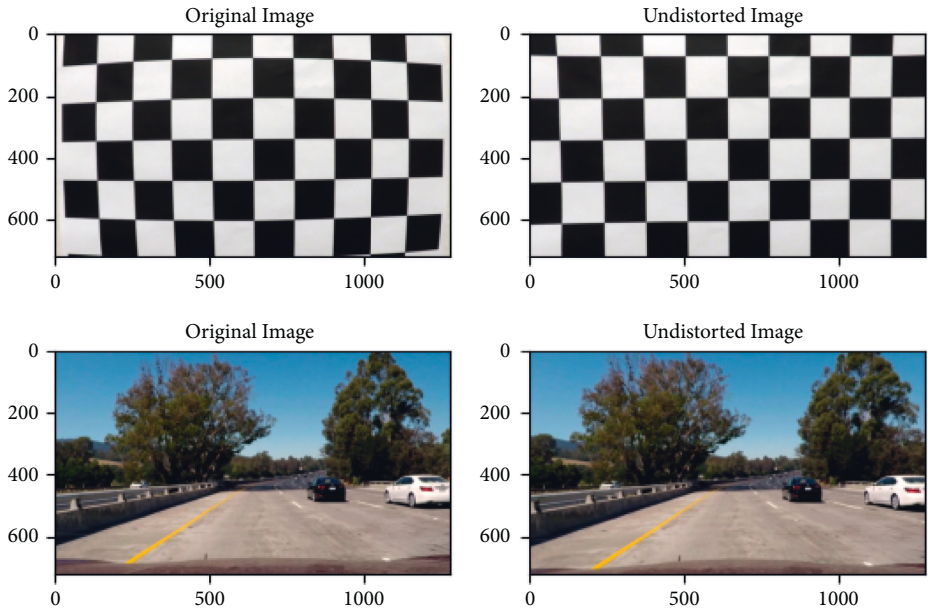


Figure 4: Result after removing image distortion.

can be achieved. In this paper, we found that the best results are obtained using a color threshold. We experiment with color space channels such as GRB, HSV, and Hue Saturation Lightness (HSL) while using the color picker technique. Compared with RGB color space, HSL color can better reflect image characteristics. Each color channel of the HSL system can be used separately. The HLS color channel is used to handle cases when the color of the line is too light or light. The L channel threshold (lightness) will reduce edges formed from shadows in the frame. The S (saturation) channel threshold will widen the white or yellow lane, and the H channel threshold (Hue) is used to represent lane color.

The best results are the S channel to filter white and the L channel to filter yellow lanes. We can see that lanes are clearly defined as shown in Figure 5.

We find that the results of the color threshold selection algorithm are the best based on test results. Lanes are blurred when the light is unevenly distributed. The road surface is white when the color threshold does not work as well as shown in Figure 6.

We try to improve the algorithm using methods such as light balance or gamma correction and choose to use the Sobel operator in the X-direction. The Sobel operator is a first-order differential operator. It calculates the gradient of a pixel using the gradient of a neighborhood of the pixel and then selects according to the given absolute value. The essence of this method is to reflect the gray difference characteristics of adjacent pixels. The operator consists of a set of horizontal matrices and a set of 3×3 vertical matrices that convolve the image in the plane as shown in (1) and (2). The obtained results are approximate values of the horizontal and vertical luminance differences, respectively. Its form is the filter operator for edge splitting using a fast convolution function. This method is simple and effective since it is widely used.

If we consider A, G_x, and GG_y y as the original image, vertical, and horizontal edge detection, respectively, the formula is expressed as follows:

$$G_x = \begin{bmatrix} -10+1 \\ -20+2 \\ -10+1 \end{bmatrix} * A, \tag{1}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A. \tag{2}$$

At each point in the image, gradient approximations are obtained using

$$G = \sqrt{G_x^2 + G_y^2}. \tag{3}$$

Based on the shape of the lane through the camera, it is usually a vertical line or a small deviation relative to the vertical direction. The Sobel operator performs edge detection on vertical and horizontal images and can remove some noise. However, we have analyzed above that the edge detection operator produces a lot of bad noise and low accuracy. Therefore, this problem needs to be solved effectively. We found the color threshold to be the best. For the above inefficient cases where the number of detected lane pixels is not enough, we will perform color threshold matching with the Sobel. Therefore, we will calculate the number of lane pixels and still perform color threshold when recognition is good. The rest will combine with edge detection. The results are shown in Figure 7.

Figure 7 shows that the edge detection and threshold superposition algorithm achieves good results with unbroken detectable lanes.

Figure 5: An example of the HLS threshold.

Figure 6: Results of color threshold algorithm.

(2) Step 2.2. Reversing Perspective Transformation. Perspective transformations are useful for simpler spaces. The parallel lanes merge into a point of distance known as a vanishing point. The closer the distance between two adjacent lanes, the lower the vanishing point that is detrimental to lane identification or object distance, as shown in Figure 8.

The goal of the algorithm is to turn a trapezoidal array of the road ahead into a rectangle in accordance with reality and restore the parallel relationship of lanes, as shown in Figure 9.

To transform perspective to birds-eye mode, it is necessary to define a rectangle on the original plane. These four points are called source points. Predefining four rectangular points on a new image is the position of destination where we want to stretch the trapezoid. When we have two sets, we can use the getPerspectiveTransform() function to calculate the transform matrix perspective. When there is a transformation matrix, we use the warpPerspective() function to convert perspective to a birds-eye system. For a single camera, we have an image with a size of 1280×720 and destination point parameters as shown in Table 1.

The aerial image of the lane is obtained from perspective transformation as shown in Figure 10. The result shows that the lane is more clearly represented and is not distorted.

3.2.3. Stage 3: Detecting Road Markings. We can easily see lane markings and significantly eliminate environmental noise. In the next step, we need to locate the lane and find a quadratic equation that fits defining lane. The process includes the following steps:

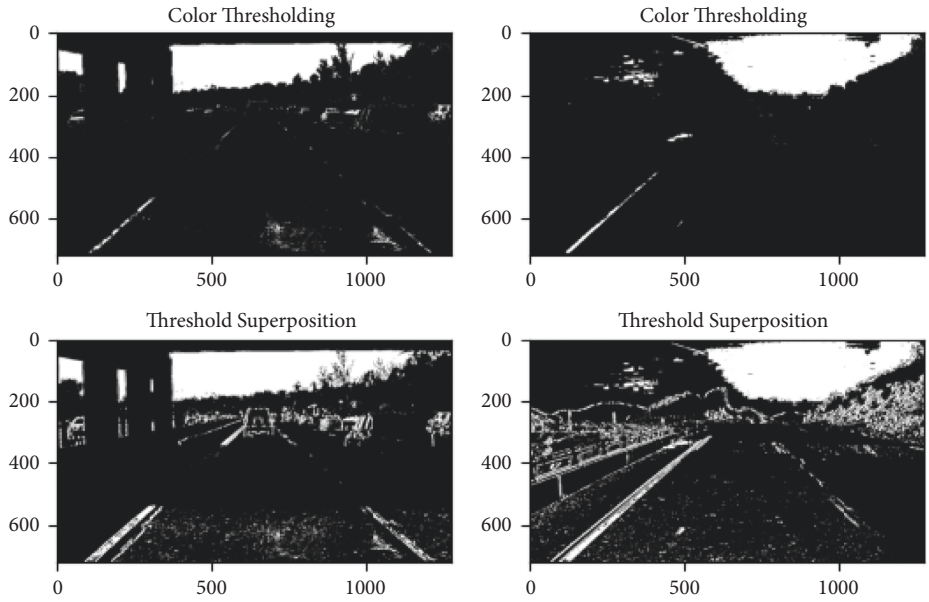


Figure 7: Results of threshold color and superposition.

Figure 8: Lane perspective results.

(i) Determining the bottom position of the image where the lane line starts using the pixel intensity distribution histogram method.

(ii) Finding cumulative slices of the lane at the bottom of the image horizontally (sliding window method).

(iii) Identifying nonzero pixels in terms of x and y of sliding window and adjusting to find a quadratic polynomial that fits those points (polyfit polynomial).

The detailed steps are as follows:

(1) Step 3.1. Finding Peak of Histogram Method. We get a binary image with two distinct independent lanes after generating a binary image from the color threshold and perspective transformation for the lane image. We need to determine the lane of pixels that are left and right. We first create a histogram along with all columns in the bottom half of the image (color intensity distribution over the bottom half) as shown in Figure 11.

We add pixels along each column of the image (count number of nonzero pixels along the x-axis) with the graph above. In a binary image from color threshold (pixels have values 0 or 1), the two highest peaks of the histogram will give a good representation of the x position of lane basis. We can use the value to find lines.

(2) Step 3.2. Slide Window. The cumulative search algorithm will perform in a horizontal slice of an image to determine the window position. Sliding windows are used to accumulate from the bottom up. When the number of pixels in the sliding window exceeds the set threshold number, the mean is taken as the center of the next sliding window.

We divide the image height into 9 parts with 9 windows that translate from bottom to top based on left and right separation. The window center of the first window block corresponding to the end position of the image is determined by the histogram method. A window width of 100 pixels is used to find pixels other than zero. If the cumulative number of pixels is less than 50, the window will be redefined.

In Figure 12, pixels corresponding to the left lane are denoted in blue, and points of the right lane are denoted by yellow representing lane segregation.

(3) Step 3.3. Searching Slide Window. The purpose we propose the sliding window search method is to find the pixels belonging to each line. We have chosen the sliding window size to be 100 pixels for the reasons as follows.

When we compare neighboring frames, lane position does not change significantly. Therefore, we only need to find around the previous frame lane with the interval dataset to 100 pixels instead of letting the window slide across the entire horizontal line. This leads to many significant benefits as significantly reducing processing speed, removing several noises that are not able to be filtered by edge and color threshold, and improving reliability.

Using a large window size will cause processing time because the pixel search area will be larger. Besides, we also encounter some noise on the road that will affect the accuracy of lane detection.

If the window size is too small, it will make it difficult to find pixels because there will be cases where the pixels to be searched are out of the search range. This both reduces accuracy and causes a loss of processing time. When the

Figure 9: Result of the image of the lane.

Table 1: Perspective coordinates.

Source	Destination
(535, 300)	(200, 0)
(750, 300)	(1000, 0)
(60, 720)	(200, 720)
(1260, 720)	(1000, 720)

Figure 10: Lane mask after reversing perspective transformation.

cumulative number of pixels is too small, the window will have to be redefined many times accordingly.

We have done resizing the sliding window. The results show that a window size of 100 pixels gives the best results in lane detection. The process details are described in Section 4.

(4) Step 3.4. Linear Regression. We need a quadratic equation that is compatible with lane points. Using the polyfit function with $p = \text{polyfit}(x, y, 2)$, we can find three parameters A, B, C of equation $A.x^2 + B.x + C$ as shown in Figure 13.

(5) Step 3.5. Finding the Best Line. We take two measures to avoid errors and eliminate bad lane equations.

We first track previous frames to find similarities. Therefore, we perform the quadratic coefficient of triple tolerance. The tolerance consists of three values 0.001, 1, and

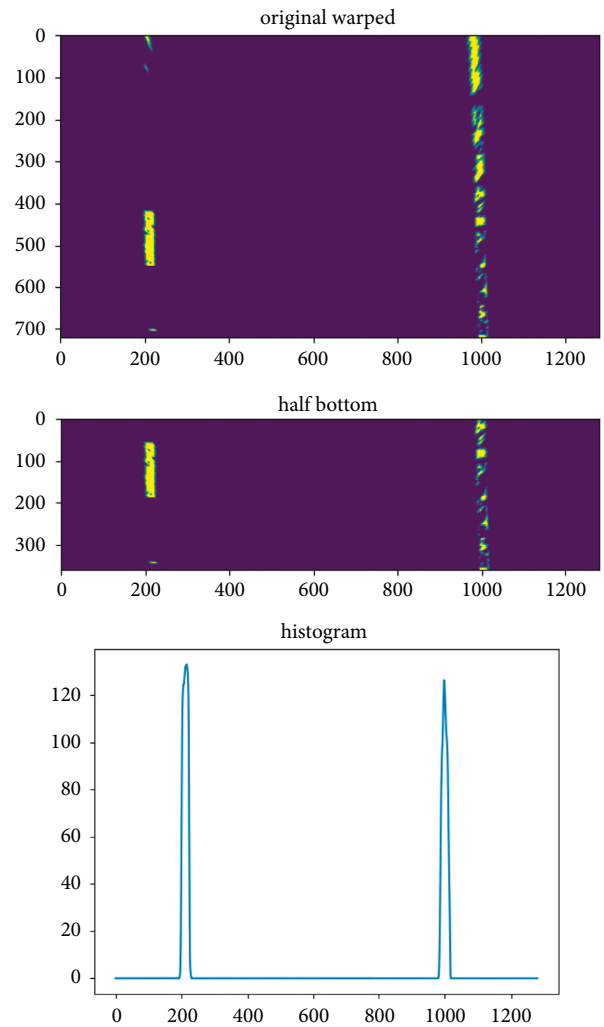


Figure 11: Finding peak of graph.

100. If more than one factor exceeds the limit, the equation will be equal to an average of five previous frames. We set the parameter as lane.detected = None.

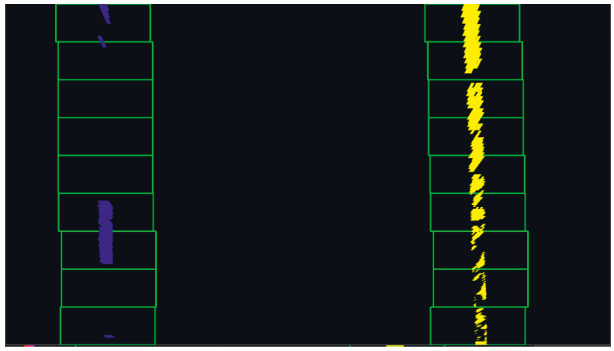


Figure 12: Result of sliding window.

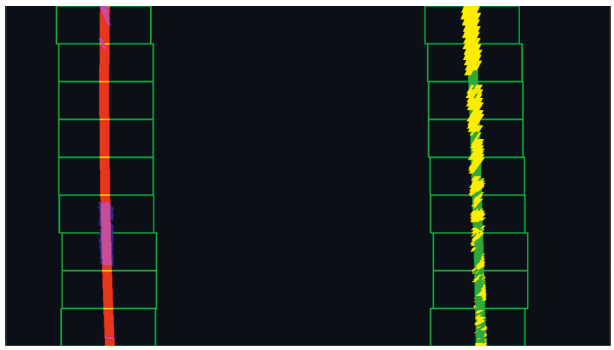


Figure 13: Result of linear regression.

Second, we remove the lane equation if the polyfit is faulty (when there are not enough nonzero pixels on the x-axis) or the average width between two lanes is greater than 800 or less than 300 pixels. The lane equation is then equal to the previous frame, and the parameter is lane.detected = None.

When lane.detected = None, we perform the sliding window again with the next frame instead of checking the entire window.

(6) Step 3.5. Displaying Lane Position. When we know the position of lane, we use the polyfit function to draw an area of the image (bounding area between two lanes). We then need to rotate the image area to the original perspective and merge it with color image. The result is shown in Figure 14.

3.3. Obstacle Detection

3.3.1. YOLOv4 Overview. YOLOv4 is a single-stage detector that effectively classifies and localizes objects in images with a single pass. The YOLOv4 model architecture can be depicted as Figure 15.

The backbone, CSPDarknet53, is a modification of the DarkNet-53 network used in YOLOv3. There are 53 convolutional layers in the network using a cross-stage-partial network.

The neck includes a path aggregation network (PAN) along with an implementation of spatial pyramid pooling (SPP). The former works as a method of aggregating parameters from different backbone levels instead of using the feature pyramid network (FPN) used in YOLOv3. SPP significantly increases the receiving field. It separates the most important contextual features and does not slow down network performance.

The head, YOLOv3, is used as the end of the string object detector for dense prediction and detection. YOLOv4 detects aerial objects effectively thanks to several innovative methods such as a bag of freebies and a bag of specials integrated into the model. YOLOv4 accurately predicts small objects using the CioU loss function to reduce the center point difference between the bounding box and the ground. The CutMix, Mosaic, and self-adversarial training techniques increase the robustness of the algorithm compared to previous YOLO versions. The implementation of methods such as CSP, Multi-input Weighted Residual Connection (MiWRC), and PAN reduces the speed and cost of computation. These are important for real-time object detection. Therefore, we choose the YOLOv4 model for obstacle detection because it is an effective model. It balances accuracy and real time by the requirements of the problem.

3.3.2. Obstacle Detection Algorithm. Algorithmic tasks include one or more objects and classify objects of an image. Before creating our model, we used a pretrained model with 80 classes of COCO dataset [32]. Since types of obstacles on the road can be cars, traffic lights, pedestrians, motorbikes, bicycles, or animals, we used a pretrained model as shown in Figure 16. Details of the process are as follows:

(i) *Step 1.* Pretraining the YOLOv4 model.

The input image of the neural network must be of a certain form as a blob. When a frame is read from a video stream, it is passed through the blab From Image function to be converted into an input blob for a neural network. The process also converts pixel values ranging from 0 to 1 and resizes an image to $(416, 416)$.

Blob is then performed forwarding over the YOLO network. YOLOv4 algorithm generates bounding boxes as predicting detection results.

(ii) *Step 2.* Threshold filtering and nonmaximum suppression.

Each output bounding box of the network is represented by a vector consisting of class and five elements center_x, center_y, width, height, and confidence box surrounding object. The output of the YOLO algorithm is multiple boxes. However, most of the boxes are redundant. Therefore, it is necessary to filter and remove them.

In the first step, if there is a low probability of object detection, a box will be discarded. We only keep those boxes with probability greater than defining threshold. The remaining boxes will perform

Figure 14: Result of lane detection.

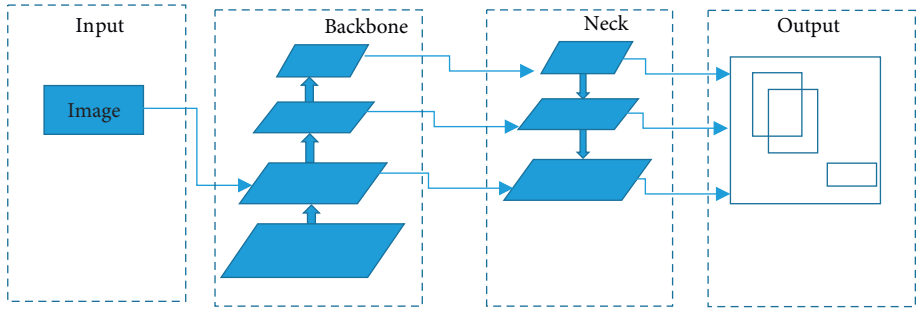


Figure 15: YOLO: single-stage architecture [31].

nonmax suppression. This reduces the number of overlapping boxes.

(iii) *Step 3.* Combining lane detection algorithm.

The cause of wrong lane identification is mainly due to blurred lanes, lost track, and interference caused by obstacles on the road. Therefore, it would be good to take advantage of the result of obstacle detection for the lane detection algorithm. The results are shown in Figure 17.

4. Simulation and Result

4.1. Simulation Setup. We perform algorithm simulation under the following conditions:

(i) Software environment: Windows 10 64-bit operating system, Python 3.7.2 64 bits, and OpenCV 4.1.0.

(ii) Hardware environment: Intel Core i5-9300H processor 2.4 GHz CPU.

(iii) The maximum length of recognizable lanes is 32 m on expressways.

To perform a proposed algorithm for different environments, we collect the TuSimple dataset (standard dataset for detecting lanes) and ours. TuSimple dataset consists of 3626 training and 2782 testing videos. Each sequence consists of 20 frames taken consecutively for one second with dimensions of 1280×720 pixels. Images include various weather conditions such as cloudy, clear sky, traffic environments, and lane conditions.

Details of simulation parameters are shown in Table 2.

Besides, we test the lane detection module on some images taken from the KITTI dataset. The simulation is performed on an operating system of Intel(R) Core(TM) i3-6100U CPU @ 2.30 GHz, 8GB RAM, and Ubuntu 20.04; the results are described in detail in Table 3.

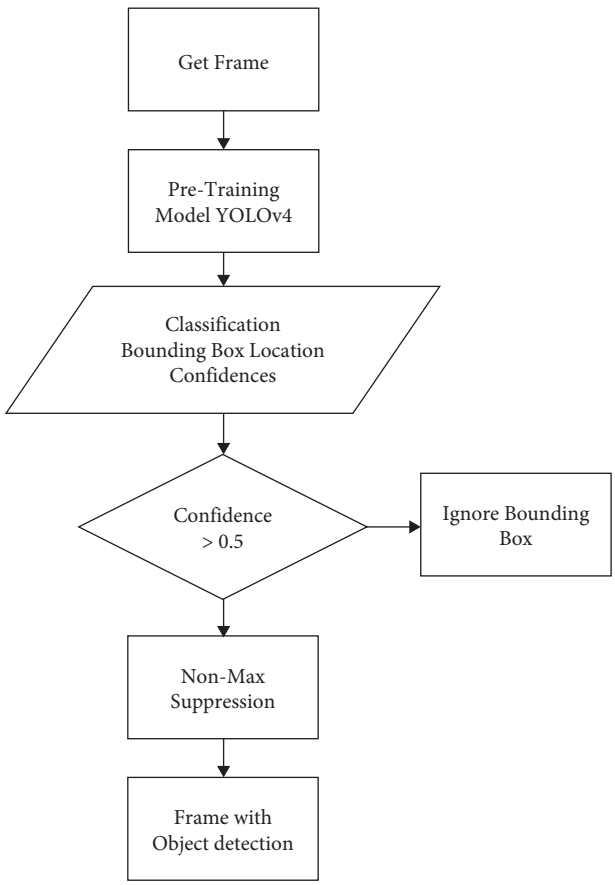


Figure 16: Diagram of obstacle identification algorithm.

4.2. Result. We perform a simulation with the above conditions, and the results are shown as follows.

4.2.1. Lane Detection. First, the results of lane detection under different conditions are as shown in Figure 18 for TuSimple dataset and Figures 19 and 20 for KITTI dataset.

In Figure 18, we see that the lanes are well recognized with areas that are highlighted in blue. To perform with the TuSimple dataset, we get the results as shown in Table 4.

It can be seen that the developing algorithm performs well under different conditions with the TuSimple dataset. The average recognition accuracy rate received is 97.91%, and the processing time is 85 milliseconds. To perform with videos on the highway above, the accuracy has decreased. However, the proposed algorithm still has good recognition performance for the TuSimple dataset. The results show that proposal has high accuracy and adaptability.

Figure 19 shows the accurate results of lane detection in low light and normal conditions on the KITTI dataset. Figure 20 shows some false lane detection results due to the ROI region selection not matching the lane curvature and camera rotation. The accuracy of our proposed method is estimated on 195 frames extracted from the KITTI dataset. In the paper, there are 166 frames of true lane detection and 29 frames of false lane detection.

The performance of the proposed method is illustrated in Table 3. In Table 3, the accuracy and processing time are 85.13% and 0.086 second/1 frame, respectively.

We next compare the proposed algorithm performance with others for lane recognition. The results are shown in Table 5.

In Table 5, we perform the proposal using HSL and Sobel filter and sliding window search (SWS) algorithm for lane detection on TuSimple and KITTI datasets. Compared with other algorithms, the SWS algorithm has a low average processing time. The algorithm is a first-order line detection algorithm since it is almost impossible to apply to lanes with certain curvatures that cause lower accuracy than our method by 2.21%. Besides, there are DL methods such as FastDraw ResNet training or ConvLSTM networking based on the dataset and using an Intel Core i5 9300H CPU hardware environment that has high accuracy.

The average processing speed is very slow with these algorithms (0.96 frames/s), which is not suitable for real-time problems due to complex models and time processing. Besides, it causes serious errors for several conditions (mountain roads) due to insufficient training data. In conditions such as mountain roads or under bridges, it causes false-positive cases due to insufficient training data. Therefore, the accuracy is 2.91% lower than the proposed algorithm. On the KITTI dataset, our proposed algorithm using Intel Core i3-6100U, CPU@ 2.3 GHz, gives an accuracy of 85.13% greater than the traditional method of 1.13%. However, the processing time is slower.

In Table 5, the proposed algorithm achieves good performance with the same dataset with an accuracy recognition rate up to 97.91% for less than 10 milliseconds. It helps to enhance driving safety for the real driving environments of smart vehicles.

4.2.2. Sliding Window Size Selection. To understand the influence of window size on correct lane detection, we performed window resizing on a small dataset of 100 videos corresponding to 2000 frames extracted from the TuSimple dataset. The window sizes tested were 50, 100, 150, 200, and 300 pixels, respectively. The experimental process is performed on an Intel(R) Core(TM) i3-6100U CPU @ 2.30 GHz, 8GB RAM, and Ubuntu 20.04 operating system. The results are shown in Table 6.

Table 6 shows that using a window size of 100 pixels gives the highest accuracy of 96.40%. The algorithm has a lower accuracy of 93.15% and 91.25% with a minimum window size of 50 pixels and a maximum of 300 pixels, respectively. The larger the window size, the slower the processing time and the reduced accuracy since the larger the area, the longer the search time and the more the noise on the road. Therefore, a window size of 100 pixels is a reasonable choice as it ensures high accuracy and suitable processing time.

The results are illustrated in Figures 21 and 22. In Figure 21, the results show that choosing a window size that is too small or too large will lead to incorrect lane

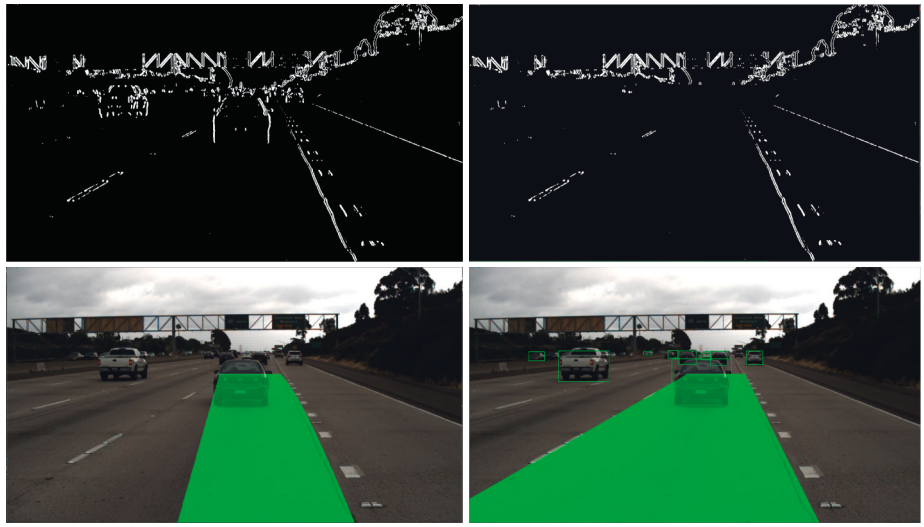


Figure 17: Using obstacle detection results to improve lane detection algorithms.

Table 2: The parameters of the simulation system.

No.	Parameter	Values
1	Road type	Limited access highway, mountain road, and urban road
2	Vehicle	Mixed traffic (car, pedestrian, traffic lights, etc.)
3	Weather conditions	Dry, clear, rain, and cloud
4	Camera	Camera HD-RGB, resolution 1280×720, and JPEG format
5	Roadway surface conditions	Dry, wet, and undamaged
6	Average speed	1.335 seconds
7	Processor	Intel Core i5-9300H CPU 2.4 GHz
8	The maximum length of detected lane	32 meters
9	YOLOv4	Input: 416×416; and output: 80 classes
10	Software	Windows 10 64 bits, Python 3.7 bits, and OpenCV 4.1.0
11	TuSimple dataset	3626 training and 2782 testing videos
12	Lane width	$0 \div 4$ meters

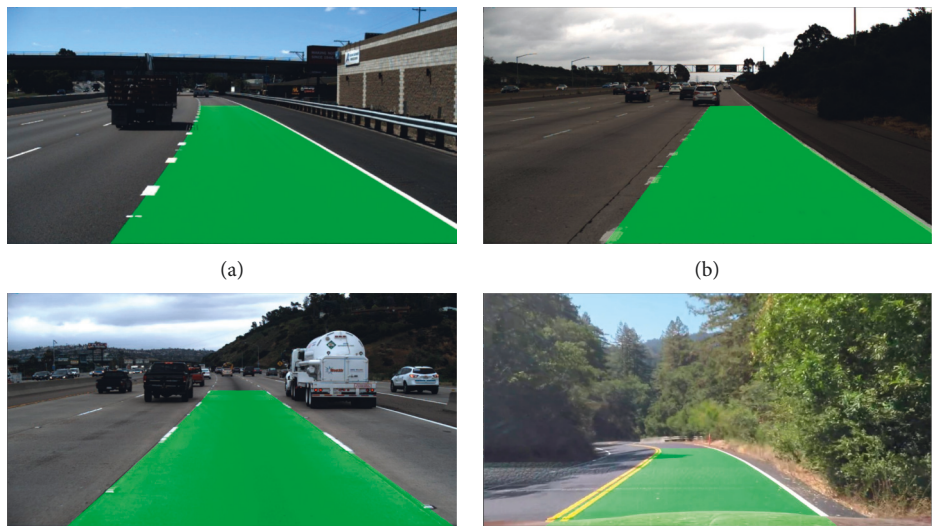
Table 3: Results of lane detection for different conditions with KITTI dataset.

Parameter	Valuable
Total number of frames (frames)	195
Correct identifiers (frames)	166
Incorrect identifiers (frames)	29
Average accuracy (%)	85.13
Average processing time (second)	0.086

recognition due to interference such as wheelchair marks or sudden changes in light.

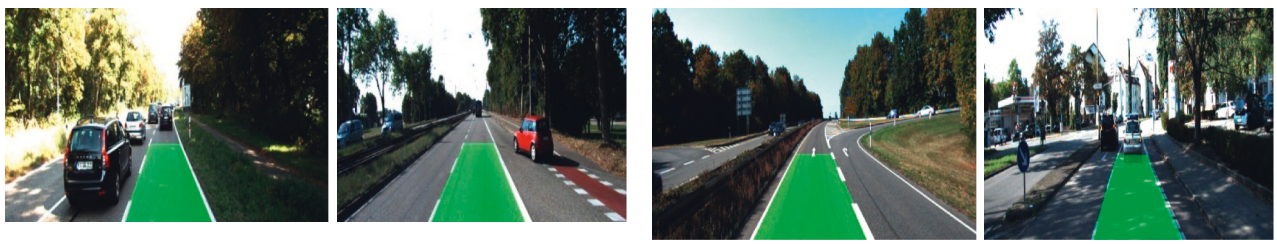
4.2.3. Obstacle Detection. We perform the algorithm on the same TuSimple dataset and compare it with others. A wide variety of obstacles can be identified with the 80-layer pretrained YOLO model. The average accuracy of mAP for individual layers is 80.70%, 98.20%, 75.06%, and 42.49% for cars, stop signs, traffic lights, and traffic signs, respectively. When we compare it with a system using HOG and linear SVM, we see that it can only recognize the presence of cars. Besides, forwarding the entire image over the network is more expensive than extracting the feature vector via SVM. We only need to switch over the network once for YOLO, and it is about 150 times for SVM and HOG. Therefore, YOLO is 10 times faster than SVM and HOG. We set up the confidence level at 50% in the paper. Compared with YOLOv4, YOLOv3 has lower accuracy for subjects close to the camera. However, the receiving distance is better. The performance results are shown in Table 7.

In the system using HOG and SVM features, the image is scanned with a sliding window. The feature vector is calculated and put into the classifier with each window. An image is detected with about 150 windows to discover cars. In the YOLO algorithm, we do not have any window since the detection is performed quickly. Each frame is transferred to the network and is processed exactly one time. Transitioning the entire image over a network will take more costs than extracting characteristics of window and SVM. However, this is only performed once time instead of 150 times with SVM and HOG. Therefore, the average processing time is also much lower than that of SVM and HOG. Besides, there are many other layers such as people, dogs, and cats that the ratio of false positives or negatives is much better. As a result, the accuracy of YOLO is 24.1% higher than HOG and SVM.



(a) (b) (c) (d)

Figure 18: Results of lane detection for different conditions: (a) sunny, (b) cloudy weather, (c) traffic, and (d) mountain road.



(a) (b)

Figure 19: Results of lane detection for different conditions: (a) shade and (b) good lighting in the KITTI dataset.

(a) (b)

Figure 20: Several wrong lane detection results for different conditions: (a) shade and (b) good lighting due to inappropriate ROI.

Table 4: Results of lane detection for different conditions with TuSimple dataset.

No.	Condition	Total number of videos	Correct identifiers	Incorrect identifiers	Accuracy rate of identification (%)
1	Daytime	524	519	5	99.04
2	Weather	625	612	13	97.92
3	Traffic environment	918	906	12	98.69
4	Road surface condition	715	687	28	96.08
5	Total	2782	2733	58	97.91

All object detection algorithms such as MS-CNN and Faster R-CNN use regions to zone the object of an image without looking entire frame. It will look at parts of the image capable of containing high objects. CNN models discover in two stages, namely, suggesting areas and classifications. First, it proposes an area of interest using a selective search algorithm that returns about 2000 zones for each image. It then classifies by applying CNN to all boxes.

Table 5: Result of comparison lane detection algorithm.

Method	Algorithm	Average accuracy (%)	Average processing time (second)	Environment system
Traditional method	Hough transform [33]	95.70	0.06540	Intel Core i7-6700K CPU@ 4 GHz
Traditional method	RANSAC + HSV [34]	86.21	0.50000	Intel Core i7-4700 CPU@ 2.40 GHz
Horizontal filter + Otsu	[35]	83.00	0.013	CPU Intel 3.30 GHz
Traditional method	Sliding window KITTI [4]	84.00	0.00318	Intel Core i5 5200U CPU@ 2.20 GHz
Deep learning	FastDraw ResNet [36]	95.00	0.06533	NVIDIA GeForce GTX 1080, GPU
Our proposal	HSL + Sobel filter + SWS KITTI [4]	85.13	0.08620	Intel Core i3-6100U, CPU@ 2.3 GHz
Our proposal	**HSL + Sobel filter + SWS TuSimple**	**97.91**	**0.08500**	**Intel Core i5-9300H, CPU@ 2.4 GHz**
Our proposal	**HSL + Sobel filter + SWS TuSimple**	**97.91**	**0.0021**	**GPU GeForce GTX TITAN X**

Table 6: Results of lane detection for different size of window with the TuSimple dataset.

No.	Size of window (pixels)	Correct identifiers	Incorrect identifiers	Average accuracy (%)	Average processing time (second)
1	50	1863	137	93.15	0.0855
2	100	1928	72	96.40	0.0872
3	150	1916	84	95.80	0.0922
4	200	1891	109	94.55	0.0942
5	300	1825	175	91.25	0.0963

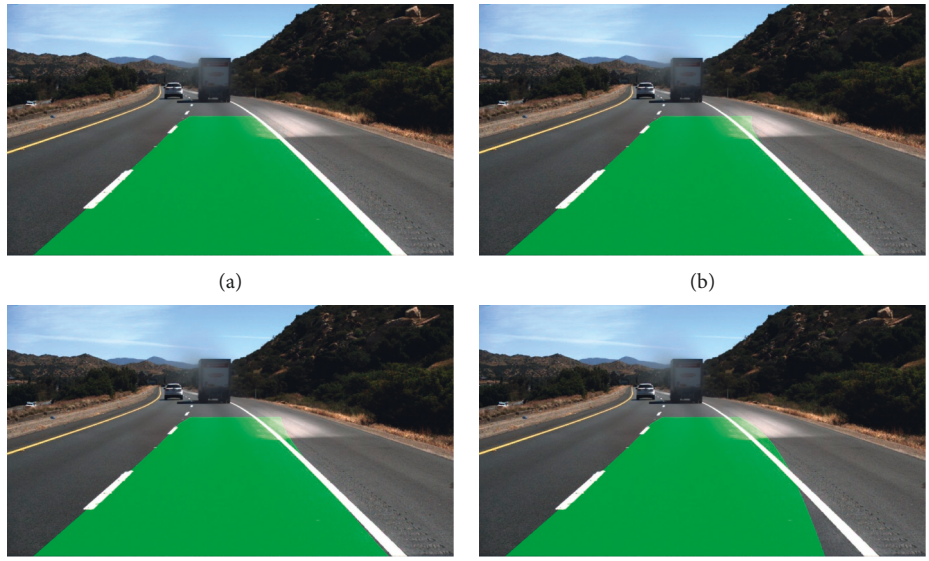
(a) (b)

Figure 21: Result of lane detection with window sizes of (a) 50 pixels and (b) and 100 pixels in the TuSimple dataset.

Although the number of calculations increases and the processing speed is very slow, it gains high accuracy with 85.05% for Faster R-CNN and 81.6% for R-CNN. However, this algorithm is difficult to implement in real-time conditions.

In the above algorithms, the system initially performs calculations of control parameters such as steering angle, distance, and navigation. The method of lane feature extraction by color threshold or edge detection is still large, and its performance is low for mountain roads or continuously changing light. Besides, characteristics of possible obstacles on the road are also factored into consideration when we retrain with the YOLO model.

From the above results, we see that the proposed system can detect lanes on a 2.4 GHz CPU which takes about 85 ms with each image, and the vehicle detection time is about 0.91 s. The total time for lane detection and vehicle detection is about 0.995 s on CPU hardware and 0.02405 s on GeForce GTX TITAN X GPU, respectively. The average accuracy was 97.91% within the 32-meter safe zone. The obstacle recognition algorithm has the same accuracy as the algorithm in [38]. However, the execution time is reduced by 2 and 100 times while using CPU and GPU hardware, respectively. Besides, we also compare our system with [35]. The results are as shown in Table 8. The results show that our system improves both accuracy and execution time compared with [35].



(a) (b) (c) (d)

Figure 22: Result of lane detection with window sizes of (a) 100 pixels, (b) 150 pixels, (c) 200 pixels, and (d) 300 pixels in the TuSimple dataset.

Table 7: Comparison of obstacle detection algorithm.

Algorithm	Average accuracy (%)	Average processing time (second)	Environmental system
HOG + SVM [37]	57.80	4.28	GPU GTX 980Ti
Horizontal filter + Otsu [35]	75.00	0.030	CPU Intel 3.30 GHz
Faster R-CNN [38]	81.60	2.00	NVIDIA TITAN X and GP106 (DrivePX2)
Our proposal (YOLO)	**81.90**	**0.91**	**Intel Core i5- 9300H CPU 2.4 GHz**
Our proposal (YOLO)	**81.90**	**0.022**	**GPU NVIDIA TITAN**

Table 8: Comparison of the proposed system in terms of accuracy and processing time.

Algorithm	Average accuracy (%)	Average processing time (millisecond)
Horizontal filter + Otsu [35]	80.00	43
Our proposal (YOLO)	**89.90**	**24**

5. Conclusion

The paper proposes two-lane detection algorithms based on computer vision and uses a pretrained YOLO model to identify obstacles in images taken from the front camera of the vehicle. In the paper, we identify lanes with nearly 98% accuracy and obstacle detection with average mAP accuracy of 74.1% when the average accuracy with car class is over 80%. However, the system needs to be performed for other datasets. Besides, it is necessary to improve the algorithm on the road with different brightness.

Therefore, we will develop an obstacle recognition algorithm with special conditions such as changing light or many obstacles using other advanced networks in the future.

Data Availability

In the paper, we used a pretrained model with 80 classes of COCO dataset [32]. We also collected the TuSimple dataset (standard dataset for detecting lane) and ours.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was carried out in the framework of the project funded by the Ministry of Education and Training (MOET), Vietnam, under Grant B2020-BKA-06. The authors would like to acknowledge the MOET for their financial support.

References

[1] J. M. Clanton, D. M. Bevly, and A. S. Hodel, "A low-cost solution for an integrated multisensor lane departure warning system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 47–59, 2009.

[2] Q. Li, L. Chen, M. Li, S.-L. Shaw, and A. Nüchter, "A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios,"

IEEE Transactions on Vehicular Technology, vol. 63, no. 2, pp. 540–555, 2014.

[3] J. Stanisz, K. Lis, T. Kryjak, and M. Gorgon, "Hardware-software implementation of car detection system based on lidar sensor data - a demo," in *Proceedings of the Conference on Design and Architectures for Signal and Image*, pp. 1–3, Montréal, Canada, October. 2019.

[4] M. Rezwanul Haque, M. Islam, M. Milon Islam, K. Saeed Alam, and H. Iqbal, "A computer vision based lane detection approach," *International Journal of Image, Graphics and Signal Processing*, vol. 11, no. 3, pp. 27–34, March 2019.

[5] W. Farag and Z. Saleh, "Road lane-lines detection in real-time for advanced driving assistance systems," in *Proceedings of the 2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 1–8, Sakhier, Bahrain, November 2018.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, Las Vegas, NV, USA, June 2016.

[7] K.-Y. Chiu and S.-F. Lin, "Lane detection using color-based segmentation," in *Proceedings of the IEEE Proceedings. Intelligent Vehicles Symposium, 2005*, pp. 706–711, Las Vegas, NV, USA, June 2005.

[8] J. Baili, M. Marzougui, A. Sboui et al., "Lane departure detection using image processing techniques," in *Proceedings of the 2017 2nd International Conference on Anti-Cyber Crimes (ICACC)*, pp. 238–241, Abha, Saudi Arabia, March 2017.

[9] C. Lee and J.-H. Moon, "Robust lane detection and tracking for real-time applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 12, pp. 4043–4048, 2018.

[10] B. Zheng, B. Tian, J. Duan, and D. Gao, "Automatic detection technique of preceding lane and vehicle," in *Proceedings of the 2008 IEEE International Conference on Automation and Logistics*, pp. 1370–1375, Qingdao, China, September 2008.

[11] J. Son, H. Yoo, S. Kim, and K. Sohn, "Real-time illumination invariant lane detection for lane departure warning system," *Expert Systems with Applications*, vol. 42, no. 4, pp. 1816–1824, 2015.

[12] M. Oussalah, A. Zaatri, and H. Van Brussel, "Kalman filter approach for lane extraction and following," *Journal of Intelligent and Robotic Systems*, vol. 34, no. 2, pp. 195–218, 2002.

[13] H. Kong, S. E. Sarma, and F. Tang, "Generalizing laplacian of Gaussian filters for vanishing-point detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 408–418, 2013.

[14] H. Amini and B. Karasfi, "New approach to road detection in challenging outdoor environment for autonomous vehicle," in *Proceedings of the 2016 Artificial Intelligence and Robotics (IRANOPEN)*, pp. 7–11, Qazvin, Iran, April 2016.

[15] Y. Yim and S.-Y. Oh, "Three-feature based automatic lane detection algorithm (tfalda) for autonomous driving," in *Proceedings of the 199 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems (Cat. No.99TH8383)*, pp. 929–932, Tokyo, Japan, October 1999.

[16] V. Gaikwad and S. Lokhande, "Lane departure identification for advanced driver assistance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 910–918, 2015.

[17] C. L. Azevedo, J. L. Cardoso, M. Ben-Akiva, J. P. Costeira, and M. Marques, "Automatic vehicle trajectory extraction by aerial remote sensing," *Procedia - Social and Behavioral Sciences*, vol. 111, pp. 849–858, 2014.

[18] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, Providence, RI, USA, June 2012.

[19] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: the kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[20] J. Fritsch, T. Kühnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 1693–1700, The Hague, Netherlands, October 2013.

[21] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3061–3070, Boston, MA, USA, June 2015.

[22] W. Song, Y. Yang, M. Fu, Y. Li, and M. Wang, "Lane detection and classification for forward collision warning system based on stereo vision," *IEEE Sensors Journal*, vol. 18, no. 12, pp. 5151–5163, 2018.

[23] S. Sivaraman and M. M. Trivedi, "A general active-learning framework for on-road vehicle recognition and tracking," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 267–276, 2010.

[24] H. Tehrani Niknejad, A. Takeuchi, S. Mita, and D. McAllester, "On-road multivehicle tracking using deformable object model and particle filter with improved likelihood estimation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 748–758, 2012.

[25] S.-C. Hsu, C.-L. Huang, and C.-H. Chuang, "Vehicle detection using simplified fast r-cnn," in *Proceedings of the 2018 International Workshop on Advanced Image Technology (IWAIT)*, pp. 1–3, Chiang Mai, Thailand, January 2018.

[26] X. Chen, S. Leng, J. He, and L. Zhou, "Deep learning based intelligent inter-vehicle distance control for 6g-enabled cooperative autonomous driving," *IEEE Internet of Things Journal*, vol. 8, no. 1–1, 2020.

[27] S. Pandi, F. H. P. Fitzek, C. Lehmann et al., "Joint design of communication and control for connected cars in 5g communication systems," in *Proceedings of the 2016 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–7, Washington, DC, USA, December 2016.

[28] S. Baskaran, S. Kaul, S. Jha, and S. Kumar, "5g-connected Remote-Controlled Semi-autonomous Car Trial," in *Proceedings of the 2020 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*, pp. 1–6, Hyderabad, India, December 2020.

[29] S. Pandit, F. H. P. Fitzek, and S. Redana, "Demonstration of 5g connected cars," in *Proceedings of the 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 605-606, Las Vegas, NV, USA, January 2017.

[30] C. Campolo, A. Molinaro, A. Iera, and F. Menichella, "5g network slicing for vehicle-to-everything services," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 38–45, 2017.

[31] C. Perauer, "Development and deployment of a perception stack for the formula student driverless competition," Hochschule Esslingen, University of Applied Sciences, Esslingen am Neckar, Germany, Ph.D. dissertation, February 2021.

[32] T.-Y. Lin, M. Maire, S. Belongie et al., "Microsoft coco: common objects in context," in *Proceedings of the ECCV 2014: Computer Vision – ECCV 2014*, pp. 740–755, Zurich, Switzerland, September 2014.

[33] F. Zheng, S. Luo, K. Song, C.-W. Yan, and M.-C. Wang, "Improved lane line detection algorithm based on hough transform," *Pattern Recognition and Image Analysis*, vol. 28, no. 2, pp. 254–260, 2018.

[34] K. Kim and D. Song, "Real time road lane detection with ransac and hsv color transformation," *Journal of Information and Communication Convergence Engineering*, vol. 15, pp. 187–192, September. 2017.

[35] V. Nguyen, H. Kim, S. Jun, and K. Boo, "A study on real-time detection method of lane and vehicle for lane change assistant system using vision system on highway," *Engineering Science and Technology, an International Journal*, vol. 21, no. 5, pp. 822–833, 2018.

[36] J. Philion, "Fastdraw: addressing the long tail of lane detection by adapting a sequential prediction network," in *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 574–611 583, Long Beach, CA, USA, June 2019.

[37] z. Kaplan and E. Saykol, "Comparison of support vector machines and deep learning for vehicle detection," in *Proceedings of the 3rd International Conference on Recent Trends and Applications in Computer Science and Information Technology*, pp. 1–5, Chennai, Tamil Nadu, May 2018.

[38] C.-T. Lin, P. S. Santoso, S.-P. Chen, H.-J. Lin, and S.-H. Lai, "Fast vehicle detector for autonomous driving," in *Proceedings of the 2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 222–229, Venice, Italy, October 2017.