The Implementation of Lane detective Based on OpenCV

Wu Ye, Shan Yuetian, Xu Yunhe, Wang Shu,Zhuang Yuchen
School of Automobile Engineering
Harbin Institute of Technology at Weihai
Weihai, China

***Abstract*- In intelligent vehicle systems, lane detection is one of the most important parts. This paper presents a lane detection algorithm that based on Hough transform. Principle of the algorithm and the implementation base on OpenCV are discussed in detail. The algorithm was verified at the end of this paper.**

Key words: lane detection, Hough transform, OpenCV

I. INTRODUCTION

With the development of construction of roads and the more and increasing vehicles, the traffic accidents also tend to be on the increase when running on the road. Obviously, many accidents could be avoided if there is a device to remind drivers to take preventive measures in advance. Therefore, the development of this device has great practical significance and potential applications. At the present time, the researching and developments of vehicles has become a hot topic.

Intelligent vehicle is an integrated system that combined a set of model technologies such as environmental awareness, planning decision-making, multi-level driver assistance functions and so on. Intelligent vehicles using the perception of the surrounding environment and its own internal a priori knowledge to identify the path want to travel and plan out the driver's decision-making and which is the base to achieve the goal of automatic vehicle or auxiliary driving.

The machine vision is one of its key technologies for smart vehicles, its main task is to identify and track the boundaries of lanes. So in this article, more attention is paid on the self-contained navigation system under the situation of the environmental awareness and regulatory environment.

In recent years, a lot of research work at home and abroad has been done and made a number of achievements in research of the machine vision. But there is still a great deal of work to do, especially for problems about the recognition accuracy and stability.

In Reference [1], a practical method for road detection is proposed. Many features of vehicle are introduced to detect the preceding vehicles. According to the characteristics of lane marks and vehicles, the improved Sobel operator convolution kernel is introduced in the image preprocessing to enhance the robustness of the algorithm.

Reference [2] presents automatic lane detection and navigation using pattern matching model. The parabola model is deployed for lane modeling and lane type classification with optimal parameters. The proposed system shows desirable performance for lane detection and classification. Disadvantage of this approach is too complicated to calculate, especially when the road turns there.

Reference [3] used a recognition method based on Morphological Filtering. This method used only the "watershed" algorithm to locate the image pixel value gray level change point (the road borders.) The advantage of this approach is it not need to consider the gray threshold value of the image, drawback is that there is no way to adopt a common boundary constraint conditions.

There are many other algorithms, such as Ref. [4] [5] [6], etc., these documents are all gave some new algorithms, but there is no so detailed.

This paper presents a model of the linear lanes, and makes a discussion about the detailed implementation steps. The algorithm was verified at the end of this paper.

II. THE PRINCIPLE OF LANE DETECTION ALGORITHM

This algorithm uses a linear lane boundary model, adopting this boundary model has the advantage of being calculated faster, and more robustness. When the roads are bending, we can divide the lanes to small sections, and each section can be calculated as a short straight line. As the algorithm is for highway condition, while the road and the highway will not be a sharp turn in the situation, so this model can guarantee the normal operation of the algorithm when turning, and outputs accurate results.

When detective the linear lane boundaries we used the Hough transform. The Hough transform is a relatively fast way of searching a binary image for straight lines.

III. THE IMPLEMENTATION OF LANE DETECTION ALGORITHM BASED ON OPENCV

A. Brief introduction of OpenCV

OpenCV is an open source computer vision library. The library is written in C and C++ and runs under Linux,

978-0-7695-4304-8/10 $26.00 © 2010 IEEE
DOI 10.1109/GCIS.2010.120

Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other language. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision including factory product inspection; medical imaging, security, and user inter face, camera calibration, stereo vision and robotics.

B. Details on the algorithm

The flow chart of main program is shown as Fig.1.

There are about 4 steps of this algorithm, I) Capture and input video signal II) preprocessing on the video signals, including cut, histogram equalization, image enhancement in a certain region of grayscale III) get the linear elements of the image using the Hough transform, then deal with these lines. IV) Output the centerline of the road based on these lines that got on step III).

1) Capture and input video signal

The algorithm adopts a camera as an input device, and obtains the road image by it. In OpenCV, the initialization of the camera is packaged as a function, you can initializes the camera by using the *cvCamInit ()* function. OpenCV can also deal with video files by making the files as a virtual camera, though the *cvCaptureFormAVI ()* function. After using the functions above, by finding automatically the drivers on the PC, OpenCV call the camera or a video file as input.

2) Preprocess the input video signals.

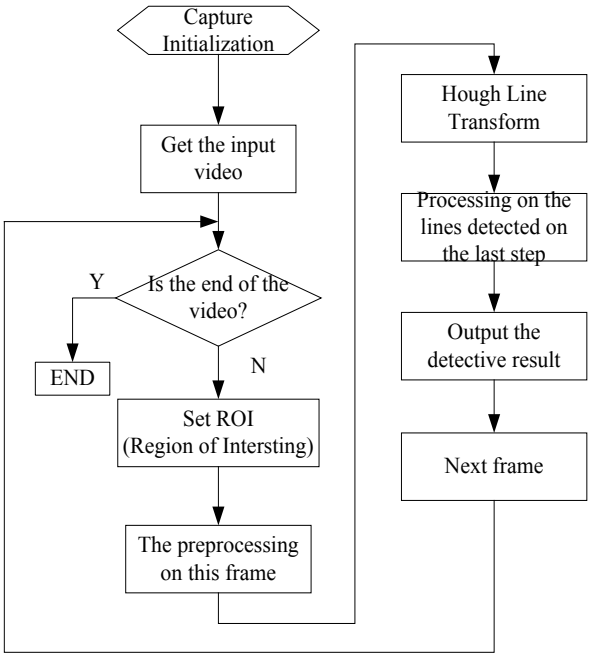

Figure 1. Flow char of main program

On this step we divided the program to the flowing parts.

Firstly, set the region of interesting (*ROI*) of the input video: in order to increasing the processing speed and delete some useless elements of the input video (the sky, buildings, etc). The ROI is 3/4 of the image form the bottom. We can set the ROI by using the *cvSetROI()* function of OpenCV. After set the ROI, this program will only deal with the ROI of the image.

Secondly, Image enhancement. After transform the color video to a grayscale video, we enhance image on a certain region of grayscale. What we want from the video is the marking lines on the roads both the yellow marks and the white marks, after we transform the color image to grayscale image, the grayscale of the marks is a narrow region. So we can enhance the narrow region for more useful information, and shield useless information in the same time.

Finally we get the binary image by using the Canny operator. The advantages of Canny operator is low false positive rate, high positioning accuracy and suppression of false edges

As shown in Fig.2, (a) shows the original image, (b) shows the grayscale image (c) shows the enhanced image (d) shows the binary image got by the Canny operator.

3) Hough transform

There will a Hough transform on the image by using the *cvHoughLines2 ()* function. The detailed statement of Hough Transform is as follows.

*CvSeq * cvHoughLines2(CvArr* image,void *line_storage,int method,double rho,double theta,int threshold,double param1=0,double param2=0);*

The first argument is the input image. The second argument is a pointer to a place where the results can be stored. The next argument *method*, can be CV_HOUGH_STANDARD,CV_HOUGH_PROBABILISTIC,or CV_HOUGH_MULTI_SCALE for (respectively) the stander Hough transform (*SHT*), the progressive probabilistic Hough transform(*PPHT*) and the multiscale variant of SHT. In this paper we use the PPHT. The next two arguments, *rho* and *theta*, set the resolution desired for the lines. The *threshold* value is the value in the accumulator plane that must be reached for the routine to report a line. For the PPHT, *param1* sets the minimum length of a line segment that will be returned, and *param2* sets the separation between collinear segments required for the algorithm not to join them into a single longer segment. As has noted above, we can see that the parameters which have the greatest impact on the final results are param1 and param2. For the convenience of adjustment in accordance with different operations, we can create 3 track bars by using the *cvCreateTrackbar()* function of OpenCV, and its values are returned to the threshold, param1 and param2.

We can get the straight lines of the image by using Hough transform, shown as Fig.3. As we can see from the Fig.3, there are too many lines on the image, and most of

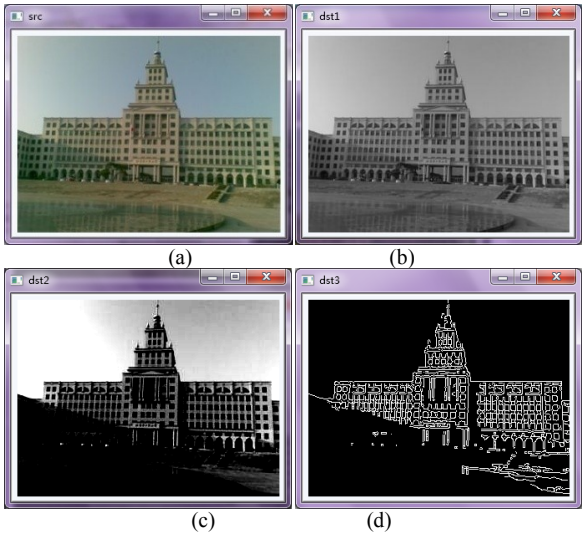

Figure 2. The example of prprocessing on input image

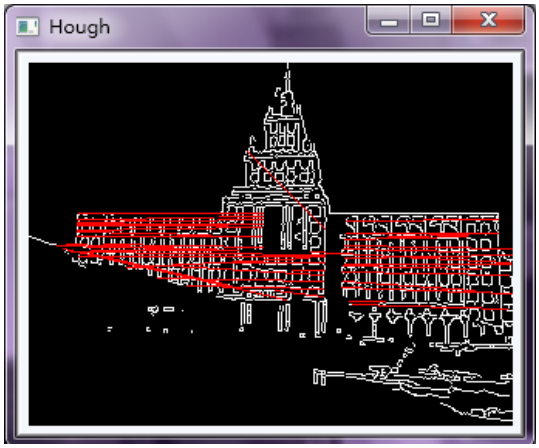

Figure 3. The example of Hough transform

them are useless. However, by using this algorithm we can get useful lines through the following method:

a) Group:

The lines that got by the Hough transform are divide to two groups by if the slope is greater than 0. As we use the linear road model, it is can believed that the slopes of left boundaries and right boundaries are different. The Hough transform give out two points of the line: the start point *(x0 y0)* and the end point *(x1,y1)*. So we can get the slope by *(y1-y0)/(x1-x0)*,and we marked the slop as tanP. It is worth noting that if *|tanP|<0.1*, we think that the line is parallel to *x-axes*. When the line is parallel to x-axes it is mostly because the obstruction on image, so we ignore these lines. At last if *tanP>0*, it will be stored in the queue *LeftLine*, otherwise it will be stored in the queue *RightLine*.

b) Get the centerline

Find the maximum length of a straight line in each group, this line is the line that we are looking for. And recorded respectively left and right borders as *maxPl* and *maxPr*. At the same time note the image frames recorded as *NumOfFrame*.

Straight line will be re-drawn in the original image.

4) Output the centerline

Output the centerline of the road based on these lines that got on step *3)*.

If we reconsider step *3)* we can see there is a problem that we can't ensure that we can't ensure that get both the right boundary and the left boundary in each frame. To get the correct result when the detective on boundaries is not so good, we do some adjustments depending on the follow situations.

Case (1)

We detected both the right boundary and the left boundary, in this situation the detection is perfect, what have to do is just find the centerline between the right boundary and the left boundary. As these two boundaries have the opposite directions. So we have to get the centerline by using the following equation;

$$Cx0 = \frac{1}{2}(Lx0 + Rx1); Cy0 = \frac{1}{2}(Ly0 + Ry1)$$
$$Cx1 = \frac{1}{2}(Lx1 + Rx0)\ ; Cy1 = \frac{1}{2}(Ly1 + Ry0)$$

In the eq. we mark the left line ,the right line and the centerline respectively as *((Lx0,Ly0),(Lx1,Ly1))* , *((Rx0,Ry0),(Rx1,Ry1))* and *((Cx0,Cy0),(Cx1,Cy1))*

Case(2)

We detected the right boundary but the left (or detected the left one but the right this algorithm tread that in the same way).

Firstly, determine that whether the results meet the conditions for approximate calculation. If it doesn't meets the system will break out. The conditions are follows (*FramesNow* representative of number of frames at this point):

① *FramesNow>30.*

Because at the beginning the system is not stable enough, we will ignore

②*FramesNow-NumOfFrames<150.*

It means that there are too many errors in this period.If we are going to get the similar results, the deviation would be very large, and so if that happened the system will be re-initialization.

Secondly, Make the left boundary that detected in the last frame (or several frames earlier) as the left boundary that would be detected in this frame.

Finally, get the centerline by reference to the detected right boundary and the approximate left boundary.

Case (3)

Neither the right boundary nor the left boundary has been detected.

For this case the line would be calculate out in the same method that *case (2)* has used, but the constraint has been strengthen: *FramesNow-NumOfFrames<100*. This strengthen on the constraint is ensure that the system is

stable. If this case happened, we can supports that the system is going to unstable, so we wouldn't allow that happen too much.

IV. RESULT OF EXPERIMENT

Under the guidance of the above algorithm, we compile the corresponding program. The programming environment is Visual C++ 2008 Express Edition+ OpenCV 2.0. The hardware environment is Pentium 4 3.0GHz, RAM 1G. After experiment, the system can process 15 frames in one second, meet the real-time requirements. The accuracy is about 90%. So this algorithm is feasible

As the system running there will be 3 windows shown as Fig.4. The first is the original image and the centerline. The red number on lower left is the number of frames at this point. The second window is the grayscale image, as the ROI has been setting yet, so its height is only 75% of first window. The last window is the binary image. Besides there is a command window, it will show some useful information about the video.

There are some examples on different conditions Shown as Fig.5.

Figure 4. The ruselt when the program running

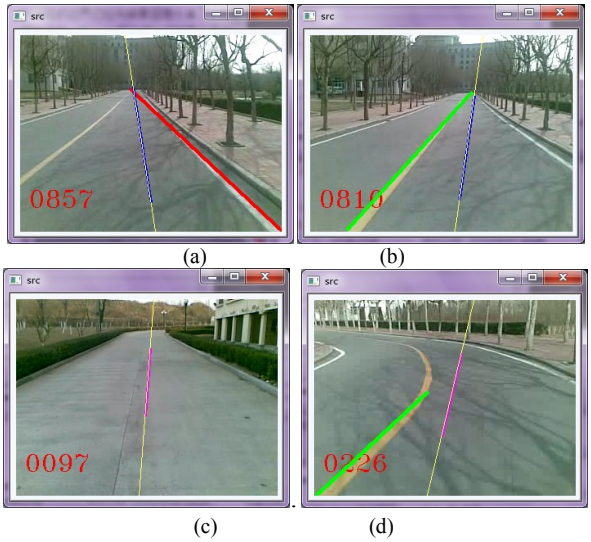

Figure 5. Rusult of differnet conditions ,(a) only the left boundary has been detectived (b) only the left boundary has been detectived (c) neither the right boundary nor the left boundary has been detectived (d) When the lane bend

REFERENCES

[1] Dezhi Gao, Wei Li, Jianmin Duan, Banggui Zheng " Practical Method of Road Detection for Intelligent Vehicle" International Conference on Automation and Logistics ,Shenyang, China August 2009

[2] C. LONGJARD, P. KUMSAWAT, K. ATTAKITMONGKOL AND A. SRIKAEW "Automatic Lane Detection and Navigation using Pattern Matching Mode", Proceedings of the 7th WSEAS International Conference on Signal, Speech and Image Processing, Beijing, China, September 15-17, 2007

[3] Serge B and Michel B, "Road segmentation and obstacle detection by a fast watershed transform," *Proceedings of the Intelligent Vehicles'94*:pp296-301,1994

[4] J. Kosecka, R. Blasi, C. Taylor and J. Malik. "A Comparative Study of Vision-Based Lateral Control Strategies for Autonomous Highway Driving". *In IEEE Int. Conf. on Robotics and Automation*, pp. 1903-1908, May 1998.

[5] Samadzadegan, F. and Sarafraz, A. and Tabibi, M. "Automatic lane detection in image sequences for vision based navigation purposes", IEVM06, 2006

[6] Chris Kreucher and Sridhar Lakshmanan, "LANA: A Lane Extraction Algorithm that Uses Frequency Domain Features," *IEEE Transactions on Robotics and Automation*.vol. 15, no. 2,pp343-350, April 1999

[7] Gary Bradski & Adrian Kaebler "Learning OpenCV", O'Reily Media, Inc. 2008