Lane Detection for Intelligent Driver Assistance System
Group #22

Authors
Yawen Deng (yd298), Yuzhe Sheng (ys766), Min Zheng (mz474)

Abstract
This project develops an algorithm that performs the lane detection in realistic driving environment. The algorithm provides guideline vector of the two side marks of the lane occupied by the vehicle. The result is evaluated using test images with manually marked ground truth, and the program is shown to achieve acceptable rate of success with acceptable precision in detection in real world driving environment where the lighting, shadowing and lane curve complexity are different. The algorithm along with the optimized parameter set therefore demonstrates its potential in the applications such as Lane Departure Warning in Intelligent Driver-Assistance Systems.

1. Introduction

This project considers the lane detection and analysis in realistic scenarios, particularly the driving environment in Ithaca. Although many literatures have addressed the similar problem, this project will also provide a quantitative evaluation of the method under carefully selected metrics. This project thus will provide detailed experiment and discussion on the extraction of fundamental information needed for the intelligent vehicle systems.

The project is based on single image frames instead of image sequences. The goal is to detect the landmarks of the center lane from images. For each image, the output is two single-pixel-width spline curves representing the left and right side of the landmarks of the center lane (the lane that the vehicle is on). Two sides of the lane marks will be treated separately in the algorithm and in evaluation. The scenario is constrained: the camera is fixed at a location and orientation. Therefore the location and size of the target lane is relatively constant. Also, the changing rate of the lane width is known or can be approximated. These basic assumptions simplify the problem and aid us in the design of algorithms and parameter tuning.

There remain some challenges too. For example, there are two sides of the lane markings, whose correlation cannot simply be described analytically, thus should be treated separately. Though constrained, the environments tested vary in terms of lighting, shadowing, and lane geometry complexity. These have posed difficulties in the algorithm development and evaluation.

The ground truth for the test image is marked manually. We design the evaluation function that combines Hausdorff distance method and FPC index and apply it to images with different environmental complexities, so that we can obtain quantified knowledge of the performance of our algorithm more comprehensively.

1.1 Background

The goal of this project is to extract edge information, particularly the lane-marks of both side of the lane that the vehicle is on, from images taken in the real driving environment. The results are shown in figure 1. The image on the left is the original image, and the image on the right is the original image with detected lane marking pixels shown in the red. The evaluation of the algorithm and results will focus on whether enough lane information can be extracted, together with the precision of the detection.

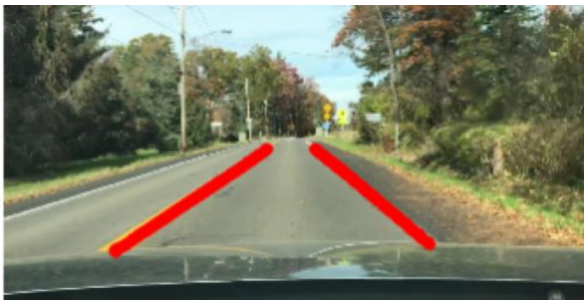
	
Input Image	Output Image

Figure 1: Results of Algorithm

This task aims to aid the advanced driver-assistance system, so that the algorithm design, evaluation, and discussion will be based on the context of real driving scenario.

Advanced driver-assistance systems (ADAS) has been drawing great attention and widely applied in practice recently. The system is designed to strengthen the safety and help drivers avoid accidents by alerting the drivers of potential danger. It is believed to greatly reduce car accidents by reducing human errors, therefore save millions of lives.

Approximately 1.24 million people die every year from automobile accidents around the world. Meanwhile more than 20 million people suffer from injuries due to traffic accidents [12]. Among these accidents, human error is the most critical reason and accounts for 93% of crashes, according to the 2008 National Motor Vehicle Crash Causation Survey [12].

One of the most popular and practical applications in ADAS is Lane Departure Warning and Avoidance, which alerts the driver when vehicle drift is detected. Apart from laser sensors and infrared sensors, Video and Computer vision based lane-detection is an essential part of input sources for ADAS. The system requires a camera input of the road and then it performs detection of the lane markings from real time camera images. This technology has been widely implemented by some leading automobile companies, including Nissan, Toyota, Honda, BMW, etc.

1.2 Previous Work

Lane detection has been an active research area in the field of intelligent vehicle applications, and various algorithms have been proposed to detect the lane markings based on different techniques. One of the commonly used techniques is built upon the Hough transformations. The pipeline shown in Figure 2. McDonald 2001 [1] implemented straight line Hough transform based on the assumption that the curvature of curvy roads tends to be small on highways. In that case, the approximated lane markings can be reconstructed from the votes in the Hough space. One of the problems associated with Hough transform is the computational costs, and there have been modified Hough transformations proposed in the past to address this problem. For example, Hierarchical Additive Hough Transform (HAHT) is able to significantly reduce the computational costs in real time lane detection task [2]. In addition, Saudi et al 2008 [3] used randomized Hough transform to achieve fast lane detection. However, in our project, the road images we test do not necessarily consist of only straight lane markings, nor is it possible to model the curves of lane markings as easily parametrized polynomials. Therefore, for the purpose of generalization of our method and the complex curve of roads, we do not limit the lane markings to be analytic shapes.

Another commonly chosen method is based on edge detection. A variety of gradient-based edge operators coupled with thresholding have been used to detect the boundaries of lane markings in images [4]. However, edge detectors alone are sensitive to the noise, and different parameters need to be tuned such as the kernel elements, the kernel sizes, and the threshold values for the post-processing. Usually, edge detections are implemented together with Hough transform for better accuracy. Low et al 2014 [5] detected the lane markings in real-time videos using the Canny edge operator followed by Hough transform to eventually make the detection process more robust to noise. Othman et al. 2010 [13] has also implemented a similar pipeline with image frames.

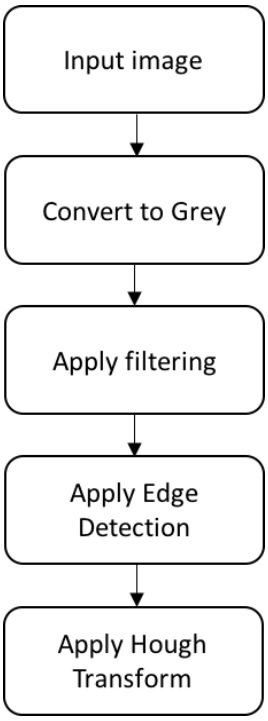


Figure 2: pipeline of conventional methods

In addition, Wang et al 2003 [6] proposed a B-spline-based method to detect the boundaries of lane markings. S-spline is able to form arbitrary shape given a set of control points, but it is not robust enough to noise and is sensitive to the choice of control point locations. Therefore, in this paper, Canny/Hough Estimation of Vanishing Point (CHEVP) algorithm is implemented to determine good initial positions for the B-snake. Another method based on the spline model is designed by Wang et al 2000 [7]. In this study, Catmull-Rom spline is used to model the parallel lanes that are subject to the perspective effects. This method is more generalized compared to other well parameterized road models such as straight lines and polynomials.

Apart, some lane detection methods improve the accuracy by estimating the vanishing points of the two lane markings in an image. For example, Kong et al 2009 [8] proposed an adaptive soft voting scheme coupled with confidence-weighted Gabor filters to implement the vanishing point estimation and this method has been proven to be efficient and effect even in challenging conditions. Additionally, Moghadam et al 2012 [9] implement an optimal local dominant orientation method together with four Gabor filters to estimate the local dominant orientation at each pixel location, from which the vanishing point can be estimated based on voting scheme.

Lane detection can also be achieved using color-based model. Chiu et al 2005 [10] proposed a color-based segmentation to detect the lane boundary following the determination of threshold in the region of interest using statistical method. Study by Sun et al 2006 [11] showed that the HSI color model representation to detect road surface within a ROI, with modified difference of intensity distribution of a row using fuzzy c-means algorithm, is robust and efficient.
There is no commonly used evaluation method and in most papers evaluation methods are not specified due to the lack of ground truth. In McDonald 2001 [1], the accuracy is decided by human operator (the human decide whether the result is correct or not). We decide to design an evaluation algorithm by ourselves.

1.2 Proof of concept example

We decided to use sliding window as opposed to Hough transform to detect the lane markings because we try to model the lane curves as some complicated shapes instead of some analytic shapes such as straight lines and polynomials. Hough transform is also computationally expensive. In addition, we apply nonlinear morphological filters such as the maximum filter to connect broken curves. Linear filter such as Gaussian will not work in our case in that Gaussian filter tends to blur the lane marking pixels with the background in the far distance. The following figures show that Gaussian filter has eliminated pixels that belong to the lane markings, whereas the maximum filter is able to connect broken pieces of the lane markings to make the detection more accurate. Therefore, for the purpose of sliding window, maximum filter does not cause much trouble because it is more likely each window can detect some of the lane marking pixels. In Figure 3, the Gaussian filter blurs the lane marking pixels and the gaps between broken regions are larger than the gaps from max filter shown on the right.

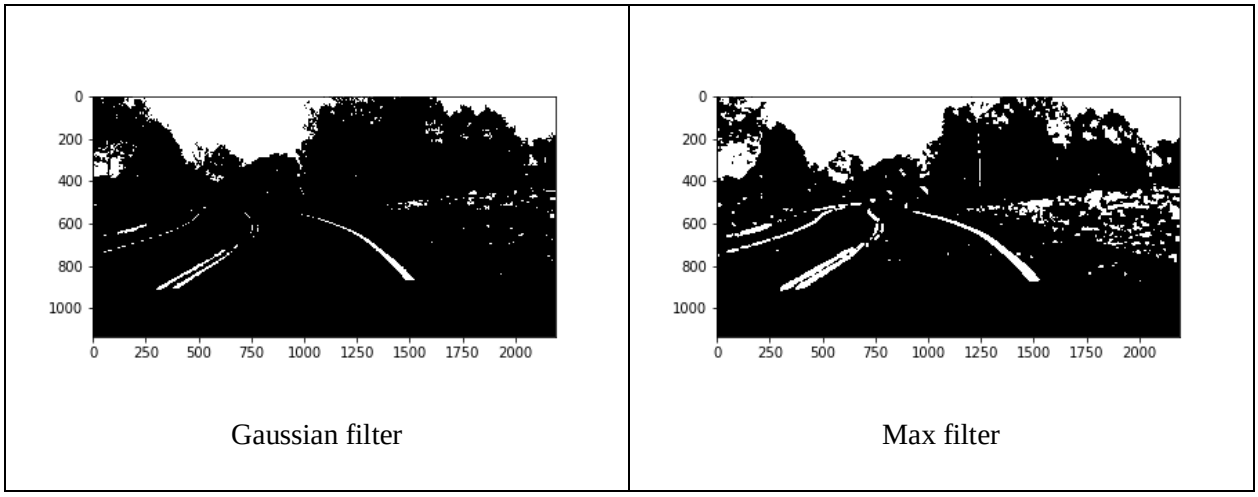


Figure 3: Gaussian filter vs. max filter

Based on the previous work, Hough transform is a commonly used tool to detect the straight lines. However, due to the complexity of the lane markings of our images, Hough transform tries to model the entire lane marking as a straight line, or a easily parametrized curve. To extract a more complex curve line in the real world driving scenario, a general Hough transform approach may be more applicable. However, in Python OpenCV and VisionX, only straight line or circle Hough transform packages are available. The following Figure 4 demonstrates an unsuccessful attempt of applying straight line Hough transform in this project. In Figure 4, all the parameters such as the ROI vertices and thresholds are

identical to those applied in our algorithm. The detected straight lines capture most part of the right side of desired lane mark, but miss the left lane mark.

Figure 4: Straight line Hough transform

1.3 Overview

In the following section of the report, discussion will focus on the development of algorithms, and the selected evaluation function based on the project objective. The ground truth marking and the selection of test images will also be introduced. Finally, result and conclusion by the evaluation will be drawn to demonstrate the performance of the algorithm, along with suggestions on the possible modifications and improvement that can be made.

2. Methods

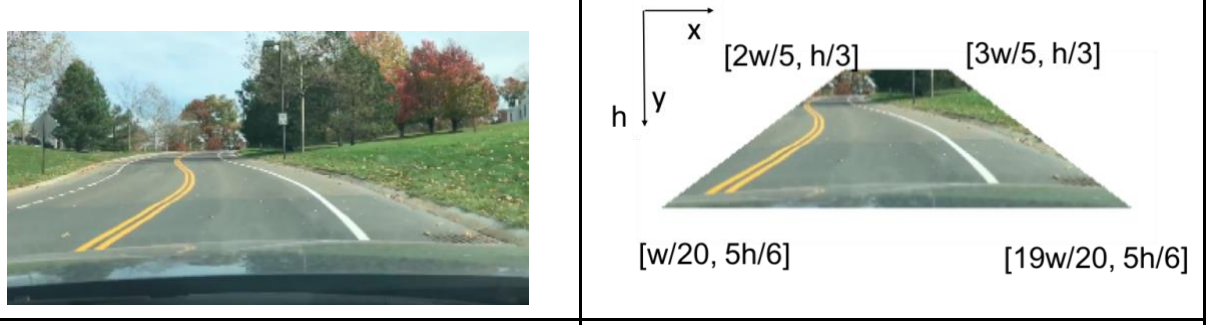
In our system, a camera is fixed on the front-view window to capture the road scene. The algorithm first extract Region of Interest (ROI) to reduce computational cost. Then we apply yellow and white masks to segment the lane markings from the background and output the image as a binary image. We use maximum morphological filter to thicken and to reduce the gaps between the lane markings. After this, we use sliding window to detect line pixels and represent them as a single-pixel-width line using spline. The camera calibration and perspective transform is not performed in this experiment, since the goal is simply detect the lanes, but no computation or analytical interpretation will be applied.

The evaluation experiment is conducted to understand the performance of the algorithm on both simple and complex environments. The algorithm is expected to work well in a simple environment, and it achieves a good rate (~80% success) in general. Considering the practical application demand of the algorithm, the result is defined as satisfactory if when comparing to the manually marked ground truth, the auto-marking contains acceptable amount of information and also achieves desired precision. The detail of the evaluation will be explained in the following sections.

2.1 Algorithm

Figure 5: Pipeline of the algorithm

In Figure 5, we can see the pipeline of our algorithm. We first select a region of interest to reduce the computational cost, then segment the lane markings into a binary image, which is fed into the max filter. Eventually, we can make lane marking decisions using sliding windows, together with spline interpolation. Considering that the environment is mostly constrained, and that the noise on the road side are troublesome, a region of interest is selected to simplify the procedure. Figure 6 shows the effect of ROI. Four vertices are selected by examining several images. The vertices are selected loosely to avoid problem of cutting useful lane information in the testing images and to avoid noise from the background. The ROI appears as a trapezoid, since it needs to get rid of the other lanes marks other than the main two. The four vertices are [⅖ width, ⅓ height], [⅗ width, ⅓ height], [$^{1}/_{20}$width, ⅚ height,], [$^{19}/_{20}$ width, ⅚ height]. From Figure 6, all information outside the polygon is treated as useless and discarded, and only the information confined by the polygon is retained.



Before ROI	After ROI

Figure 6: Effect of the ROI

Figure 7 shows the effect of masking. Since lanes markings are mainly yellow and white, we apply yellow and white masking to segment the lane markings from the background. We first detect the color range from the training images and then define thresholds. To apply yellow mask, the algorithm first convert the image from RGB to HSV. The reason to use HSV image is that it separates hue, saturation and value/brightness in an image, and HSV model is more robust to shadows. Hence it will work regardless of lighting changes. To apply white mask, the algorithm first convert the image into grayscale. Then we filter the image based on threshold for the color white. We output our image as binary image with 1 denoting the potential candidates of lane marking pixels and 0 denoting the background, as shown in Figure 7. By converting it into binary image, the computational cost will be further reduced. The range of yellow is [0, 100, 170] ~ [255, 240, 255] in RGB. The range of white 210 ~ 255 in grey scale.

	
Before Masking	After Masking

Figure 7: Effect of Masking

Figure 8 shows the effect of max morphological filter. After masking, the continuous lane marking regions become broken, which may cause problems in the following steps. Therefore, we use a maximum filter that assigns the maximum pixel value in the neighborhood defined by the kernel size to the center pixel to increase the region of the lane markings. The lane markings become thicker and the gaps among broken regions are reduced after being filtered. The kernel size is tuned as 9, since smaller kernel takes too long to perform, and larger kernel enhances the background noise.

	
Before Max Filter	After Max Filter

Figure 8: Effect of the Max Filter

The histogram which records the number of times each x coordinate appears is shown in Figure 9. From this point, the left and right lane marks are treated and detected separately. The start points of the lane markings are firstly detected. The histogram of the image pixels within the y range (750:900) is computed. The left and right parts are split at the mean of histogram, denoted as mid in Figure 9. Then for each of the left and right side, the mean x coordinates are computed, denoted as mid_l and mid_r. The maximum histogram bin is searched from 0 to mid_l for left side, and from mid_r to the end for right side. The resultant locations, denoted by the circles, are the start points of the left and right lane marks. Such method is developed as a solution to the highly curved cases, where the C-shape road gives high value in the histogram.

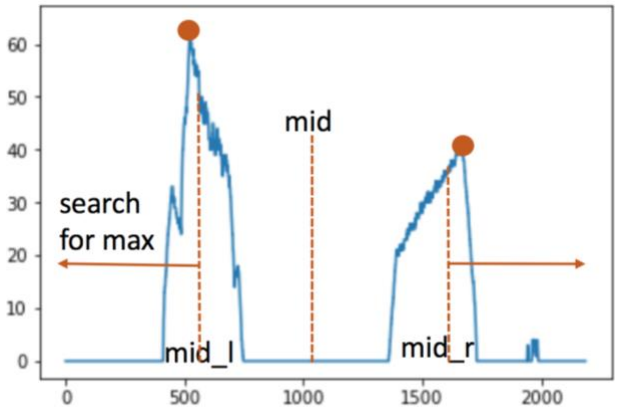


Figure 9: Search for the two starting x locations for the sliding windows based on histogram

Figure 10 shows the comparison of choosing start points in different ways. If we simply search the maximum value from mid point, the results are shown on the right side. The start points chosen in this way is far from the true start points. The left part shows the results of searching start point from mid_l and mid_r. The results are quite close to the true start points.

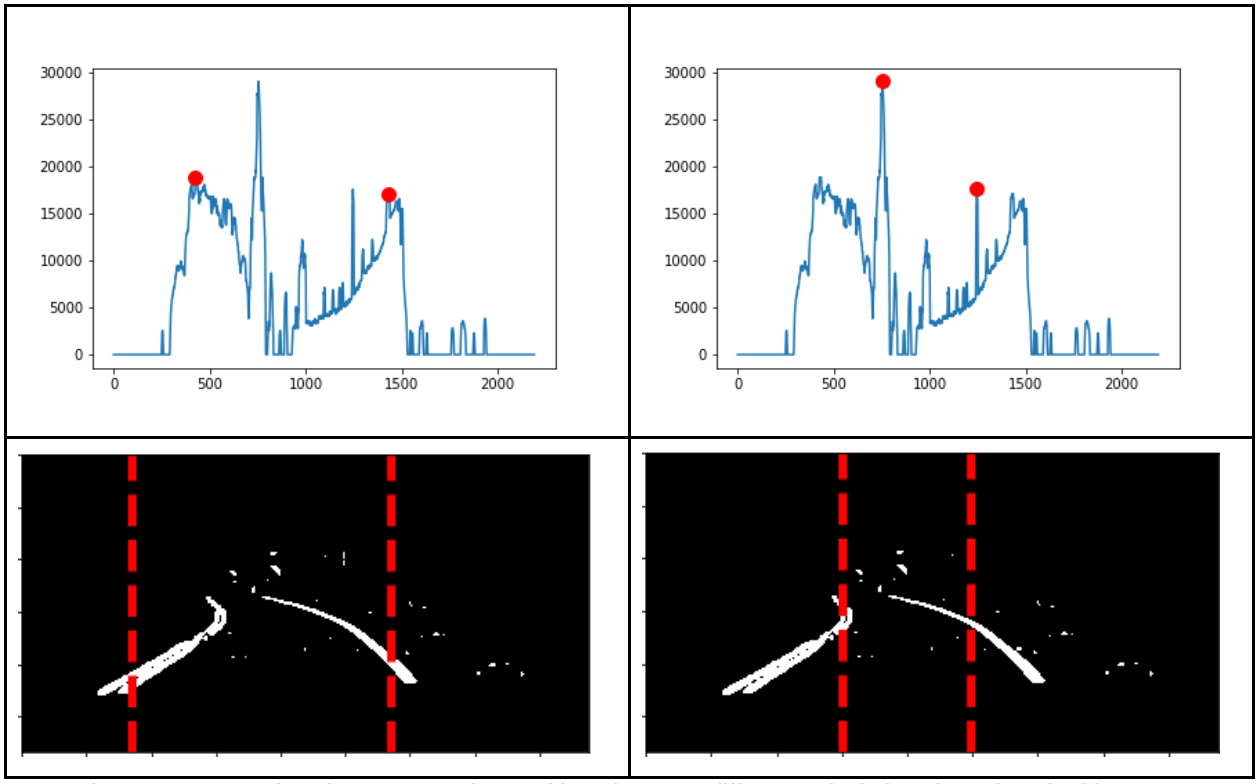


Figure 10: Comparison between starting positions from two different calculations based on the histogram

With the start points of both sides of the lane, two separate sliding window procedures are applied to detect the pixels on the lane marks. The result are shown in Figure 11. A total number of 30 windows (n_windows) will be applied, and the window height is computed according to the approximated height of the lane and window number. The start points determine the x location of the first set of windows. The window is constructed to search the lane mark pixels from the very bottom of the region of interest of the image. In Figure 11, the green boxes represent the 30 sliding windows, the red regions are marked as the left lane side, and the blue regions are the right lane marks.

In the window, all of the foreground(lane) pixels are detected, recorded, and averaged to determine the next window position(x_next). Two threshold values, named minpix_top and minpix_window is defined. For every window, the number of lane pixels found on the top edge of the window(y_low) is compared to the minpix_top; the numbers of pixels detected in the whole window is compared to minpix_window. If both minpix_top and minpix_window are met, the lane is likely to continue in the next window. Therefore the x_next is selected as the mean x value of lane pixels at y_low. If only the minpix_top is met, x_next will be the mean x value of lane pixels at y_low. If only the minpix_window is met, x_next will be mean x value of lane pixels of the entire window. If none of the threshold is met, the lane may have reached an end, so x_next is the same as the current x. Besides, we also add a momentum term to control how much further the window should be moved.

Also, considering the shrinking of the lane width due to the view perspective, the window width should decrease along the detection too. Therefore, the window width decreases by 160/ n_windows each time. The thresholds (minpix_window, minpix_top) decreases 5/n_windows each time.

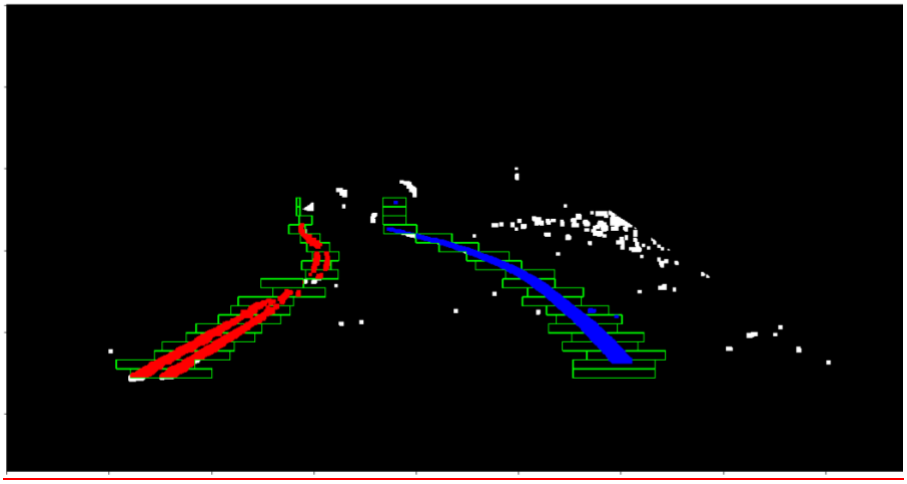


Figure 11: Sliding windows and labeled pixels for the left and right lane markings

For the effectiveness of output result and the convenience in evaluation, the results should be a single-pixel width curves, meaning that for a range of y, every y value has exactly one x coordinate. Figure 12 demonstrates the modeling the process. For every y value, we choose the median value of the corresponding x coordinates. We choose median instead of mean because it is more robust to outliers. Then we use spline to find the missing points and to smooth the results. The detected pixels are shown in the light blue shaded region, and for each existing y coordinate, we denote the medians of the x coordinates associated with that y coordinate by dark blue dots. We obtain the yellow dot shown by the interpolated point by evaluating the spline fit function at that missing y coordinate.

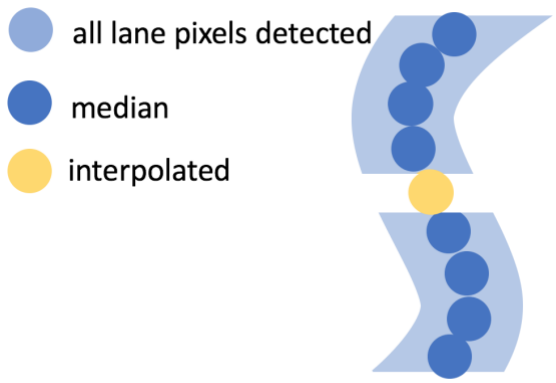


Figure 12: Demonstration of the interpolation for the output results to fill the missing point

2.2 Experiment

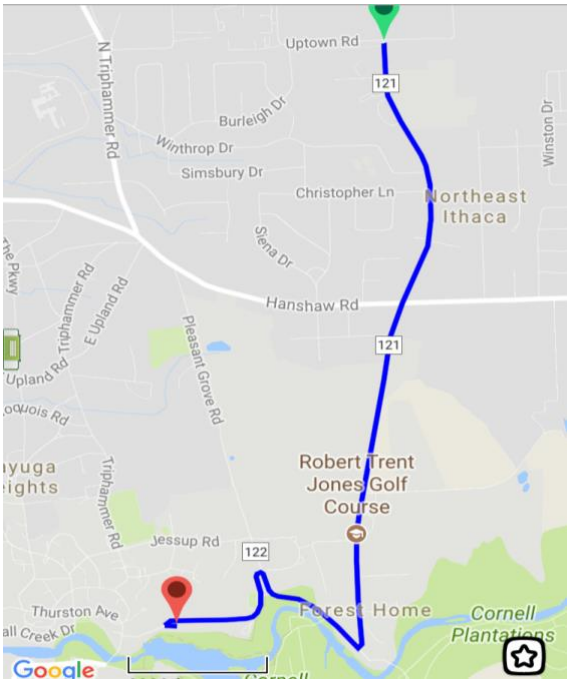


Figure 13: The driving routes from which the videos are captured

The video is taken from the paths indicated by the map above shown in Figure 13 in Ithaca, NY. The video is taken using an iPhone 7 camera, mounted on the tripod that stands on the top of the dashboard. During the experiment, the camera has not changed location or orientation. The average speed on campus is 21.25mph (9.5m/s). The total length of the video is about 4 mins. Then, total of 80 images were uniformly sampled from the video to avoid bias. So that on average, one frame is selected from a 3 seconds segment of video. With such sampling rate, issues like high similarities in test images can be avoided, therefore evaluation is more reliable. 5 bad image frames are discarded, for being highly blurred due to camera focus issue, or containing unexpected pedestrian crossings.

2.2.1 Documented Data Set

Figure 14: The original image within labeled ground truth

For each image, a minimum number of 3 and a maximum number of 8 marking is made for each side of lane marks. Based on our images, 3 labels to 8 labels are necessary and sufficient to capture the geometry of the actual lane markings. In addition, we intend to label the centerlines of each lane markings, similarly to our detected results. The ground truth marking should also try to include the endpoints (of the min and max y coordinates) of both lane marks, so that the comparison is more reliable. In Figure 14, the blue dots represent the manual labels of the centerlines of each the left and right lane markings.

The ground truth labeling result is represented by a set of points. In order to make comparison to the resultant curve, the ground truth is linearly interpolated, so that for every y coordinate the lane marking is labeled, there is exactly one x coordinate at that specific y coordinate. Such interpolation is also demonstrated to give a more reliable evaluation result as shown in Figure 15. Although all of the ground truth labels are on the resultant curve, in fact there may be significant variations between the result and the actual lane markings, which will not be detected without the interpolation.

Meanwhile, per the assumption made before that for the ground truth labeling, every line segment between ground truth markers is approximately straight. Meanwhile, by linear interpolation, the original ground truth markers will be retained.

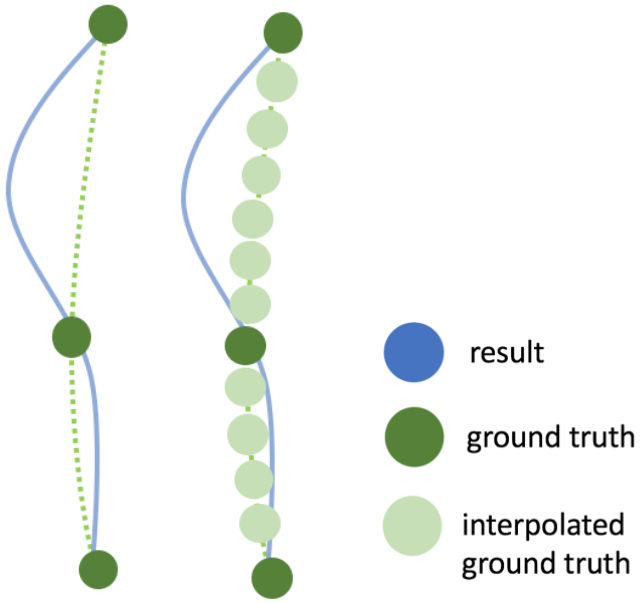


Figure 15: Illustration of ground truth interpolation

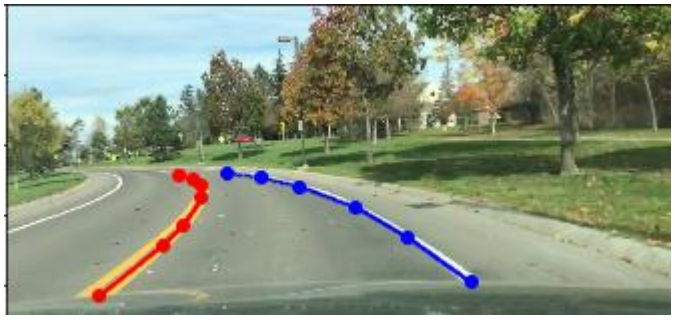


Figure 16: Labeled lane markings with the linear interpolated lines. The red dots and red lines are for the left lane marking, whereas the blue dots and blue lines are for the right lane marking. Dots are the actual manual labeled ground truth, and lines are interpolated lines.

Also, analysis on the human marking is performed to better understand the comparison of human marking and auto-marking later. For a set of test images, each member of the team marks each of them, and the result is also compared using the methodology described above.

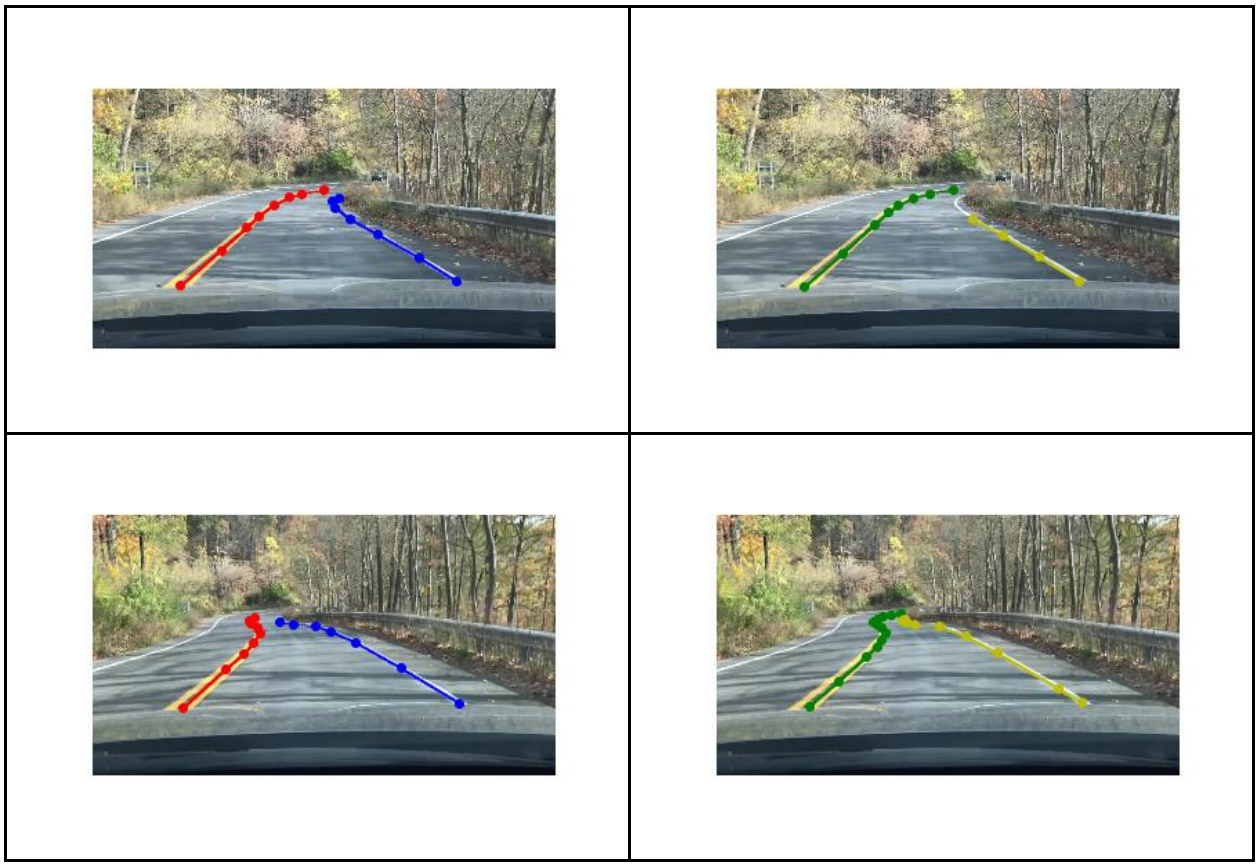


Figure 17: Illustration of ground truth labeling for different people

In Figure 17, the red and blue labels are generated from one of our team members, whereas the green and yellow labels are from another member. From the figures above, we can see that the two team members have approximately the same labeling scheme. For 10 test data that are labeled by more than 1 member, the average FPC of human labeling is 0.94. Therefore, our manual labeling serves as a valid ground truth to be compared to the outputs from our program.

2.2.2 Evaluation function and hypothesis

The evaluation should examine the practical purpose of the result. Thus, the first part of the evaluation should consider the detected lane similarity in the y-axis. Once the result is proved valid, the precision of the detection is measured by the similarity in the x-axis. The evaluation is separated into x and y axis, instead of applying the traditional Hausdorff distance in the Euclidean space, since our evaluation method will be beneficial to better understand the cause of failure or imprecision.

The first part of the evaluation considers the detected lane similarity with ground truth in the y-axis. This step is to check if the result contains enough information to be useful practically. For example, if the result only consists of very few points, even if those points perfectly matches with the ground truth, such

result is still not satisfactory, since our objective is to provide the vehicle lane information to guide or assist driving.

The result and the ground truth line segments are projected in the y-direction, and their similarity is compared as demonstrated in below in Figure 18. False negative and false positive are both undesirable, and should receive penalty in the evaluation. Hence the fraction pixels correct (FPC) is applied, with A denoting the ground truth and B denoting the comparison data:

$$FPC = \frac{|A| - (|A \cap \bar{B}| + |\bar{A} \cap B|)}{|A|}$$

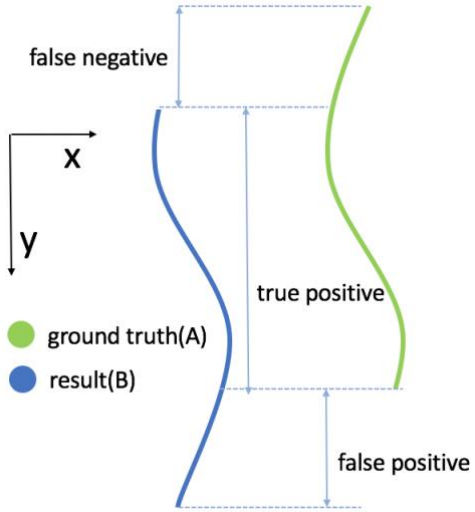


Figure 18: Illustration of the evaluation of similarity in the y-direction

A detection for an image is considered successful if for both side of the lane, the FPC is greater that 0.7. The average reaction time for driving is at least 0.7s. In our experiment video, we approximate that the driving speed is around 10 m/s, so that the driver should have at least 7m reaction distance. Per our approximation using the reference (road signs), if the road is straight, 70% of the height before vanishing point should result in at least 7m. If the road is curved, the reaction distance given should be even longer.

Once the result is considered as a valid detection, the similarity of the detected curves and the ground truth in the x-axis is then evaluated. Firstly, any ground truth data that is higher or lower than the auto-marking is discarded, since the penalty has been applied in the previous evaluation stage. To demonstrate that the algorithm is able to ensure driving safety, the result should be considered satisfactory if the maximum of its error does not exceed a threshold of 100 pixels. Given the resolution of the images is 1080P, and the average lane width is about 300 pixels, such threshold corresponds to ⅓ of the lane width.

Demonstrated in Figure 19, for every point in resultant curves, the distance to the ground truth in the x direction at the same y coordinate is denoted as di, where i ranges from 1 to the intersected pixel numbers.

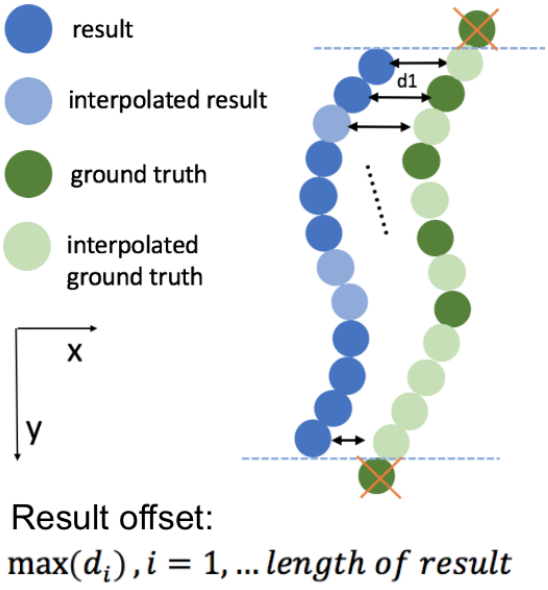


Figure 19: Illustration of the evaluation precision (offset in the x-direction)

The proposed hypothesis is that the algorithm achieves success detection rate of 80%, since previous work by Othman et al 2010 [13] has achieved 80% success detection rate for curved single lane environment, which is most similar to our environment setting. Although such performance is not considered optimal to be applied in the industry, it can serve as a goal for the current stage and can be improved by future works.
Also, the algorithm is expected to achieve less than 100 pixels of maximum offset in the x-direction, as it corresponds to ⅓ of the lane width in reality.

2.2.3 Experiment procedure

The images appear in different environment complexities. For example, the shadowing and lighting variation is different; the complexity of the lane curve is different too. While some lane marking segments are almost straight, various lane markings appear to be complicated curves.
However, the images are not grouped by their complexities due to the lack of quantifiable criteria for complexity measurements of the environment. We also separate the images into training image set and testing image set randomly to avoid bias. For the algorithm development, the parameter tuning, and optimization, we selected around 10 images. For testing, we use 65 images, considering the tradeoff of evaluation reliability and the time consumption for manually marking. Figure 20 shows two potential candidate images for the easy environment setting and for the complicated environment setting.

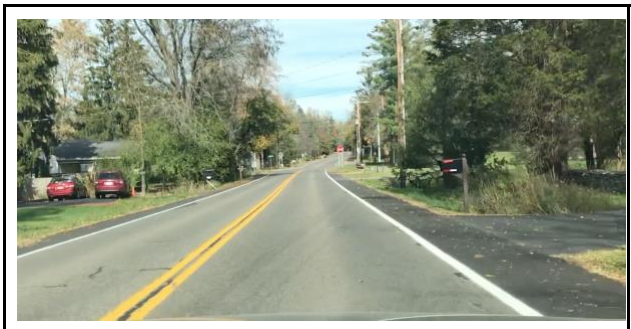
	
Simple environment	Complex environment

Figure 20: Different environment complexities

3. Results

For all the test images, the successful detection rate is 86%, which has exceeded the hypothesis. The average FPC among all test images is 0.73. The FPC is 0.89 among the 56 successful detection, with the same FPC(0.89) for left and right lane separately. Therefore left or right lane has the same success rate of being detected with enough lane marking information extracted by our program. Note that the FPC for the success detection is near the human labeled ground truth(0.94). Overall, the algorithm demonstrates a satisfactory success detection rate.

For all the success detection (56 images), the average offset distance in the x-direction is 83.4 pixels. Considering that the image size is 1920x1080 pixels, the algorithm has been demonstrated to have high performance and reliability in reality, by exceeding the hypothesis for the precision. The algorithm has also shown its robustness with most cases with heavy shadows. In Figure 22, both the left and right images are subject to serious shadows from the background trees. However, our detection is still successful in labeling the lane markings.

The histogram in Figure 21 also demonstrats that for the 83.9% of detected result, our algorithm has achieved a high precision detection with offset less than 100 pixels. However, there are still 3 images that has an offset more than 250 pixels, which may be less reliable in reality. If those low-precision detection are therefore considered as non-successful detection, the algorithm is able to achieve 81.5% success detection rate with high precision.

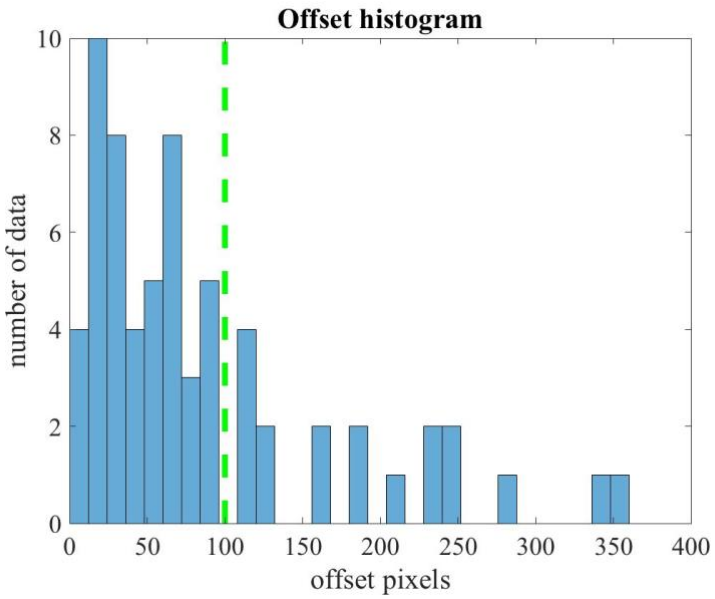


Figure 21: Histogram that records the number of testing images corresponding to each offset number

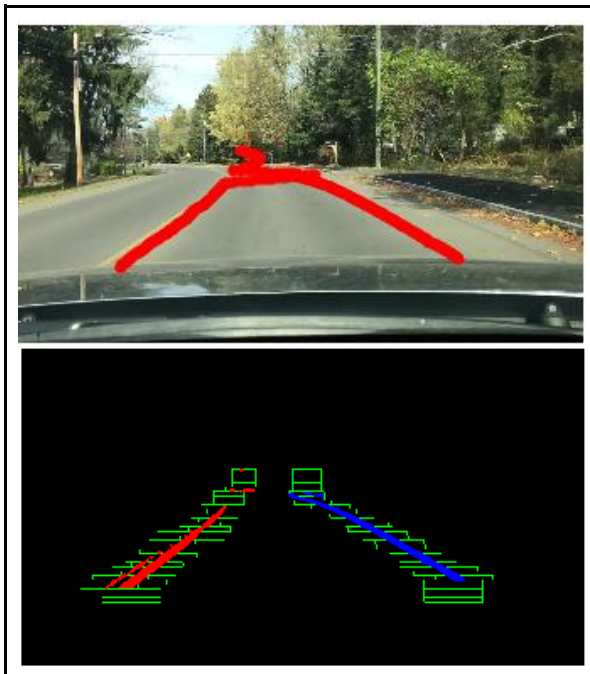
Figure 22: Successful lane detection with the influence from the shadows.

4. Discussion

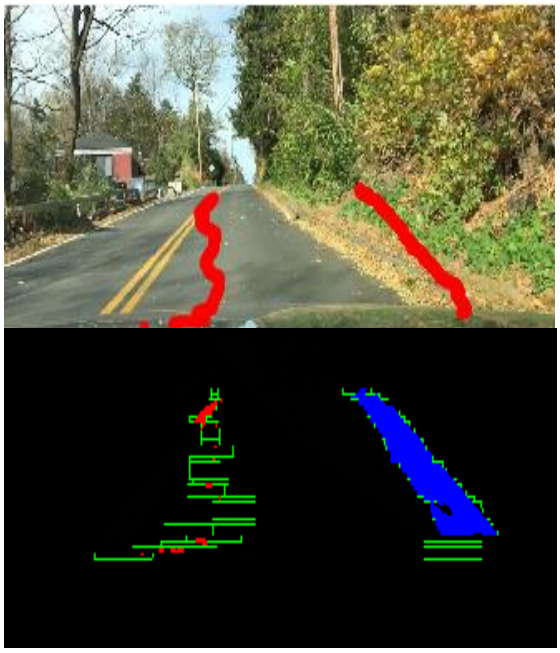
As we can see from the previous results, our detection algorithm performs better on the straight roads than on the curvy roads. This is expected because sliding windows tend to confuse pixels from the background noise with the real lane marking pixels. This is more problematic for curved roads, in that the other lane markings become extremely close to the lane markings to be detected, so our algorithm will misclassify the other lane markings as the desired lane markings. However, with consideration of the size of the HD images we obtained, the maximum distance between the detected pixels and the real lane pixels is still small.

There are mainly four reasons for failure. 1) Incorrect thresholds. In our algorithm, we apply yellow and white masking and output the image as a binary image. For some images, the threshold might be too strict so that some lane pixels are filtered out. In Figure 23 a), the very end part of left lane pixels are filtered out; 2) Incorrect start points. The choice of start points are crucial to sliding window. In Figure 23 b), the start points are incorrectly chosen. Since the leaves happen to have the same color as lanes, the algorithm picks leaves instead of lanes; 3) Shadows. Shadows influence the color of lanes and the algorithm might filter out lane pixels with shadows. In Figure 23 c) the shadow parts of lanes are filtered out; 4) Noises.

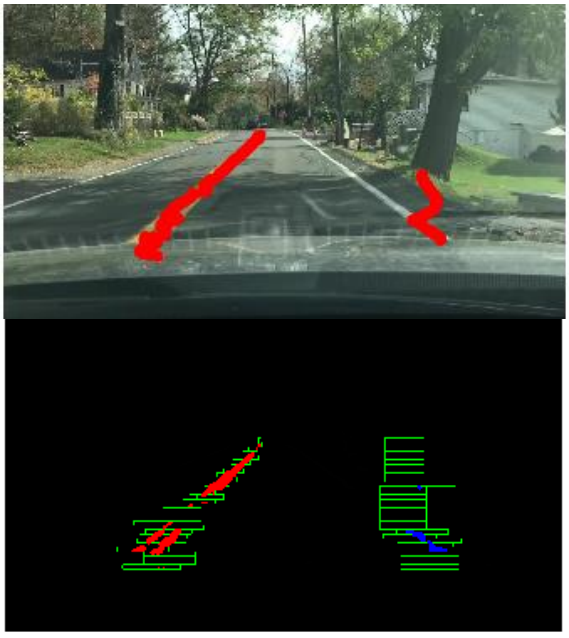
Noises like trees and grass have similar color as lanes. The algorithm misclassifies those noises as lane pixels. In Figure 23 d), trees near the very end of lanes are misclassified as lane pixels.



a) Incorrect Thresholds



b) Incorrect Start Points



c) Shadows

d) Misclassification of Noises

Figure 23: Demonstrations for the failed detection

The percentage of each failure reason are shown in Figure 24. Among failure images, 40% of them are caused by noise, 30% are caused by incorrect start points, 20% are caused by incorrect threshold, and 10% are caused by shadows.

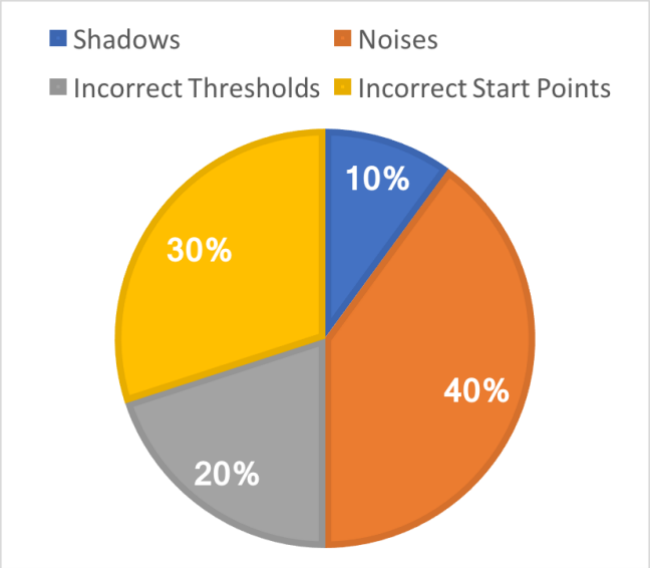


Figure 24: Pie graph for the failed reasons

Conclusion

According to the testing result and analysis, the algorithm is shown to achieve over 80% rate of successfully and precisely detecting lane marks in varied environment complexities, including environment that contains heavy shadow. The average detected lane offset is less than ⅓ of the real lane width, also showing its potential in safety enhancement systems for real world driving scenario. Hence, the algorithm has demonstrated its capability and robustness of providing guideline vector of the two side marks of the lane occupied by the vehicle. The evaluation function applied also reflects the project objective well. Overall, from the aspect of algorithm pipeline and evaluation function, the project has demonstrated its potential of being applied in the development of Lane Departure Warning in Intelligent Driver-Assistance Systems.

References

[1]. J. B. McDonald. “Application of the hough transform to lane detection and following on high speed roads.” Proceeding of Irish Signals and Systems Conference, pp. 340-345, 2001.
[2]. R. K. Satzoda, S. Sathyanarayana, T. Srikanthan, and S Sathyanarayana. “Hierarchical additive Hough transform for lane detection.” IEEE Embedded Systems Letters, Vol.2, Issue 2, pp. 23-26, 2010.
[3]. A. Saudi, J. Teo, M. H. A. Hijazi, and J. Sulaiman. “Fast lane detection with randomized Hough transform.” International Symposium on Information Technology, 2008.
[4]. Y. Xuan, B. Serge, B. Michel. “Road tracking lane segmentation and obstacle recognition by mathematical morphology.” Proceedings of the Intelligent Vehicles, pp. 166-170, 1992.
[5]. C. Y. Low, H. Zamzuri, and S. A. Mazlan. “Simple robust lane detection algorithm.” International Conference on Intelligent and Advanced Systems, 2014.
[6]. Y. Wang, E. K. Teoh, and D. Shen. “Lane detection and tracking using B-Snake.” Image Vision Computing, Vol. 22, Issue 4, pp. 269-280, 2004.
[7]. Y. Wang, D. Shen, and E. K. Teoh. “Lane detection using spline model.” Pattern Recognition Letters, pp. 677-689, 2000.
[8]. H. Kong, J. Audibert, and J. Ponce. “Vanishing point detection for road detection.” IEEE Conference on Computer Vision and Pattern Recognition, 2009.
[9]. P. Moghadam, J. A. Starzyk, and W. S. Wijesoma. “Fast vanishing-point detection in unstructured environments.” IEEE Transactions on Image Processing, Vol. 21, Issue 1, pp. 425-430, 2012.
[10]. K. Chiu, and S. Lin. “Lane detection using color-based segmentation. ” Proceedings of Intelligent Vehicles Symposium, 2005.
[11]. T. Sun, S. Tsai, and V. Chan. “HSI color model based lane-marking detection.” Intelligent Transportation Systems Conference, 2006.
[12]. "Global Status Report on Road Safety 2013: Supporting a Decade of Action", 2013.World Health Organization.
[13]. O. O. Khalifa, I. M. Khan, A. A. M. Assidiq, A. –H. Abdulla, and S. Khan. “A Hyperbola-Pair Based Lane Detection System for Vehicle Guidance.” 2010.

Program Documentation

A. Python functions:

1. **GRAYSCALE**

NAME

Grayscale **-** Grayscale transform function

SYNOPSIS

Input: img, color image (3D array) (3 channels).

Output: gray-scale image (1 channel).

DESCRIPTION

Applies the Grayscale transfor. This will return an image with only one color channel you should call plt.imshow(gray, cmap='gray').

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

2. **GAUSSIAN_BLUR**

NAME

Gaussian_blur - Gaussian filter

SYNOPSIS

Input: img: gray scale image

kernel_size: integer.

Output: blurred gray scale image using the Gaussian Filter with specified kernel size.

DESCRIPTION

Applies a Gaussian filter to smooth the gray scale image.

SEE ALSO

Max_blur

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

3. **MAX_BLUR**

NAME

Max_blur - Morphological max filter

SYNOPSIS

Input: (1-channel)binary image.

kernel size: Integer.

Output: gray scale binary image after the max filter.

DESCRIPTION

Morphological filtering: within the kernel, assign the center pixel the value equal to the maximum pixel values in the neighboring region defined by the kernel size.

SEE ALSO

Guassian_blur

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

4. **REGION_OF_INTEREST**

NAME

region_of_interest - Applies an image mask based on region of interest

SYNOPSIS

Image_name: A String that stores the name of the image

Input: image: arrays

vertices: an array that stores the vertices of the polygon.

Output: Image the same size as the input image but with pixels outside the polygon defined by the vertices having value of 0.

DESCRIPTION

Applies an image mask.

Only keeps the region of the image defined by the polygon formed from `vertices`. The rest of the image is set to black.

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

5. **STARTPOINTS**

NAME

startPoints - Computes the starting x-coordinates for sliding windows

SYNOPSIS

Input: image, image_name (a String that stores the name of the image)

Output: left_x: left initial x-coordinate of the sliding window center. right_x: right initial x-coordinate of the sliding window center.

DESCRIPTION

Computes the starting x-coordinates (left and right) as the initial x-coordinates of the centers of the sliding windows (left and right) based on the histogram.

SEE ALSO

Hist_mean, slidingWindow

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

6. **HIST_MEAN**

NAME

hist_mean - calculate the mean x-coordinate based on the histogram

SYNOPSIS

Input: histogram: each x value has a bin, representing the number of times the x value appears.

start: Integer that denotes the lower boundary of search area.

end: Integer that denotes the upper boundary of search area.

Output: the average x location (integer) between start and end.

DESCRIPTION

Calculate the average location (x-coordinate) between start and end based on the histogram.

SEE ALSO

region_of_instrest, slidingWindow

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

7. **SLIDINGWINDOW**

NAME

slidingWindow - detect the all the pixels belonging to the lane markings

SYNOPSIS

Input: image: original image that has 3 channels.

roi_image: retaining the pixels values within polygon

left_x: (integer) the initial x coordinate of the left sliding window center

right_x: (integer) the initial x coordinate of the right sliding window center

n_window: (integer) the number of windows

a: (float) the momentum term that is used to control how much further the following window will be moved according to the location difference (When the x-coordinate of the next window center is delta_x away from the current x-coordinate of the sliding window, move the next window FURTHER by a*delta_x).

Image_name: String that stores the name of the image.

Output: leftx: (integer array) the x-coordinates of all the pixels belonging to the left lane markings.

lefty: (integer array) the y-coordinates of all the pixels belonging to the left lane markings

rightx: (integer array) the x-coordinates of all the pixels belonging to the right lane markings.

righty: (integer array) the y-coordinates of all the pixels belonging to the right lane markings

DESCRIPTION

detect the all the pixels belonging to the lane markings (left and right)

For each window, find the direction in which the following window should be moved. Then model all nonzero pixels inside that window as the lane marking pixels.

SEE ALSO

region_of_instrest, hist_mean

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

8. **LANE_DECISION**

NAME

lane_Decision - take the average of x position for each y

SYNOPSIS

Input: image: original 3 channel color image.
leftx: (integer array) the x-coordinates of all the pixels belonging to the left lane markings.
lefty: (integer array) the y-coordinates of all the pixels belonging to the left lane markings.
rightx: (integer array) the x-coordinates of all the pixels belonging to the right lane markings.
righty: (integer array) the y-coordinates of all the pixels belonging to the right lane markings.
Image_name: A string that stores the name of the image
Output: pos_xleft: the mean, or the linearly interpolated x-coordinates of detected lane markings associated with each left y-coordinate.
Y_left: an increasing interger array that stores all y-coordinates bounded by lefty.
pos_xright: the mean, or the linearly interpolated x-coordinates of detected lane markings associated with each right y-coordinate.
Y_right: an increasing interger array that stores all y-coordinates bounded by righty.

DESCRIPTION

For the detected coordinates of x and y, for each y, take the average of x position as the x-coordinate at that y-coordinate.

SEE ALSO

slidingWindow

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

9. **EVALUATION**

NAME

evaluation -Evaluate the performance of the algorithm

SYNOPSIS

Input:
image: original image to be drawn on.
pos_xleft: Float array that stores the x coordinates of the detected left lane for each y.
Y_left: an increasing interger array that stores all y-coordinates bounded by detected left y locations.
pos_xright: Float array that stores the x coordinates of the detected right lane for each y.
Y_right: an increasing interger array that stores all y-coordinates bounded by detected right y locations.
image_name: a String that stores the name of the image.
Output:
An array of String that stores the name of the image, the FPC for the left and right lane

markings, the FP rate for the left and right lane markings, the FN rate for the left and right lane markings, the x-coordinate maximum displacement, and whether the detection is successful or not depending on the FPC.

DESCRIPTION

First find the Reeve's Index (FPC) of each of the left and right lane markings. If FPC > 0.7, then the detection is considered as successful. For successful detection, find the maximum x-coordinate between the the detected pixels and the ground truth.

SEE ALSO

mean_PosX

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

10. **MEDIAN_POSX**

NAME

median_PosX -find the median x position corresponding to a y position

SYNOPSIS

Input:

X: Integer array that stores ALL the detected x coordinates

Y: Integer array that stores ALL the detected y coordinates

Range: Sorted Integer array that stores all interger y coordinate bounded by Y

flag: 0 or 1. 0: do not insert -1 for missing y position. 1: insert -1 for missing y

Output:

pos_x: Integer array that stores the MEDIAN x-coordinate or -1 for each y-coordinate (-1 if and only if no x-coordinates are detected for that y value).

pos_y: Integer array that stores all y coordinates

DESCRIPTION

For each element r in Range, find the MEDIAN x position corresponding to that r (y-coordinate) there is any. Store -1 if there is no that specific y coordinate for the ground truth ONLY. For the detected pixels, do NOT store -1 in that missing y coordinate.

SEE ALSO

evaluation

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

11. **SIMIALRITY_MEASURE**

NAME

Similarity_measure -calculate FPC

SYNOPSIS

Input:

Y1: detected height range stored in an Integer array (sorted)

Y2: ground truth height range stored in an Integer array (sorted)

Output:
FP: the ratio between the number of FP and the ground truth height
FN: the ratio between the number of FN and the ground truth height
FPC: the Reeves index

DESCRIPTION

Y1 denotes the detected height range, and Y2 denotes the Ground truth height rangesimilarity_measure calculates the fraction of pixels correct (FPC or the Reeves Index)
FPC = (|Y2| - |Y2 intersects (not Y1)| - |(not Y2) intersects Y1|)/(Y2)

SEE ALSO

evaluation

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

12. **FILL_MISSINGPOINT**

NAME

fill_MissingPoint - Fill in the missing x coordinates

SYNOPSIS

Input:
X: Integer array that stores the x-coordinates or -1 (when no x locations are detected from that y)
Y: All y coordinates

DESCRIPTION

Fill in the missing x coordinates if there is any by linear interpolation

SEE ALSO

evaluation

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

13. **PROCESS_IAMGE**

NAME

process_image - Process the image with the whole algorithm pipeline

SYNOPSIS

Input:
image: colored image
image_name: a String that stores the name of the image
Output:
same as lane_decision function
pos_xleft, Y_left, pos_xright, Y_right that stores the x y coordinates of the left and right lane markings.

DESCRIPTION

Process the image. First apply yellow and white mask. Then threshold the image to obtain an enhanced lane marking image. Then apply the maximum filter to connect broken pieces of lane markings. Then find the ROI of the filtered image given a specified vertices array. Then use sliding window method to detect all pixels associated with the left lane and right lane pixels.

AUTHOR

Y. Deng, Y. Sheng and M. Zheng

B. Matlab functions

1. **LABEL**

NAME

label - label and store ground truth

SYNOPSIS

Input: folder name, mouse click input for x and y coordinate of ground truth points

DESCRIPTION

load and display image with imshow, and allows the user to label on the image separately. The labeled file is stored in csv file with name_l and name_r, where name is the name of the original image.

SEE ALSO

evaluation

AUTHOR

Y. Deng, Y. Sheng and M. Zheng