SW SPECIFIC DETAILED DESIGN DOCUMENT

COMPREHENSIVE ASSIGNMENT

Team: Imperium

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

TABLE OF CONTENTS

INTRODUCTION

Purpose

As our task from Capstone design course is to implement fully autonomous car. However, in autonomous vehicle system one of the most important feature is detecting road lane line in real time. Computer Vision gives us a feasible and low-cost solution in road lane line detection. The aim of this project is to detect road lane-line despite we could face several challenges in road appearance with varying light-conditions, selecting ROI from model and turn prediction while riding.

Scope

In this project we mainly focus on feature extraction from road image and reduce the complexity of calculation of turn prediction techniques. Also, we are going to implement algorithms which will solve challenging road appearances. Our system detects left and right road lane lines separately and capability of predicting turns. After completion of project we could solve main part of autonomous vehicle problem.

Overview

This document is divided into various subsections. The sections of the Software Design Document are:

1. Introduction
2. General Description
3. Specific Requirements
4. Conclusion and Future Work

GENERAL DESCRIPTION

In General description section we will give a brief overview of functions and working principles. Also, you will get overall view of a project.

Product Functions

There several task should be included in our system in order to complete road lane line detection. Our team mainly focuses on completing several steps as mention below:

- First and most essential part is to get rid of noise in lane line pixels from given video in order to get accurate detection results.
- Next step we should perform is to detect edges. To get desired results first we first convert our RGB image to gray scale image and convert to binary image. Then, apply Canny edge detection algorithms
- One of the most key steps is to get ROI interest. We get rid of unnecessary part of stream video to make our algorithms work faster and detect only required past of image.
- In order to detect lane-lines we use Hough line detection
- As we achieve to detect lane-lines, we separate each line to left and right side.
- The most valuable feature is turn prediction.

Technology used

Nowadays Computer Vision is powerful to work with videos and images. Computer Vision based road lane line detection gives us low-cost solution as the vehicle pose can be derived from the detection. We choose Python language to implement project and took advantage from OpenCV library functions.

Constrains

The Road Structure constrains the product. Also, poor quality of road gave us a lot of hard work to detect. Also detecting will be problematic when road lines was not visible and was old. Predicting the motion should be clear as car changes direction

SPECIFIC REQUIREMENTS

Hardware requirements

Our project should be implemented using Raspberry Car as planned before. But as the result of COVID-19 outbreak we are supposed to complete only software part of the project.

Software requirements

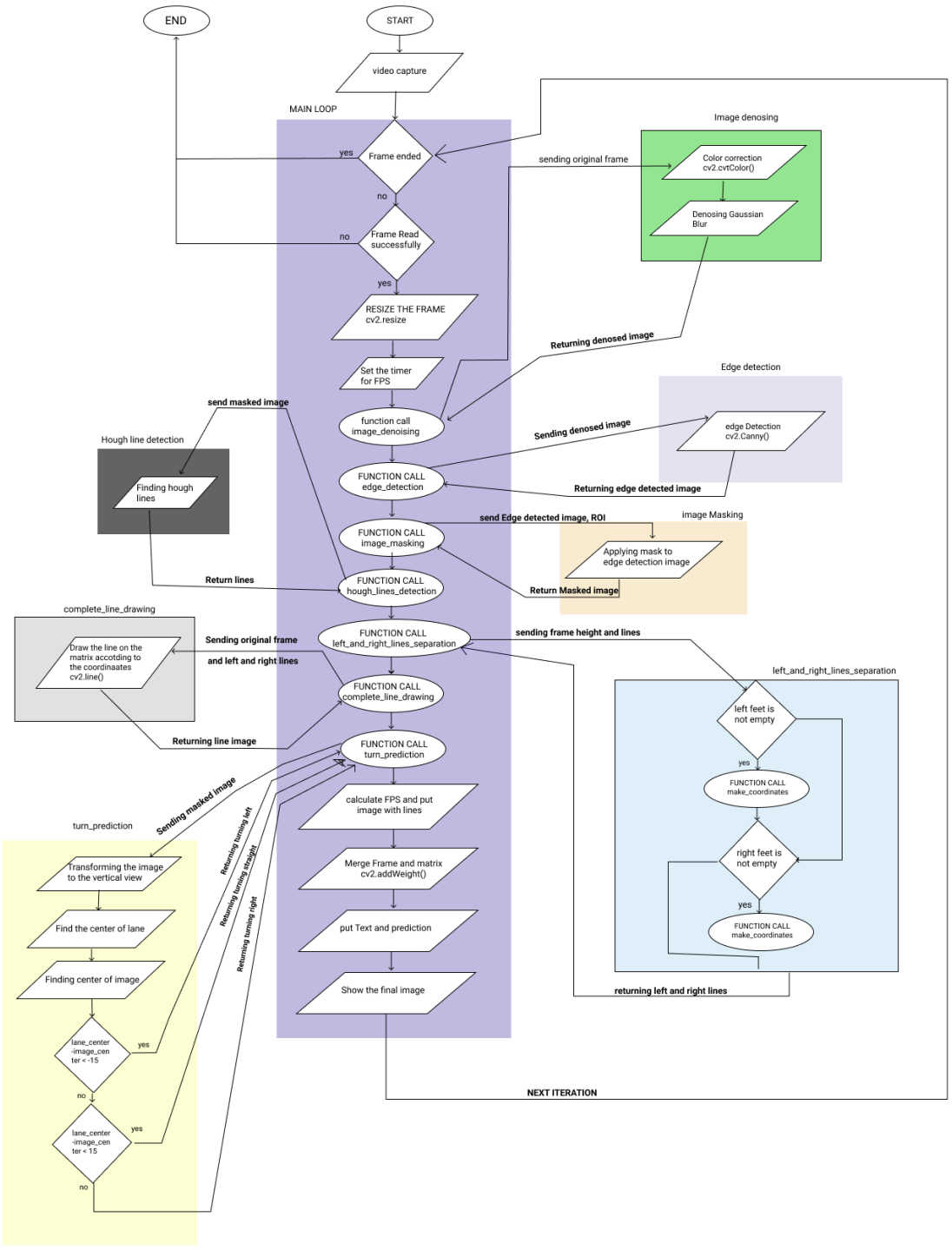
OpenCV: Open Source Computer Vision is a library of programming functions mainly oriented towards machine vision in real time. Comprising more than 2500 advanced algorithms, it contains quite a variety of classical algorithms in Computer Vision that can be used for image analysis, tracking and face recognition, object identification, identifying behavior, traces, and other features.

Numpy: To deal with numerical manipulations. In this library we have used matrices in order to store the value of pixels of given image, in order to find slopes, intercepts

Matplotlib: In order to draw the plot

Code Implementation

In order to make better Visualization of working process of our algorithm we provide this flow chart below.


END
START
video capture
MAIN LOOP
Frame ended
yes
no
Frame Read successfully
no
yes
RESIZE THE FRAME cv2.resize
Set the timer for FPS
function call image_denoising
FUNCTION CALL edge_detection
FUNCTION CALL image_masking
FUNCTION CALL hough_lines_detection
FUNCTION CALL left_and_right_lines_separation
FUNCTION CALL complete_line_drawing
FUNCTION CALL turn_prediction
calculate FPS and put image with lines
Merge Frame and matrix cv2.addWeight()
put Text and prediction
Show the final image
NEXT ITERATION
Image denosing
sending original frame
Color correction cv2.cvtColor()
Denosing Gaussian Blur
Returning denosed image
Edge detection
Sending denosed image
edge Detection cv2.Canny()
Returning edge detected image
send masked image
Hough line detection
Finding hough lines
send Edge detected image, ROI
image Masking
Applying mask to edge detection image
Return Masked image
Return lines
complete_line_drawing
Draw the line on the matrix accotding to the coordinaates cv2.line()
Sending original frame
and left and right lines
Returning line image
sending frame height and lines
left_and_right_lines_separation
left feet is not empty
yes
FUNCTION CALL make_coordinates
right feet is not empty
yes
FUNCTION CALL make_coordinates
returning left and right lines
turn_prediction
Sending masked image
Transforming the image to the vertical view
Find the center of lane
Finding center of image
lane_center -image_cen ter < -15
yes
no
lane_center -image_cen ter < 15
yes
no
Returning turning left
Returning turning straight
Returning turning right

Steps Performed:

Image Noise Removing

First step is to remove noise from given video frame. There are many ways to reduce image noise such as: 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. We implemented Gaussian Blur using OpenCV library with the GaussianBlur() function.

Figure 1 Before and after image using Gaussian Blur (Right image after denoising)

Edge detection

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 blur and grayscale phase

should help to differentiate main lane lines. The outcome will produce a black and white image.

Figure 2 Resultant image after applying Canny edge detection

Selecting Region of interest

To make our code work faster and more efficient we decided to remove upper part of image. To mask our image first created mask image which is filled with zeros. Then masked our input image upper part using Bitwise operator with mask image.

Figure 3 Masked image

Hough line 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. The result of detected line you can see below.

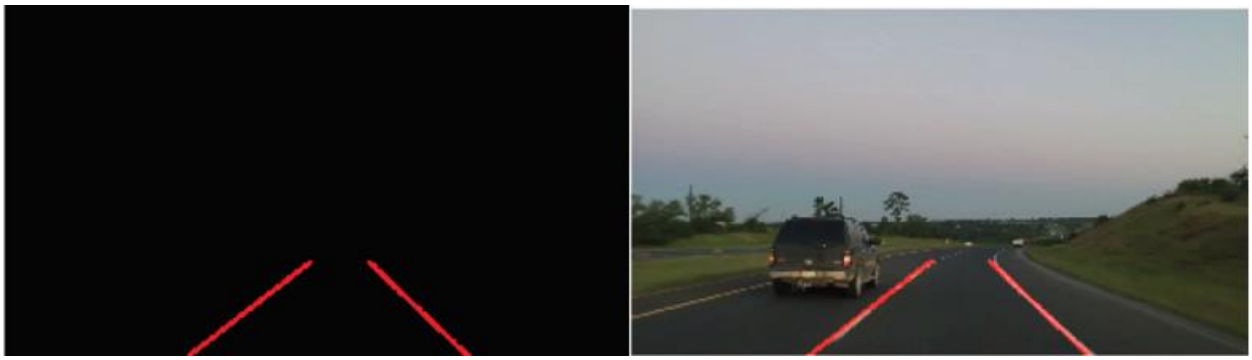


Figure 4 Resultant image after Hough line detection

Line Separation

After detection lines we could separate detected lines using Polar system. 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.

Turn Prediction

Last step is movement prediction.

Figure 5 Result of Turn prediction

Figure 6 Result of Turn prediction

Resolution and FPS

Resolution	FPS range
400x300	410-420
640x480	170-180
800x600	150-160
1024x768	130-140
1280x720	70-80

Figure 10 Resolution 400x300

Figure 9 Resolution 800x600

Figure 8 Resolution 640x480

Figure 7 Resolution 1024x768

Figure 11 Resolution 1280x 720

CONCLUSION AND FUTURE WORK

In this project we could achieve implementing our own real-time Vision based road lane line detection algorithms which works efficient and faster. Also, we could predict motion of moving car, detected lane lines and connected small separate lines using regression.

In future we would like to test our algorithm in autonomous Raspberry car. Also, we wanted to add sign detection features and detecting lines with our own trained model using Tenserflow.

REFERENCES

[1] https://towardsdatascience.com/carnd-project-1-lane-lines-detection-a-complete-pipeline-6b815037d02c#.hcsizymg8

[2] https://www.youtube.com/watch?v=gWK9x5Xs_

[3] https://drive.google.com/file/d/1RMq9j_mxkqPX_C4OYdoJmdfnp15ef0S/view?usp=drivesdk