Real-time Vehicle Detection using Haar-SURF Mixed Features and Gentle AdaBoost Classifier

Sun Shujuan[1], Xu Zhize[2], Wang Xingang[1], Huang Guan[1], Wu Wenqi[1], Xu De[1],

1. Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
E-mail: sun-hit@hotmail.com

2. Institute of Computer Graphics & Computer-Aided Design, Tsinghua University, Beijing 100080, China
E-mail: xu-zz12@mails.tsinghua.edu.cn

Abstract: On-road vehicle detection is one of the key techniques in intelligent driver systems and has been an active research area in the past years. Considering the high demand for real-time and robust vehicle detection method, a novel vehicle detection method has been proposed. This paper presents a real-time vehicle detection algorithm which uses cascade classifier and Gentle AdaBoost classifier with Haar-SURF mixed features. We built up a large database including vehicles and non-vehicles for training and testing. A pipeline is then presented to solve the detection problem. Firstly, lane detection is employed to reduce the search space to a ROI. Secondly, the cascade classifier is applied to generate some candidates. Finally, the single decision classifier evaluates the candidates and provides the target vehicle. The experiments and on-road tests prove it to be a real-time and robust algorithm. In addition, we demonstrate the effectiveness and practicability of the algorithm by porting it to an Android mobile.

Key Words: Vehicle Detection, Haar-like features, SURF descriptor, Gentle AdaBoost

1 Introduction

With the rapid growth of vehicle number in urban traffic, auto accidents have also become more and more often. So the Intelligent driver assistance or other driver helper systems have received considerable attention over the years. On-road vehicle detection, one of the key techniques in intelligent vehicle system, is now an active research area. In almost any intelligent vehicle system, vehicle detection often acts as the first step, so the demand for robustness and timeliness is high. There are two types of vehicle detection, one is vehicle detection from a fixed camera which is used in traffic monitoring systems, the other is on-road vehicle detection, where the camera is mounted on the vehicle and is moveable. This paper will focus on on-road vehicle detection. Due to the importance and challenge of the on-road vehicle detection problem, many algorithms and methods are proposed. Sun [1] has written an overview of on-road vehicle detection methods. In the literature, on-road vehicle detection can be decomposed into two steps as follows:

(i) Hypotheses generation: the goal is to generate a reduced search area and provide potential positions of vehicles.

(ii) Hypotheses validation: the goal is to verify the hypotheses generated from the last step using some complex algorithms.

Hypotheses generation is based on simple, low level algorithms which estimate the vehicle locations. They can be divided into two categories: one is knowledge-based, these methods use some prior knowledge, such as vehicle shape, vehicle color and highway characteristics, the other is motion-based, these methods mainly use optical flow or some multiple view geometry algorithms.

Hypotheses validation can be classified into two classes, template-based and appearance-based. Template-based methods use predefined patterns of vehicles as templates, and calculate the correlation between the templates and the input images. Appearance-based methods use machine learning techniques, and they learn the characteristics of the vehicles from the training set. Each training sample is represented as a local or global feature vector. Then the feature vectors are fed to a classifier, such as Neural Network (NN) or support vector machine (SVM). The classifier can give a decision boundary to classify the vehicle class and the non-vehicle class. For example, Goerick[2] used histogram of Local Orientation Coding (LOC) as features and trained a Neural Network.

In addition to these methods, Viola and Jones [3] proposed cascade of boosted classifiers, which inspire a lot of recent work of on-road vehicle detection. In this type of work, they often use Haar-like features and design associated week classifiers. In [4], Haar and histogram of oriented gradients feature is used and cascade detector is employed. In [5], they used SIFT rather than Haar-like features in the detector, while linear SVM are used as week classifiers. This paper is also an appearance-based method but use different features and different week classifiers from these methods.

This paper is organized as follows: In Section 2, we present a brief introduction of feature space, mainly about Haar features and SURF (Speed-Up Robust Feature) features. An introduction of AdaBoost especially Gentle Boost is presented in Section 3. In Section 4, we will describe our algorithms in detail. We will go through the pipeline and analyze the results. The paper concludes in Section 5 with a discussion and future work.

2 Feature Selection

In this paper, we use the Haar-like features and SURF descriptor features as the candidate features. Here is a brief introduction to them.

978-1-4799-7016-2/15/$31.00

2.1 Haar-like Features

Haar-like features provide information about the grey-level distribution of two adjacent regions in an image. The detection method based on Haar-like feature is better than other methods that directly deal with pixels. Firstly, it is easy to encode a program with relatively limited training data. Secondly, the calculation speed of feature detection is much quicker than pixel-based ones.

Figure 1 show the set of Haar-like features. These filters consist of two or three or four rectangles. It can be at any position and scale within the image. To compute the output of a filter on a certain region of image, the sum of all pixels values in the grey region is subtracted from the sum of all pixels values in the white one and normalized by a coefficient.

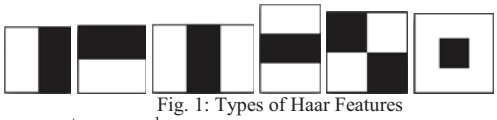

Fig. 1: Types of Haar Features

The 1st and 2nd Haar-like features indicate horizontal or vertical intensity difference between two areas (grey and white region). The 3rd and 4th rectangle features describe the intensity difference between the middle region and aside areas. And the 5th rectangle features show the diagonal differences. The last Haar-like feature is the difference between the center and surrounding areas.

For image with size of 36*36, the number of features in different size and scale is huge. Viola and Jones [3] introduced the integral image which is an intermediate representation of an input image to improve the speed of the calculation. Sum of the rectangle can be calculated by using only three additions and four references in the integral image. The integral image is shown as follow.

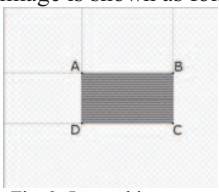

Fig. 2: Integral image

In an integral image, the value of each pixel S (i, j) is the sum of all pixels from the original image within the rectangle from the top-left corner to position (i, j), which value is computed as follow:

$$S(i,j)=\sum_{x=0}^{i}\sum_{y=0}^{j}I(x,y) \tag{1}$$

I(x, y) is the pixel intensity in the original image of the position (x, y). Thus, the intensity of rectangle A-B-C-D is:

$$S(C)+S(A)-S(B)-S(D) \tag{2}$$

The difference of two adjacent rectangle can be computed by using only six references in the integral image. For a filter with three rectangular regions, only eight references are needed. At the same time, integral image allows to perform fast variance normalization, necessary to reduce the effect of different lighting conditions.

2.2 Speeded-up Robust Features(SURF)

Speeded-up Robust Features (SURF) is a novel scale and rotation invariant interest point detector and descriptor. It was originally presented by Herbert Bay and built upon SIFT. SURF descriptor is used in our method as the candidate features. It has in-plane rotation, scale invariant and certain level of tolerance for view point and illumination changes. From the scale space using Hessian-matrix approximation with the approximate second order Gaussian derivatives, the interest points are detected.

For each interest point, the Haar wavelet response in both x and y direction within a circular neighbor-hood of the interest points are calculated. From a sliding window, the orientation with the largest wavelet response is picked as the point's dominant orientation. At last the SURF descriptor is the Haar wavelet responses within a rectangular region. The rectangle region is divided into 4*4 sub-patches. For every sub-patches, a 4 dimensional descriptor is built as part of the final feature vector as shown in Figure 3.

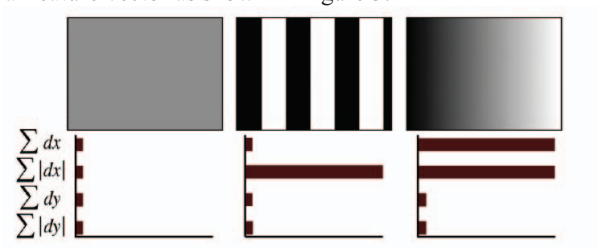

Fig. 3: SURF 4 dimensional descriptor

In our work we chose SURF descriptor as our candidate features other than SIFT or other features. Since SURF uses integral image, this makes the computational cost of SURF relatively less sensitive to the image size than SIFT. SURF is based on gray images, which does not require region segmentation, thus largely reducing the computational cost. It depends on gradient information on sub-patches, instead of individual gradients, which makes SURF less sensitive to noise. And it not only uses the sum of wavelet responses, but also the sum of absolute values of them in the patches for information of polarity of the intensity changes.

3 Gentle AdaBoost Classifier

AdaBoost algorithm is the short for "Adaptive Boosting". It is a machine learning algorithm formulated by Yoav Freund and Robert Schapire. Boosting is a way to combine the performance of many weak classifiers and then produce a final powerful classifier. It was proposed in the computational learning theory literature and has received much attention in recent years.

While boosting has evolved somewhat over the years, many variants are proposed, among them LogitBoost, Real AdaBoost, Gentle AdaBoost and Discrete AdaBoost. Gentle AdaBoost is the most commonly used version because of its better generalization ability. This paper is also focused on Gentle AdaBoost.

The concise description of Gentle AdaBoost in the two-class classification setting is as follows. The algorithm takes a training data set $(x_1, y_1), \ldots,(x_n, y_n)$ as input, with x_i being a vector valued features extract from the i^{th} sample whereas y_i is the label of the i^{th} sample, y_i=-1 means a negative sample, y_i=1 means a positive sample.

We define the final powerful classifier as

$$F(x)=\sum_{i=1}^{M}f_m(x) \tag{3}$$

And $f_m(x)$ is the weak classifier selected from each turn. Stumps are used for the weak learners. Stumps are

single-split trees with only two terminal nodes. The weak classifier $f_m(x)$ minimize the overall test error

$$\sum_i w_{t,i}(y_i - f_t(x_i))^2 \quad (4)$$

as much as possible, where w represents the training data with weights $w = (w_1, w_2, \ldots, w_n)$ and t means the t^{th} round. After each round, the algorithm will increase the weights of the observations misclassified by $f_m(x)$ by a factor that depends on the weighted training error. The AdaBoost procedure trains the classifiers $f_m(x)$ on the same weighted training samples for the first iteration, and then give higher weight to samples that are currently misclassified and give lower weight to cases that are currently correctly classified. This is done for a sequence of weighted samples, and then the final classifier is defined to be a linear combination of the classifiers from each stage. A detailed description of Gentle AdaBoost is given in the following code box.

Algorithm 1 Gentle AdaBoost

GENTLE-ADABOOST()

1 Start with weights $w_i = 1/N$, i = 1,2,...,N,F(x) = 0.
2 **for** $m \leftarrow 1$ **to** M
3 **do** (a) Fit the regression function $f_m(x)$ by weighted
4 least-squares of y_i to x_i with weight w_i
5 (b) Update $F(x) \leftarrow F(x) + f_m(x)$
6 (c) Update $w_i \leftarrow w_i \exp(-y_i f_m(x_i))$ and renormalize.
7 Output the classifier $\text{sign}[F(x)] = \text{sign}[\sum_{m=1}^{M} f_m(x)]$

Gentle AdaBoost used resample scheme to implement step 3 of the Algorithm 1 by weighted sampling from the training data. For some reason, it seems that AdaBoost is resistant to over-fitting. In order to make boosting performs comparably well we need to observe the performance on the test data set continually, which will help us choose a more appropriate number of iterations to avoid over-fitting.

4 Implementation

The pipeline of the vehicle detection system is organized in Figure 4. Firstly, a lane detector described in Section 4.1 is employed to the input image, which reduces the search space from the whole image space to a ROI. Secondly, a cascade classifier in Section 4.3 will act to further reduce the search space to only a couple of candidates. Then, a single decision classifier will evaluate the candidates. Each candidate will be given a rank value. Higher rank value means higher possibility to be a vehicle. Finally we combine the rank value of the candidates and the lane information to get the target vehicle. A candidate with higher rank and closer distance to the lane center will be more likely to be chosen as the target.

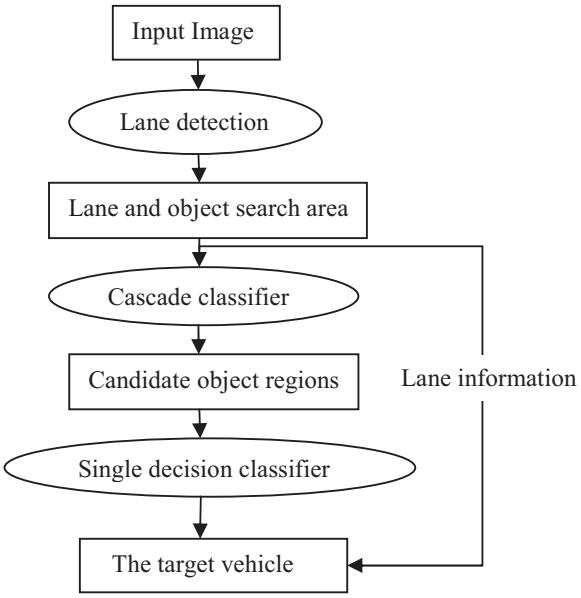

Fig. 4: The pipeline of the vehicle detection system

4.1 Lane Detection

Lane detection is performed before the vehicle detection. The final result of lane detection is the intersection of the two detected lane, which is useful for the latter vehicle detection. The intersection of the detected left and right boundary of lane will enormously reduce the search space of the vehicle detector, for all vehicles can only appear within the lanes.

To address the lane detection problem, we proposed a method based on Hough transform. It is an improved method of the classical Hough lane detection. In the previous method [11], Canny-edge extraction was applied firstly, after that Hough line detection was performed on the edge map. The previous method is somehow not robust and stable.

Our method first calculates the gradient of the x direction of the image, using Sobel gradient. In the generated gradient map, the edge of the lane is enhanced so we do edge extraction of Canny on the gradient map to get the edge, rather than on the original image. Furthermore, we take advantages of the temporal information, since in the temporally adjacent frames lane will not change much. So the ROI (Region of interest) is selected based on the current frame and the last frame. It will produce more stable result and avoid jittering. At the same time, it will reduce the interference of other lines which are not lane, such as building on the road side, and the painted arrow sign on the road.

After we get the ROI, Hough transform is performed in the ROI. The lines which got the most votes are selected as candidates. We define a metric to select the final decision out of the candidates. The formal formula of the metric is defined as follows:

$$Sim(l_i, l_0) = \frac{1}{1 + dis(l_i, l_0)} \quad (5)$$

The $Sim(l_i, l_o)$ denotes the similarity between l_i and l_o, l_i is the candidate lane of current frame, while l_o is the result lane of last frame. The distance metric is defined as:

$$dis(l_i,l_0)=\frac{\alpha|K_{li}-K_{l0}|}{\max(|K_{li}-K_{l0}|)}+\frac{(1-\alpha)\sqrt{(C_{li}x-C_{l0}x)^2+(C_{li}y-C_{l0}y)^2}}{\max(\sqrt{(C_{li}x-C_{l0}x)^2+(C_{li}y-C_{l0}y)^2})}$$
$$(j=1...n) \quad (6)$$

The metric considers both the similarity of the slope of the two lines and the distance between the centers of the two lines. The lines that have higher similarity to the lanes from the last frame is preferred, on the other hand, the lines that have shorter distance to the center of lanes from the last frame is also more likely chosen. The denominator of the formula is the normalization term.

Figure 5 is the test result of lane detection, the green are the candidate lanes and the red are the selected final lanes.

Fig. 5: Result of Lane Detection

4.2 Database

In this section, we describe the image database used for the training and testing. To get more actual vehicle and background data, we collect 312 real video on the high way. Each video lasts at least one hour. From these videos, we get a huge amount of positive and negative samples. The database used for experimentation contains more than 6,530 images of one or more rear view of vehicles, resulting in more than 10,362 vehicle images of many kinds of cars. The dataset was labeled manually by enclosing the vehicle in a bounding box.

We have constructed the databases as follows:

(i) Positive database (31,086 samples): the positive sample data set contains 10,362 vehicle images. Then we augment the database. The number of vehicle images is tripled by adding random disturbance and applying random affine transformation. Thus, we get a quite large positive database. The number of positive samples in the database comes to 31,086.

(ii) Negative database (225,416 samples): the negative sample data set composed of negative examples which are taken randomly in a set of 112,708 arbitrary images with random size and random position in the image (which do not contain any vehicle).

(iii) Training database and Test database: From all the positive and negative samples, we perform a 5-fold cross-validation, randomly partition the data into 5 pieces. Four-fifths are used as training set and the left fifth are set to test the classifier from the training result, which repeat 5 times. This testing set is used to check whether the strong classifier reach the minimum acceptable correct detection rate and the maximum acceptable false alarm rate. Whereas, we can also select the best number of rounds for training through the testing result to avoid over-fitting. The training and testing set is randomly selected from the entire database for every time.

4.3 Cascade Classifier (Haar)

The cascade classifier works as shown in Figure 6. It is composed of a set of serially connected classifiers. The classifiers in the top are of relatively low hit rate, but require less calculation and can get the result quickly. Thus lots of non-targets will be rejected by them. The left overs of the last classifier (F_n) are the vehicle candidates.

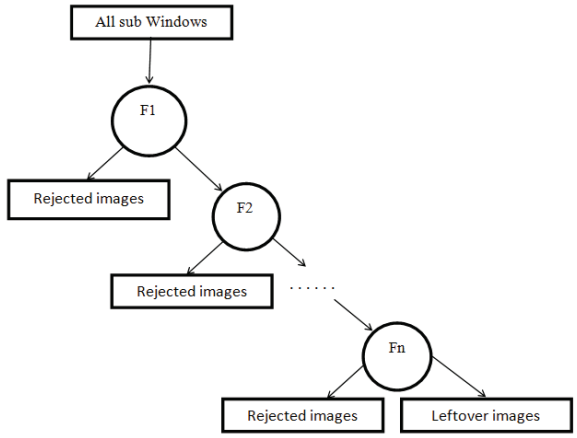

Fig. 6: Cascade Classifier

We use the cascade classifier to get the candidate object region in each frame more quickly and accurately. There are some other methods to get the candidate object region such as region growth and finding the shadow under the car. For region growth, it needs a very suitable threshold for pixel growing if we want to obtain a good growing result. But the threshold is sensitive to the lighting condition and the shadow from the buildings and trees. At the same time, region growth is more time-consuming which cannot be accepted in the mobile phone applications. The other method is to find the shadow under the car to select all candidate objects in the image. It also encounters the threshold problem the same as region growth.

Cascade classifier is adding more and more strong classifier continuously. It can reject the background region quickly, saving time to compute the object region. When a sub-window image is fed in the cascade classifier, it will be sent to the next classifier only when it comes through the current classifier. Otherwise, the image will be recognized as a background image and be rejected by the classifier. A sub-window image will be recognized as object when it comes through all the classifier in the cascade structure. The front classifier use simple features and structure, while the back use more complex features and structure. The most of the sub-windows are background regions, so they will be rejected by the cascade classifier earlier. Only the sub-windows which are likely to contain real object need to pass all the cascade structures. So the calculation time can be greatly reduced.

With the OpenCV Haar-Training tools and all the positive and negative samples, we get the classifier of vehicles with resolution of 32*32. In our cascade classifier, we set the stages to be 16. In each training stage, the positive sample number is 1000 and the negative sample number is 2000.The feature type we use is Haar-like feature and the classifier type we choose is the Discrete Adaboost. The min-Hit-Rate is set to 0.997 and the max-False-Alarm-Rate is set to 0.5. The parameter of weight-Trim-Rate is 0.95. For each weak classifier in the Adaboost, the max-depth is set to 1 which

means the weak classifier is a binary tree with depth of 1. And the Max-Weak-Count is set to 100 means the weak classifier is composed with 100 binary trees. And the cascade classifier is composed with 16 weak classifiers. The hit-Rate of the cascade classifier is 0.997^{16}=0.95306, and the false-alarm rate is 0.000015.

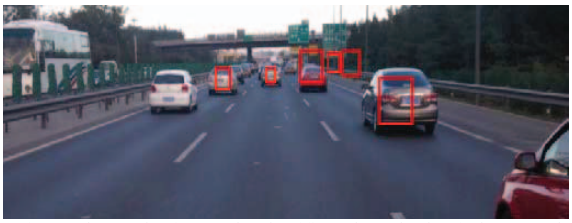

Fig. 7: The result of Cascade Classifier. The red sub-windows are the candidates.

4.4 Single Decision Adaboost Classifier(Haar & SURF)

In this paper, we train a Gentle AdaBoost classifier with SURF descriptor feature and Haar-like feature as the single decision classifier. Firstly, sub windows of different scales and different positions are generated. Here we generate a total number of 1016 sub windows for a 36 * 36 sample, among them the minimum window size is 4*4. After that, we extracted the SURF features and Haar-like features for each sub window. The Haar-like feature is described in 2.1. The SURF features can be expressed as follow:

$$\sum_{i=1}^{n} d_{pi}x, \sum_{i=1}^{n} d_{pi}y, \sum_{i=1}^{n} |d_{pi}x|, \sum_{i=1}^{n} |d_{pi}y| \qquad (7)$$

After the SURF feature extraction, each sample can be represented as a 10160-dimensional feature vector. In each round of the training, 1000 positive and 2000 negative samples are randomly selected respectively according to the weights of all the samples. Each round we can get a week classifier, which is a binary decision tree with single node. We denote it as stump (d, t, a, b), where d is the number of feature dimension (count from 1-10160), t is the threshold of the d^{th} dimension of features, a/b is the output of the week classifier of each sample respectively, as follows

$$h(x) = \begin{cases} a, x > t \\ b, x < t \end{cases} \qquad (8)$$

where x is the feature's value in d^{th} dimension for each sample.

In each round of training, the weights of the samples will be adjusted in this way: the weight of misclassified sample will be increased, while the weight of correctly classified sample will be decreased. After each round of training, the weight of the week classifier generated is also calculated, according to the error rate. After hundreds of rounds, we get hundreds of week classifiers. The final classifier is the combine of these week classifiers, in a weighted way. We can present it as:

$$H(x) = sign[\sum_{i=1}^{n} \alpha_i h_i(x) - thr] \qquad (9)$$

In our experiment, the training set includes 31,086 positive samples and 225,416 negative samples. The 300-round training result ROC curve is as Figure8. The solid curve is the result of the AdaBoost and the dashed curve is the result of Logistic Regression. The blue curve is result of the training set and the red curve is the result of the testing set. We can find out that Adaboost is better than Logistic Regression and the training result is better than test result which is normal. The 300-rounds training result is not the best. Figure 9 is the graph of 400-round's result, note that the coordinate is in semi-log. We can find both the training and test result is better than the 300-rounds' result. With the increment of the training round, Figure 10 is the result of 500-rounds, it shows a perfect well training result, but the performance on test result is not improved that much, which is similar with 400-round's test result. That means if the rounds is more than 500, the classifier is tend to be over-fitting. Finally, we set the training round number to 500 to get the final classifier which the detection accuracy is 99.65% at the position where TPR+FPR=1 and the mean-square-error = 0.001585.

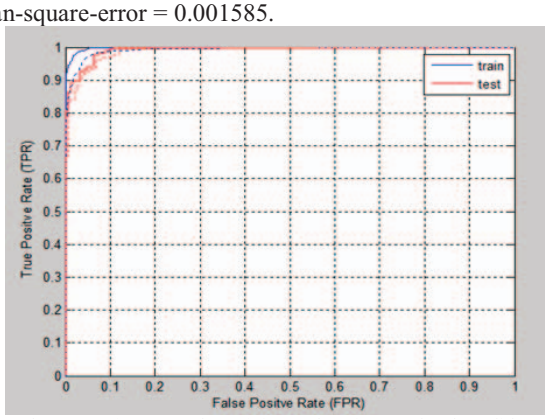

Fig. 8: The ROC of 300 rounds' training and testing result. The solid curve is the result of the AdaBoost classifier and the dashed curve is the result of Logistic Regression. The blue curve is result of the training set and the red curve is the result of the testing set. From the ROC we can find AdaBoost is better than Logistic Regression and the training result is better than test result. The detection accuracy is 97.8% at the position where TPR+FPR=1.

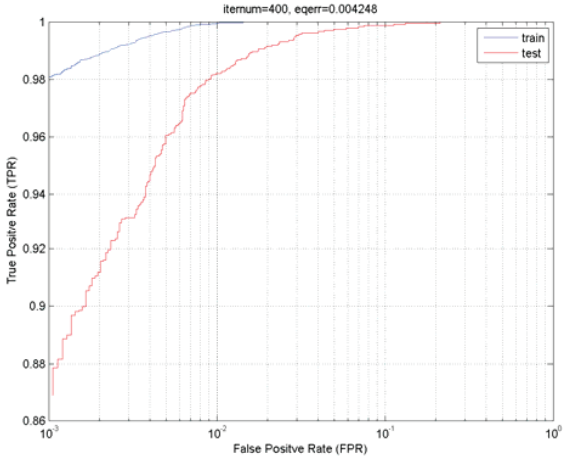

Fig. 9: The semi-log ROC graph of 400-rounds's result. The detection accuracy is 98.7% at the position where TPR+FPR=1 and the mean-square-error = 0.004248.

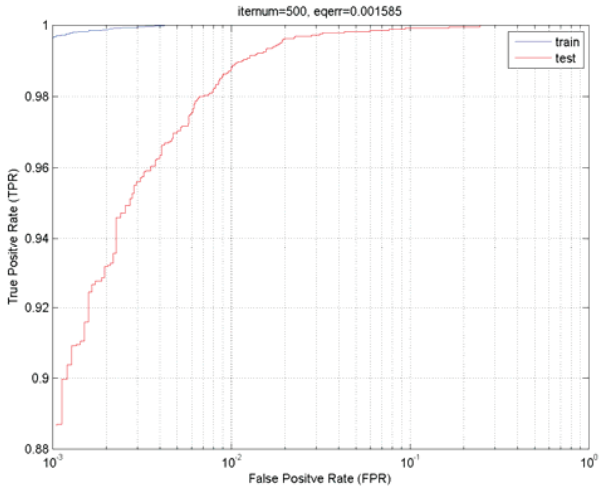

Fig. 10: The semi-log ROC graph of 500-rounds' result.
The detection rate is 99.65% at the position where TPR+FPR=1 and the mean-square-error = 0.001585.

4.5 Video and On-Road Experiment Result

In this paper, a large number of tests are carried out on lane detection and vehicle detection algorithm based on the real videos shot in the high-ways. Video clips that are shot under different circumstances are tested, including ideal light, strong light, multi-lane and other complex scenes. In all cases, vehicle detection algorithm is tested, and the accuracy is relatively high. With the cascade classifier (Haar) and the single decision classifier (Haar and SURF), the average detection accuracy is 91.2% and the average false alarm rate is 13.6% in our 20 hour's video test.

In order to meet the condition of actual application environment, the algorithm in this paper is implemented in C++ language. The test results prove the algorithm is real-time. The average processing time on the PC platform is about 21.72 ms (about 50fps) for each frame in the test video. We also have transplanted the algorithm to Android 4.2 version. On the mobile phone, the average processing time is about 192-286 ms (about 5fps) for each frame and the average object detection rate is 83.7%.

We have already released the Android application in the web which is called Smart-Go. Some test results are shown below, Figure 11 is the result of lane detection. Figure 12 is the result of vehicle detection in PC platform. Figure 13 is the result of vehicle detection in different light conditions. Figure 14 is the result of vehicle detection in an Android mobile. These all show that our algorithm is robust and reliable.

Fig. 11: Selected lane detection result. These cases are more challengeable than others in the test. In the left image, there is an arrow sign in the ground which is very similar to lane. In the middle image, the lane is not very clear. In the right image, the light condition is not good. Our method can deal with these situations.

Fig. 12: Vehicle detection results. Zoom in for more details. Note that (d)(e)(f) are non-trivial cases. In (d), only the white car is detected because the red car locates outside the lanes. In (e), the target car is far away from camera. In (f), there is occlusion in front of the target vehicle. Our methods can produce fairly good result.

Fig. 13: Vehicle detection in different light condition. The left image is in relatively high light. The middle is in a shadow. The right image is in a relatively poor light condition.

Fig. 14: Screenshot of vehicle detection in Android platform.

5 Conclusions

In this paper, we present a cascade classifier and a single decision vehicle detection classifier, along with a lane detector. The cascade classifier and lane detector reduce the search space significantly while the single decision classifier based on Gentle AdaBoost with Haar-SURF features point out the final position of the vehicle with the lane information. Two features space are studied and evaluated: Haar-like features and SURF features. The detector based on Haar and SURF mixed features achieves a high accuracy and can perform in real-time.

However, there are also some drawbacks in this system. One is the lane detector will sometimes misguided the vehicle detector, for instance, when the lane detector gives totally wrong lanes, the vehicle detector may turn out to search in an wrong area. Another problem is when the light condition is too dark or too bright, the vehicle detector may not work properly.

So our future work will be devoted to the improvement of the lane detector and make the vehicle detector more insensitive to the light condition. Furthermore, we may combine the vehicle detection and vehicle tracking to outperform the current method.

References

[1] Sun Z, Bebis G, Miller R, On-road vehicle detection: A review, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.28, No.5, 694-711, 2006.

[2] Goerick C, Brauckmann M, Local orientation coding and neural network classifiers with an application to real time car detection and tracking, Mustererkennung, 1994.

[3] Viola P, Jones M J, Robust real-time face detection, International journal of computer vision, Vol.57, No.2, 137-154, 2004.

[4] Negri P, Clady X, Hanif S M, et al, A cascade of boosted generative and discriminative classifiers for vehicle detection, EURASIP Journal on Advances in Signal Processing, 2008.

[5] Zhu Q, Yeh M C, Cheng K T, et al, Robust real-time face detection, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol.2, 1491-1498, 2006.

[6] Stanciulescu B, Breheret A, Moutarde F. Introducing new adaboost features for real-time vehicle detection, arXiv preprint arXiv:0910.1293, 2009.

[7] Bay H, Tuytelaars T, Van Gool L, Surf: Speeded up robust features, Computer Vision–ECCV 2006, 404-417, 2006.

[8] Yao Y, Li C T, Hand posture recognition using surf with adaptive boosting, British Machine Vision Conference, 2012.

[9] Friedman J, Hastie T, Tibshirani R, Additive logistic regression: a statistical view of boosting, The annals of statistics, Vol.28, No.2, 337-407, 2000.

[10] Bradski G, Kaehler A, Learning OpenCV: Computer vision with the OpenCV library, " O'Reilly Media, Inc.", 2008.

[11] Yu B, Jain A K, Lane boundary detection using a multiresolution hough transform, Proceedings of the International Conference on Image Processing, Vol.2, 748-751, 1997.

[12] Lowe D G, Distinctive image features from scale-invariant keypoints, International journal of computer vision, Vol.60, No.2, 91-110, 2004.

[13] Canny J, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, No.6, 679-698, 1986.

[14] Kalal Z, Mikolajczyk K, Matas J, Tracking-learning-detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.34, No.7, 1409-1422, 2012.

[15] Fawcett T, ROC graphs: Notes and practical considerations for researchers, Machine learning, Vol.31, 1-38, 2004.