REQUIREMENT ANALYSIS DOCUMENT

COMPREHENSIVE ASSIGNMENT

Team: Imperium

Bokhodir Urinboev (leader):	U1610249
Dilorom Alieva:	U1610061
Feruza Latipova:	U1610072
Rakhmatjon Khasanov:	U1610183

TABLE OF CONTENTS

INTRODUCTION

Pose of vehicle with respect to road lanes plays an important role in advances of fully autonomous vehicle navigation. Nowadays, Computer Vision provides us implementation of easy and low-cost solutions of road lane detection. However, there will be challenges such as worn-out lanes, shadows, different light-conditions. In this project we have to suggest our own algorithms which tackles all these problems. There are several steps could be done to make lane line detection such as: removing noise, edge detection, mask image in order to extract important part of image, complete the line and predict the turn.

REQUIRENMENT ANALYSIS

Lane Detection Algorithm

In this project in order to make our self-driving car more realistic we have to complete Lane Line detection in real-time. Lane Line detection is possible with the Computer Vision techniques using OpenCV. We can use OpenCV to process the input images to discover any lane lines kept within, and also to make a lane representation. We will use several methods to detect lines and by getting left and right lines to predict turning of our vehicle.

IMAGE DENOISING

For lane-detection in self-driving cars we have first get rid of image noise to edge-detection functions work properly. There are many ways to reduce image noise for example: bilateral filtering and median blurring. However, in this project we will use Gaussian Blur to each frame to achieve removing noise in tiny pixels using as it is most useful and fastest. In image Processing Gaussian Blur is used to reduce an amount of noise in an image. This phase would eliminate noise from the picture and tiny information such as distant items irrelevant to our objective. We can implement Gaussian Blur using OpenCV library with the GaussianBlur() function.

EDGE DETECTION FROM BINARY IMAGE

Next step we have to perform is detecting edges. In order to acquire desired results first we have to convert our RGB image to gray scale image using OpenCV cvtColor() function. Then we have to apply full frame binarization then apply Canny edge detection with tuned thresholds. The Canny algorithm identifies edges on a picture by looking for rapid color shifts between a pixel and its neighbors. The blur and grayscale phase should help to differentiate main lane lines. The outcome will produce a black and white image.

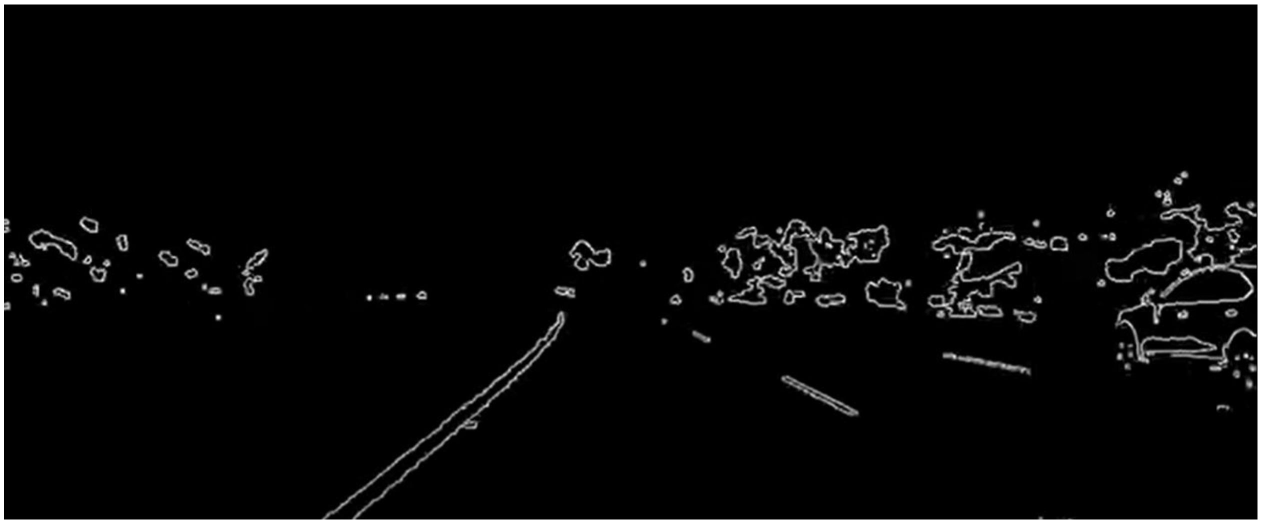

Figure 1 The result of Canny edge filter applied image

REGION OF INTEREST

It is necessary to eliminate as much of the noise within a picture as possible. Given the camera's location and orientation, we can assume lanes are in the lower half of the frame. As we need to detect the edges of only lower part of image, we have to get rid of unnecessary part of frame. We can isolate the region easily by using a trapezoid shape. Although hardcoding the best way to search for lines can be enticing, lines will travel about while you drive.

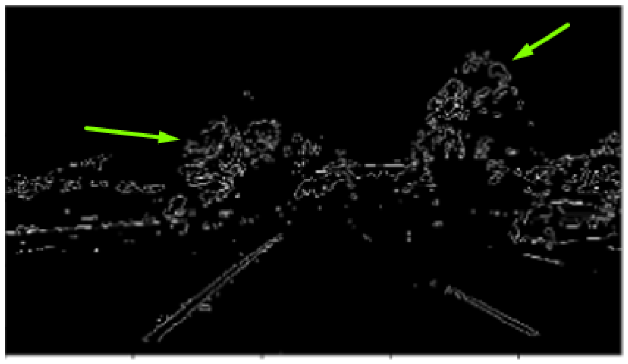

Figure 2 Part of image that should be removed

HOUGH LINES DETECTION

After detecting edges, next step should be done is detecting lines. So, the task we have to perform detect line and draw semi-transparent lines. To connect edges using line which was detected using Canny edge detector we use Hough Line detector. Hough Line detector is used to in image processing to detect shapes and lines. Using probabilistic Hough lines, we define where lane lines are positioned on the road. The algorithm Hough transform extracts all the lines that move through each of our edge points and group them by similarity. The feature Hough Lines P in OpenCV returns an array of endpoints ordered lines (x1, x1, x2, x2). The result of used Hough lines should be in picture below

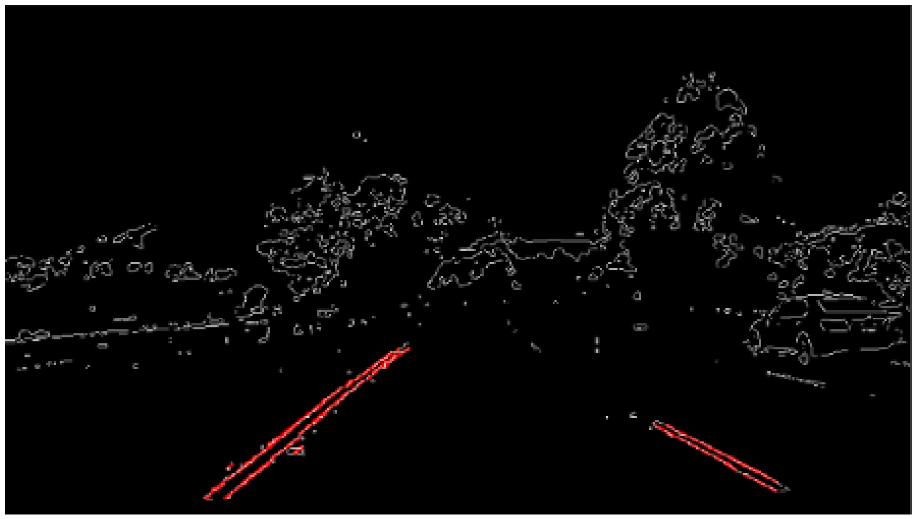

Figure 3 Result of detected line using Hough method

LINE SEPERATION

After detecting lines from given frame, we have to predict whether to turn left or right. In order to do that we have to determine which line is left and right. We can separate left or right line it with the help of polar systems. If slope is negative number it means left line and vice versa. As soon as we separate left and right lines our next task will be to merge small close together. We can merge several close lines and remove unnecessary line using regression. Linear regression is an effort to find the optimal partnership for a dot party. We get a line which passes from each point at the closest possible distance. On both the left and right series of lines we implement linear regression. Our output should be in given figure below

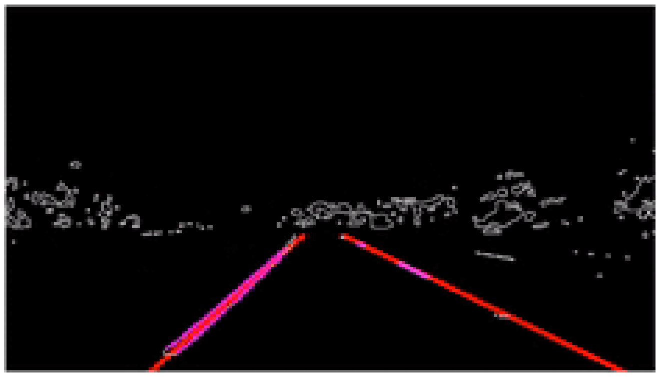

Figure 4 Output of regression line

DRAWING THE COLOR LINE

Next step should be performed is drawing complete line in detected regression lines. It could be done using OpenCV line() function and gives us opportunity to draw detected lines on real-time.

PREDICT TURN

The last step in order to complete the task is predicting the turn of moving car according to the video. Our project should display whether moving car is turning left or right.

CONCLUSION

Vision based lane line detection is one of the most essential part of autonomous car. In this project we are going to detect real-time vision-based lane line using several algorithms mentioned before using target video. We use are going to use OpenCV library to create our own algorithms to detect lines and predict the turn of car using majority mathematical calculations. To enhance accuracy of lane line detection we have solve several problems such as:

- Removing noise from tiny pixels,
- Extract ROI from given frame in order to reduce the amount of data should be processed and to improve the efficiency of the detection time
- Detecting each line correctly and if lines exist in both directions separate them to left and right lines
- Finally, show turned direction

After performing this tasks our project is going to work accurate and detect lane lines from target video.

REFERENCES

[1] https://www.youtube.com/watch?v=gWK9x5Xs_
[2] https://drive.google.com/file/d/1RMq9j_mxkqPX_C4OYdoJmdfnp15ef0S/view?usp=drivesdk
[3] https://towardsdatascience.com/carnd-project-1-lane-lines-detection-a-complete-pipeline-6b815037d02c#.hcsizymg8
[4] https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
[5] https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/
[6] https://www.learnopencv.com/hough-transform-with-opencv-c-python/