A Prototype of Emergency Lane Occupation Detection System

Yuhao Ding, Boyan Xu
Prof. Zhibin Chen

New York University Shanghai

Abstract. Illegal occupancy of emergency lanes is a common type of traffic violation which may result in serious consequences by preventing the access of special vehicles, such as ambulances and fire trucks. This project aims to develop a system to automatically monitor and detect the illegal occupancy of emergency lanes by using the in-vehicle cameras of social vehicles. Specifically, A computer vision algorithm is then developed to distinguish those illegal occupancy. Then a application framework is designed to implement the algorithm on a proper hardware system with appropriate computational power and efficiency. This team wants to utilize methods in computer vision technology to implement the algorithm and research into the possible hardware system to perform in the real world, especially considering the upcoming smart-vehicles era.

Keywords: Illegal Occupancy in Emergency Lanes, Computer Vision, On-Vehicle Camera

1 Introduction

1.1 Research Questions and Significance

In the transportation domain, computer vision technology has been aiding law enforcement agencies in various scenarios, for example, detecting drivers that run on a red light, or illegally parked on the street. This team would like to explore this area of study.

One serious illegal act often conducted by drivers is the occupation of Emergency Vehicle Lane. These actions may lead to a serious consequence when special vehicles, such as an ambulance or fire truck, have to use the lane. In China, several severe cases happened because of the traffic jam caused by this occupation of emergency lanes, where the ambulance cannot pass which lead to the death of the patients.

However, it is often hard to regulate this phenomenon. Some drivers cannot resist the intuition of occupying these lanes while the main lane is in a traffic jam. For law enforcement agencies, the simple fact is that they cant deploy cameras over all the roads and highways. Their only way is to catch them in person. Given these facts, we would like to use our approach to reverse this situation.

We noticed that two things have or may have the ability the situation:

1). Many car owners have used the in-vehicle cameras which protect themselves from false claims when an accident happens. The camera will provide the users with a clear image of what happened in front of their cars. We notice that on-vehicle cameras now meet a common standard of 1080p, which have greatly exceeded the minimal requirement for detecting and recognizing a vehicle.

2). 5G technology is beginning to enter into commercial lives. The technology allows instant and large amounts of data transmission.

Based on these two facts, we aim to modify a physical in-vehicle camera to perform algorithms that allow them to transmit data to the cloud. On the cloud, we may deploy a computer vision technology to analyze the videos. If the cloud finds out that there is an emergency car lane in the image and there is a car illegally on that lane, the cloud can automatically submit the image to law enforcement. We can expect that as processors on vehicles progress, the algorithm can be implemented completely locally without using a cloud.

1.2 Project Design

The project design consists of two main parts: Algorithm Design and hardware implementation.

1) Computer Graphic Algorithm

One core element of our project is a computer vision algorithm that detects: 1) vehicles inside the image, 2) lanes inside the image, and 3) possible vehicles that are occupying emergency lanes.

During the design phase of the algorithm, we have certain objectives:
a) The computation power required for the algorithm should be optimized because of the low computation ability of cameras.
b) Considering the aim of the product to to facilitate The algorithm should focus on higher precision rather than recall. As conditions of images vary, the algorithm should make efforts in avoiding making false predictions.

2) Hardware

There could be two general approaches to achieve the purpose. One method could be modifying existing cameras to enable detection of illegal vehicles. However, current on-vehicle cameras mostly do not support large computations. Therefore, in our project we aim to design a framework that can be used on future smart vehicles, and only design a Minimal Viable Product with our own processors and cameras.

2 Algorithm Implementation

We aim to deploy a computer vision algorithm that detects illegal occupies in emergency lanes.

This main algorithm could break into two sub-problems:
1) Detection of car vehicles
2) Car Lane Recognition, including detecting of full lines

2.1 Detection of Car Rears

Our group decides to make sure of Haar Cascade Classifier to detect car rears [1]. The dataset is taken by Brad Philip and Paul Updike in California. contains 469 images of labeled car rears. As the data contains images on highways, it's image quality is similar to the on-vehicle camera we are using. Also, the height of the camera in the dataset, is also similar to the real-world situation where the camera is located. Figure 1 shows one example data.

Fig. 1: Example Data in the Dataset

The dataset also contains 450 negative images that are close to the positive images but contains no car.

Then a cascade classifier is trained using these data. We desire a very good hit rate and low FlaseAlarmRate. The training process simply uses the opencv traincascade command integrated in OpenCV, with hyperparameters: $-stages = 20, -minHitRate = 0.999, -maxFalseAlarmRate = 0.5$.

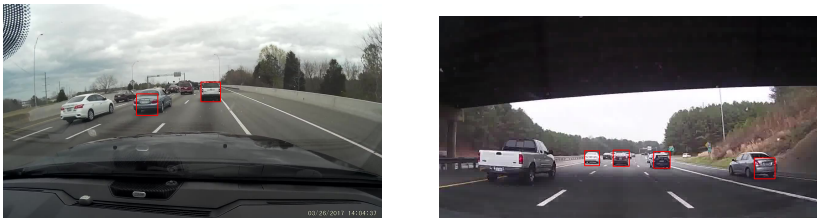
After we retrieve the model, we evaluate the performance of the model using a test set containing similar images of 200 images. The testing result shows that the model works with more than 98% accuracy. The model can successfully detect any cars within close range.

Figure2.a best illustrates the efficiency and accuracy of the model.

In images where multiple cars are presented, thus forms a "congested traffic" scenario, the model also works fine. Figure2.b provides an example outcome.

As shown in the image, closer and straighter targets can be detected, while the others will be missed.

We comment that there is no need to detect cars at a far range, so such accuracy is desirable enough. Combined with the results from the next section,



(a) Example I (b) Example II

Fig. 2: Example Outcome of Car Detection

we will find that the best accuracy comes when the emergency line is right next to the car, and when the lanes are far from the car, the accuracy will be really low. Therefore, only vehicles with in a close range need to be detected.

There might be FalseAlarms in some images, but most FalseAlarms happens to detect the same car with two overlapping results, simply adding a "-minInteval" command can solve the issue.

Finally, we look into the running time and the computational power needed for this process. As cascade classifiers are designed to use in situations with lower computational power, the process can be finished really fast with a computer. On a normal computer, the process takes 0.06s, which is very reasonable and executable.

2.2 Car Lane Detection

For this algorithm, we decides to use Computer Vision methods to detect car lanes. Though there are complex machine learning methods with extremely high accuracy, after testing we find out that the machine learning methods takes too much computational power. We seek an easy, but quite accurate way to meet our project's purpose.

We first notice that, the emergency lanes on highways are mostly white full lanes. Therefore, our first step is to fill out unnecessary colors from the image.

(a) Original Image (b) White Mask Image

Fig. 3: White Color Filtering

Our first step is to create a white color mask that only filter out the white color inside the image, this is a very quick and low-cost operation but could greatly improve algorithm accuracy. Figure 3 shows the result of such filtering. One important task is to decide the bounds for white color mask. As images conditions varies and depends on the brightness of the image, the white color of the lane varies. Also we don't want the color range to be too broad because that will increase the amount of outliers inside the image.

After testing, the eventual RGB colors we are using to bound the white colors is shown in Table 1.

Table 1: Lower and Upper Bound of the White Color Mask

	Red	Green	Blue
lower bound	0	0	160
upper bound	180	25	255

One future improvement of this might be, making use of the brightness detector on the vehicle, the algorithm could auto-detect brightness and change the bounds of the colors for better accuracy.

Following the white mask, the algorithm performs a canny edge detector. The algorithm will detect the edges inside the image. They are two parameters that need to be specified in Canny edge detection, we choose default size of Gaussian Filter to be (5,5), and deviation is 0. Figure 4 and 5 shows the example outcome of such operation.

(a) Original Image (b) After Detection

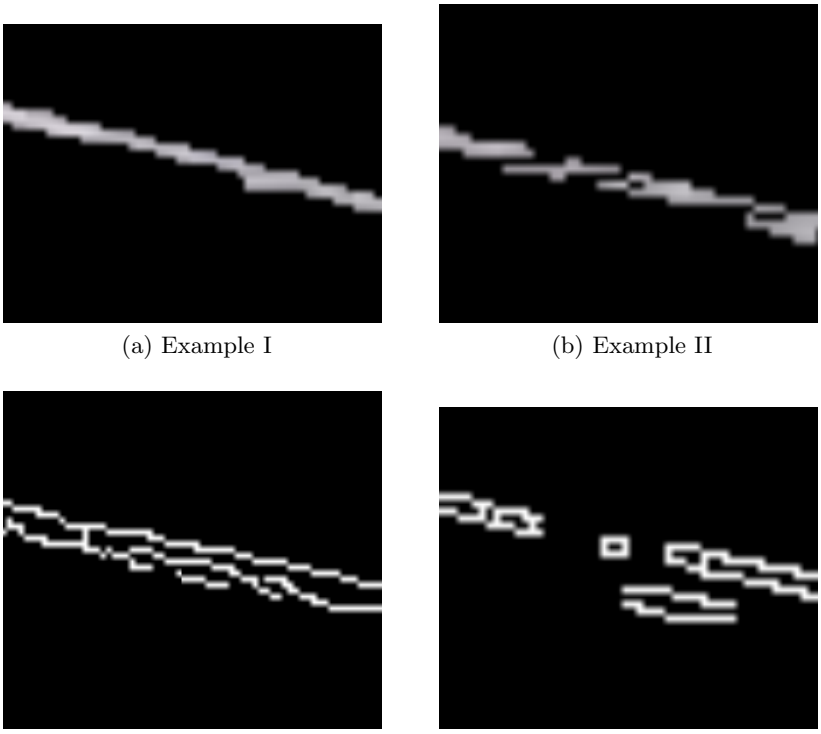
Fig. 4: Canny Edge Detection Example I

Canny edge detector mainly have five steps: Gaussian filter, Finding the intensity gradients, Apply non-maximum suppression, Apply double threshold to determine potential edges, Track edge by hysteresis. These steps minimize the influence of outliers. In the final step, their is a blob analysis that works with the criteria: As long as one strong edge pixel is neighboring, this weak edge can be preserved.

Figure 6 contains four typical edges we may get from a lane detection.

(a) Original Image (b) After Detection

Fig. 5: Canny Edge Detection Example II



(a) Example I (b) Example II

(c) Example III (d) Example IV

Fig. 6: Example Lane Edges

Example a,b comes from images that are thin, this is usually the case when the car is far from the lane, thus provides a flatted image of the lane. Example c comes from a clearer image as opposed to example d that provides scattered image of the car lane. In our case, Example a,b,c are all desirable, and Example d may pose risks for the algorithm.

As the result is satisfactory among most images, the next step is to apply a line detector that detect lines in the image, here we use the Hough Line Detector to achieve our goal.

After the detection, we are likely to have a set of detected lines. Given that we are trying to detect the emergency lane, we use the following algorithm to find the final result lane:

1) Set $\epsilon = 2$, group any two lines $a_1x + b_1y = c_2$, and $a_2x + b_2y = c_2$, and satisfies $|a_1 - a_2| \leq \epsilon$, $|b_1 - b_2| \leq \epsilon$ and $|c_1 - c_2| \leq \epsilon$, as one line. There is to remove multi-detection of the same line. 2.) We find the line that satisfies $-\frac{a_1}{b_1} < 0$ and $min - \frac{a_1}{b_1}$, as the emergency line should be the leftmost line in the image.

Here is a final result after the detection algorithm, shown in figure 7 and 8.

(a) Original Image (b) After Detection

Fig. 7: Car Lane Detection Example I

(a) Original Image (b) After Detection

Fig. 8: Car Lane Detection Example II

2.3 Final Result

Eventually we combine the above algorithms. After line detection, we simply mask the area below the line, and detect the car inside the area. Figure 9 shows the final result.

The car is detected as parking in the emergency lane, in this frame, the photo of the car (inside the red rectangle), will be transmitted to authority to report possible illegal actions.

Fig. 9: Final Result

3 Hardware Implementation

Our team's next step is to produce a hardware system that can support the above algorithm. This part is still in progress, here is a detailed hardware plan we are trying to use in this implementation.

The algorithm can be deployed to several potential terminals with different fairly different specs, ranging from autonomous vehicle to mobile devices. We adopt a PC and an iPhone to test the performance of the algorithm and web communication on those different devices.

3.1 Test PC Spec

The vehicle processors built for autonomous driving are equipped with both high performance GPU and mulit-core CPU units. For example, NVIDIA's latest processor designed for autonomous driving: NVIDIA Xavier has the following spec:

- GPU: 512-Core Volta GPU with 64 Tensor cores, which provides 11 TFLOPS[FP16]
- CPU: 8-core Carmel ARM v8.2 64-Bit CPU
- Memory: 16GB 256-bit LPDDR4x

Therefore, to test the feasibility of our algorithm for those hardware designed for future autonomous vehicles, we adopt a modern PC with weaker specs as below: - GPU: GeForce GTX 1080, which provides 138 TFLOPS[FP16]
- CPU: 4-core Intel i7-7700 CPU
- Memory: 16GB DDR4-2400

3.2 Test Mobile Device Spec

To test the feasibility of our algorithm for those hardware designed for future autonomous vehicles, we adopt a hardware with the following specs:
- GPU: Adreno 512
- CPU: Octa-core (4x2.2 GHz Kryo 260 Gold 4x1.8 GHz Kryo 260 Silver)
- Memory: 4GB

4 Network Communication

4.1 Data Format

The video format we chose is .flv, and we use MediaCodec provided by Android system and ffmpeg to encode video in that format.

4.2 Transfer Protocol

We adopt RTMP protocol to transfer the compressed video data to a NGINX server.

Figure 10 shows the expected layout of the application.

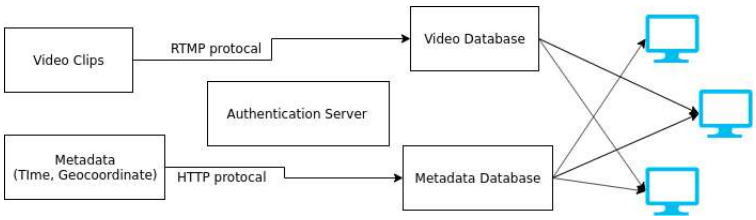


Fig. 10: Layout of the Application

5 Future Plans with the Project

The future plan of this project mainly deal with the hardware implementation, and this teal will also find ways to optimize algorithm.

For the hardware implementation, the aim will be building a minimal viable product that could be used on the highway to real-time detect illegal cars. This involves putting the existing algorithm into smaller chips and combine them with a camera. Our previous specification notes the processor we are using, they might be subject to change based on the actual implementation. The main purpose, is to lower the cost while retain enough computational power for the project.

For the algorithm, the focus will be on optimizations its accuracy. While the basic outline if clear, we have several known deficiencies in our algorithm:
1. They are several kinds of cars that can't be detected, this is due to a relatively small training set we are using. This could be improved while we use a much bigger training set.
2. The lane detection algorithm does not have a overwhelmingly good accuracy, this results from under-optimized hyper-parameters we are using. We will conduct a more thorough testing process to improve the process.
3. The algorithm hasn't been optimized for videos. While there are multiple frames for a video, it will take significantly high amount of time to detect every frame. Also, if there is an illegal car on the highway, it will be detected and reported several times in or algorithm, we will further extend the algorithm to solve these issues.

References

[1] Brad Philip and Paul Updike. "Car dataset". In: *California Institute of Technology SURF project* (2001).