TOSHKENT SHAHRIDAGI INHA UNIVERSITETI
INHA UNIVERSITY IN TASHKENT

Computer Science and Software Engineering

Comprehensive Assignment: Code report

Capstone Design

TEAM NAME: DEUS EX MACHINA

TEAM NUMBER: 50

TEAM MEMBERS:

1. JAMOLIDDINKHUJA ODILKHUJAEV: U1610092
2. KAMOLA AZIMOVA: U1610101
3. MAHLIYOKHON OLIMJONOVA: U1610133
4. MOKHLAROYIM TUYCHIBOEVA: U1610148

Table of Contents

Table of Figures

Abstract

For vehicles to have the option to drive without anyone else, they have to comprehend their encompassing world like human drivers, so they can explore their way in avenues, delay at stop signs and traffic lights, and abstain from hitting deterrents, for example, different vehicles and walkers. In view of the issues experienced in recognizing objects via self-governing vehicles, an exertion has been made to exhibit path discovery utilizing the OpenCV library. The explanation and method for picking grayscale rather than shading, distinguishing edges in a picture, choosing the area of enthusiasm, applying Hough Transform, and picking polar arranges over Cartesian directions have been talked about.

Introduction

During the driving activity, people utilize their optical vision for vehicle moving. The street path checking, go about as a steady reference for vehicle route. One of the requirements to have in a self-driving vehicle is the improvement of an Automatic Lane Detection system using an algorithm.

Computer vision is an innovation that can empower vehicles to comprehend their environmental factors. It is a part of artificial intelligence that empowers programming to comprehend the substance of picture and video. Present-day Computer vision has progressed significantly due to the progresses in profound realizing, which empowers it to perceive various articles in pictures by analyzing and contrasting a great many models and cleaning the visual examples that characterize each item. While particularly effective for grouping assignments, profound taking in experiences genuine impediments and can flop in unusual manners.

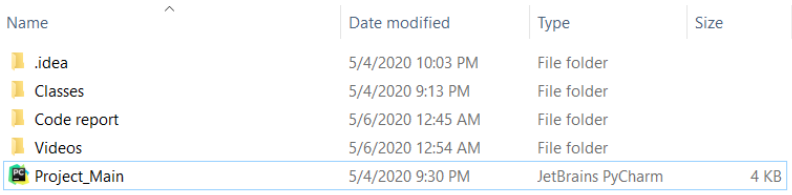
This implies a driverless vehicle may collide with a truck without trying to hide, or more awful, inadvertently hit a person on foot. The current Computer vision innovation utilized in autonomous vehicles is likewise helpless against ill-disposed assaults, by controlling the AI's input channels to compel it to commit errors. For example, specialists have indicated they can deceive a self-driving vehicle to maintain a strategic distance from perceiving stop signs by sticking black and white labels on them.

Project structure

This section will give an overview about the hierarchy of the project.

MAIN DIRECTORY:

We will begin to describe our structure with Main directory which consists of Videos, Classes subdirectories and main code file.

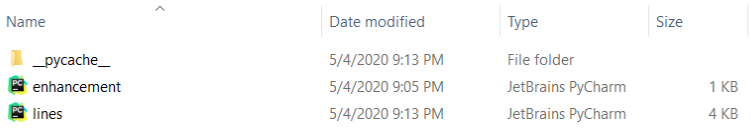


Name	Date modified	Type	Size
.idea	5/4/2020 10:03 PM	File folder	
Classes	5/4/2020 9:13 PM	File folder	
Code report	5/6/2020 12:45 AM	File folder	
Videos	5/6/2020 12:54 AM	File folder	
Project_Main	5/4/2020 9:30 PM	JetBrains PyCharm	4 KB

Figure 1: Main Folder

CLASSES DIRECTORY:

In classes directory you can find enhancement.py and lines.py files



Name	Date modified	Type	Size
__pycache__	5/4/2020 9:13 PM	File folder	
enhancement	5/4/2020 9:05 PM	JetBrains PyCharm	1 KB
lines	5/4/2020 9:13 PM	JetBrains PyCharm	4 KB

Figure 2: Classes Folder

Enhancement.py file contains Enhancement class with functions to convert image to a grayscale, denoise, do canny operation and segmenting mask, which will be discussed in subsequent chapters.

The other file in the Classes directory is a Lines.py, which as a name says it, works with lines to calculate and show the lane lines. Here you can find functions such as apply_houghlines, calculate coordinates of lines and calculate and visualize lines.

VIDEOS DIRECTORY:

Going back to one folder, we can find Videos. However, you might not see any videos because of the limitations of our e-learning system. Actually, it should be seen output Videos with: 1280x720, 1024x768, 800x600, 640x480, 400x300 resolution.

Figure 3: Videos Folder

Having finished with the folders structure we can move on to actual project.

Methodology

Library

The project involves detection of lane lines in an image using Python and OpenCV.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which incorporates a far-reaching set of both exemplary and best in class PC vision and AI calculations. These calculations can be utilized to distinguish and perceive faces, recognize objects, characterize human activities in recordings, track camera developments, track moving items, extricate 3D models of articles, produce 3D point mists from sound system cameras, fasten pictures together to create a high goals picture of a whole scene, find comparative pictures from a picture database, expel red eyes from pictures taken utilizing streak, follow eye developments, perceive landscape and build up markers to overlay it with enlarged reality, and so forth. OpenCV has in excess of 47 thousand individuals of client network and assessed number of downloads surpassing 18 million. The library is utilized broadly in organizations, inquire about gatherings and by administrative bodies. (OpenCV, n.d.). (OpenCV, n.d.)

INITIAL PREPARATIONS

In order to convert image to grayscale we first need to import necessary libraries, which will make available the functionalities needed to read the original image and to convert it to gray scale.

```
import cv2 as cv
import numpy as np
import time
from Classes.enhancement import Enhancement as en
from Classes.lines import Lines
```

Figure 4: Preparations

In order to read the original video, we can simply call ***VideoCapture*** function of the cv2 library, then make some error checks for a robust code.

```
if __name__ == "__main__":

    # Initialize the video name, use default if you are not changing the name.
    vName = "Videos/Texas2.mp4"
    # The video is read in as a VideoCapture object
    cap = cv.VideoCapture(vName)
```

Figure 5: VideoCapture

However, we need to apply our functions to each frame of the video, to achieve that we can loop though video until it will stay open and make error check.

```
while cap.isOpened():
    # param: ret a boolean value from the frame
    # param: frame gives current frame
    ret, frame = cap.read()
    if not ret:
        break
```

Figure 6: Frame read

Additional Note is that channels are going to be stored in **BGR** (Blue, Green and Red) order by default.

As we successfully read the video, we need to convert it to gray scale.

Grayscale Conversion

To do it, we have to call the ***cvtColor*** function, which gives the opportunity to convert the frame from a color space to another.

```
# First mission convert to gray
gray = en.convert_gray(frame)
```

To have easier way to navigate though code, functions were divided into several classes, which were discussed *Project Structure* part of the document. Therefore, we are calling ***convert_gray*** function from ***Enhancement*** class.

```
def convert_gray(self):
    # Convert to grayscale, because one channel is enough to compute edges
    return cv.cvtColor(self, cv.COLOR_BGR2GRAY)
```

As first input, this function receives the current frame. As second input, it takes the color space conversion code. Since we need to convert our frame from the BGR to gray color space we need to use the **COLOR_BGR2GRAY** code.

Here is the result of the code compilation:

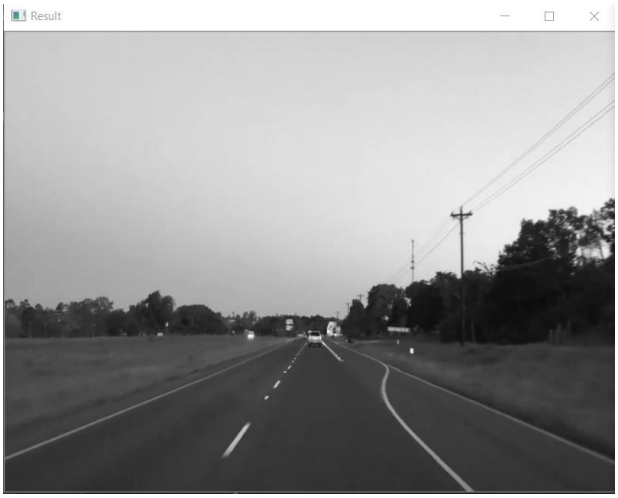


Figure 7: Grayscale Image

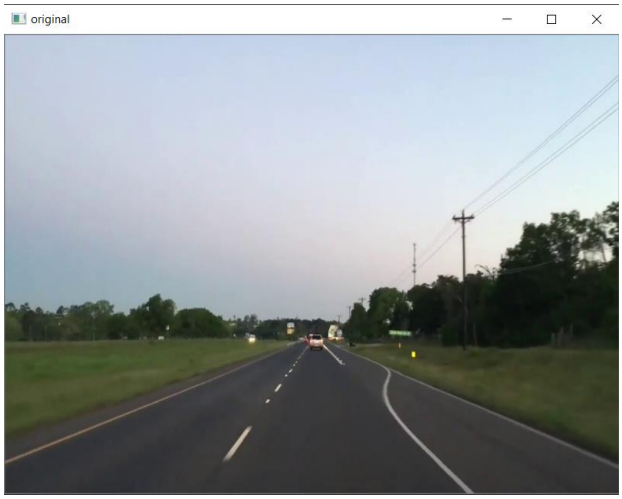


Figure 8: Original Image

Blurring the frame

Each of the pixels for a grayscale image is described by a single number that describes the brightness of the pixel. In order to smoothen an image, the typical answer would be to modify the value of a pixel with the average value of the pixel intensities around it. Averaging out the pixels to reduce the noise will be done by a kernel. This kernel of normally distributed numbers(np.array([[1,2,3],[4,5,6],[7,8,9]]) is run across our entire image and sets each pixel value equal to the weighted average of its neighboring pixels, thus smoothening our image. In our case we will apply a 5x5 **Gaussian kernel**:

```
blur = en.deNoise(gray)
```

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image. So, edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges).3

```
def deNoise(self):
    # Apply 5x5 Gaussian blur
    return cv.GaussianBlur(self, (5, 5), 0)
```

We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.

Here is the result of the blurred image:

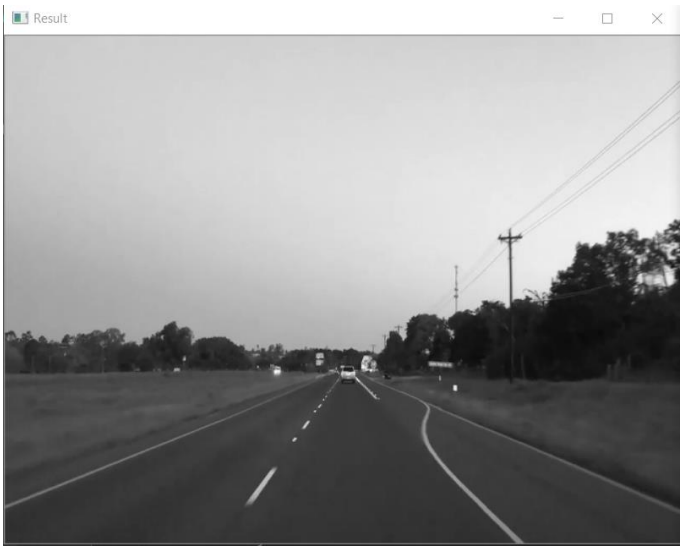


Figure 9: Grayscale Image

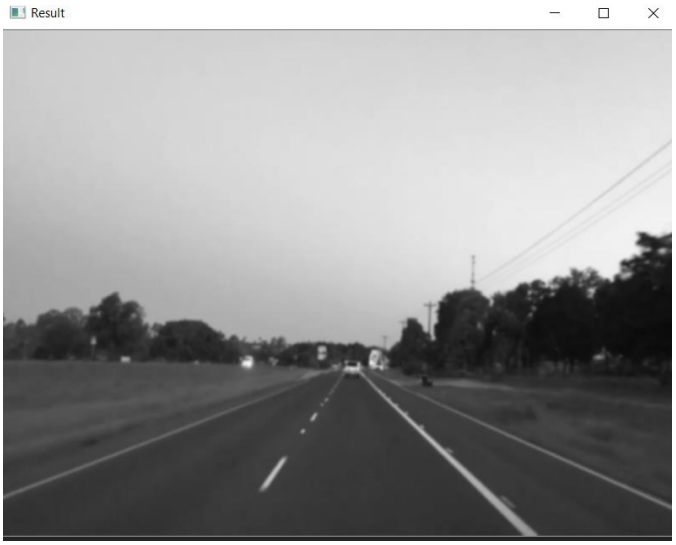


Figure 10: Blurred Image

Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. it is a multi-stage algorithm for edge detection. The various stages of Canny Edge Detection algorithm are:

1. Noise Reduction
2. Finding Intensity Gradient of Image
   Intensity Gradient
3. Non-Maximum Suppression
4. Hysteresis Thresholding

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

As input, it will take blurred image, low_threshold and high threshold. They allow us to isolate the adjacent pixels that follow the strongest gradient. If the gradient is larger than the upper threshold then it is accepted as an edge pixel, if it's below the low threshold then it is rejected. If the gradient is between the thresholds then it is accepted only if it's connected to a strong edge. Areas where it's completely black correspond to low changes in intensity between adjacent pixels whereas the white line represents a region in the image where there is a high change in intensity exceeding the threshold.

```
# do Canny
canny = en.do_canny(blur)
```

```
def do_canny(self):
    # Canny edge detector with threshold values 50 and 150
    return cv.Canny(self, 50, 150)
```

Third argument is *aperture_size*. It is the size of Sobel kernel used for find image gradients. By default, it is 3. Last argument is *L2gradient* which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function, which is by default False:

$$Edge\_Gradient\ (G) = |G_x| + |G_y|$$

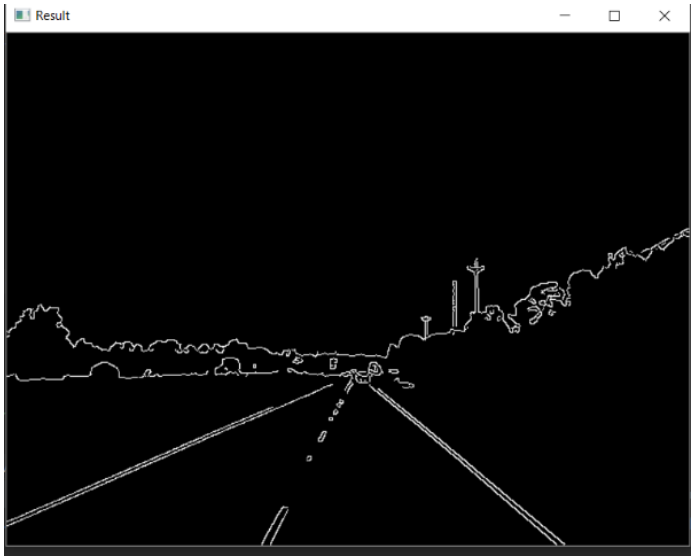


Figure 11: Canny Edge Detection

Region of Interest

The dimensions of the image are chosen which will contain the road lanes and mark it as our region of interest or the kernel. Then a mask is created which is same as the dimension of the image which would essentially be an array of all zeros. We know an image is an array of pixels, *zeros_like* creates an array of zeros with the same shape as the image. Meaning that both mask and row are going to have the same number of rows and columns. Although the mask is going to be completely black because it only contains zeros. Now we fill the triangle dimension in this mask with the intensity of 255 so that our region of interest dimensions is white. We use OpenCV's *fillPoly* function in order to fill the polygon on the mask. Now we will do a bitwise AND operation with the canny image and the mask which will result in our final region of interest.

```
# Segment the mask
mask = en.segment_mask(canny, kernel)
```

```
def segment_mask(self, kernel):
    # Image filled with zero
    mask = np.zeros_like(self)
    # fill Mask with ones other areas with 0
    cv.fillPoly(mask, kernel, 255)

    return cv.bitwise_and(self, mask)
```

Here is the result after applying masking on the frame:

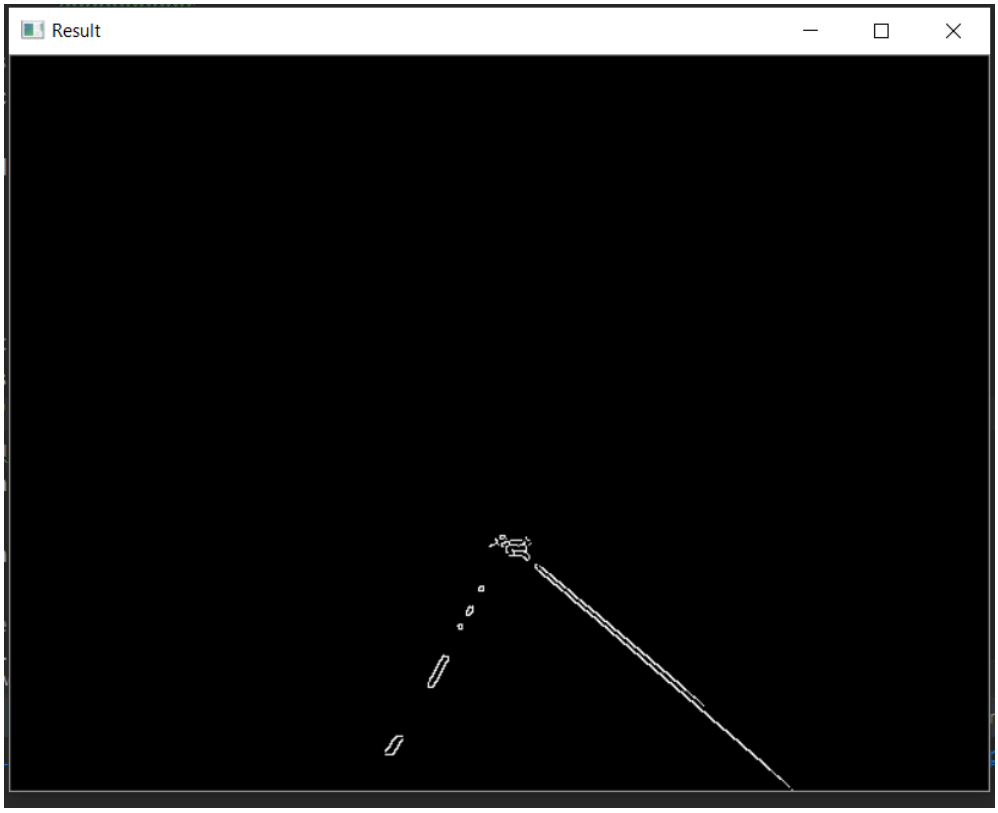


Figure 12: Masked image

Hough Transform

Finally, we need to make a Hough transform.

```
# Apply hough lines
hough = Lines.apply_houghlines(mask)
# Averages all lines from hough into one line
lines = Lines.calculate_lines(frame, hough)

# Visualizes the lines
visualize = Lines.visualize_lines(frame, lines)

# Overlays lines on frame by taking their weighted sums and adding scalar value of 1 as the
result = cv.addWeighted(frame, 0.9, visualize, 1, 1)
```

Hough lines takes frame, distance resolution of accumulator (larger => less precision),
Angle resolution (larger => less precision),
Threshold of minimum number of intersections,
Min and Max distance between disconnected lines

```
def apply_houghlines(self):
    # Hough lines takes frame, distance resolution of accumulator(larger => less precision),
    # Angle resolution (larger => less precision),
    # Threshold of minimum number of intersection,
    # Min and Max distance between disconnected lines
    hough = cv.HoughLinesP(self, 2, np.pi / 100, 100, np.array([]), minLineLength = 100, maxLineGap = 150)
    return hough
```

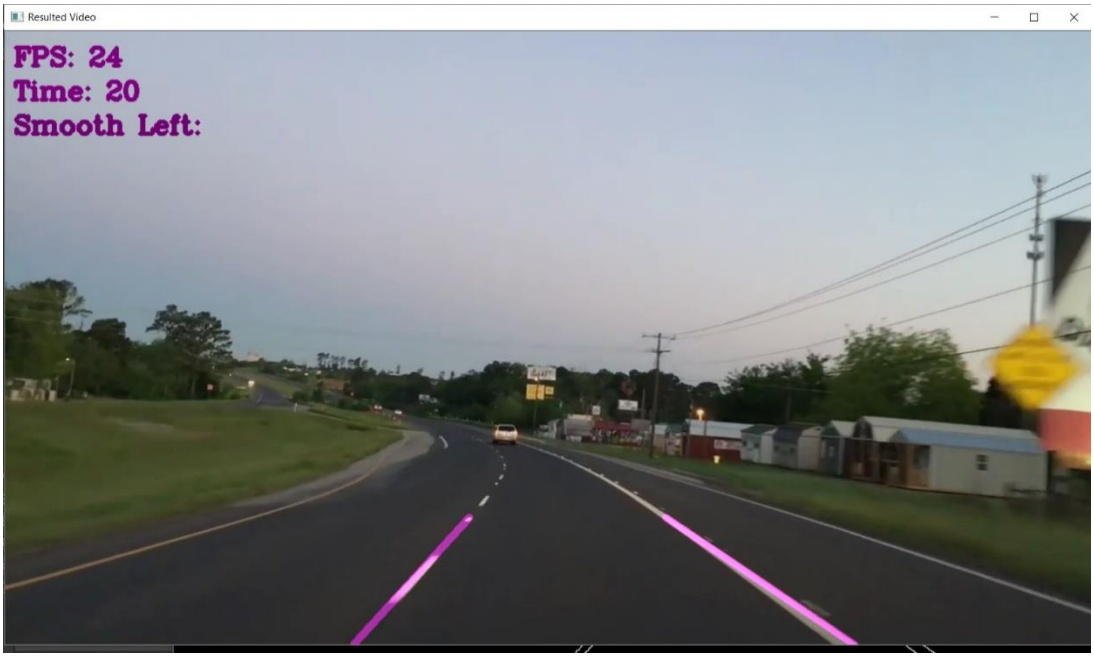
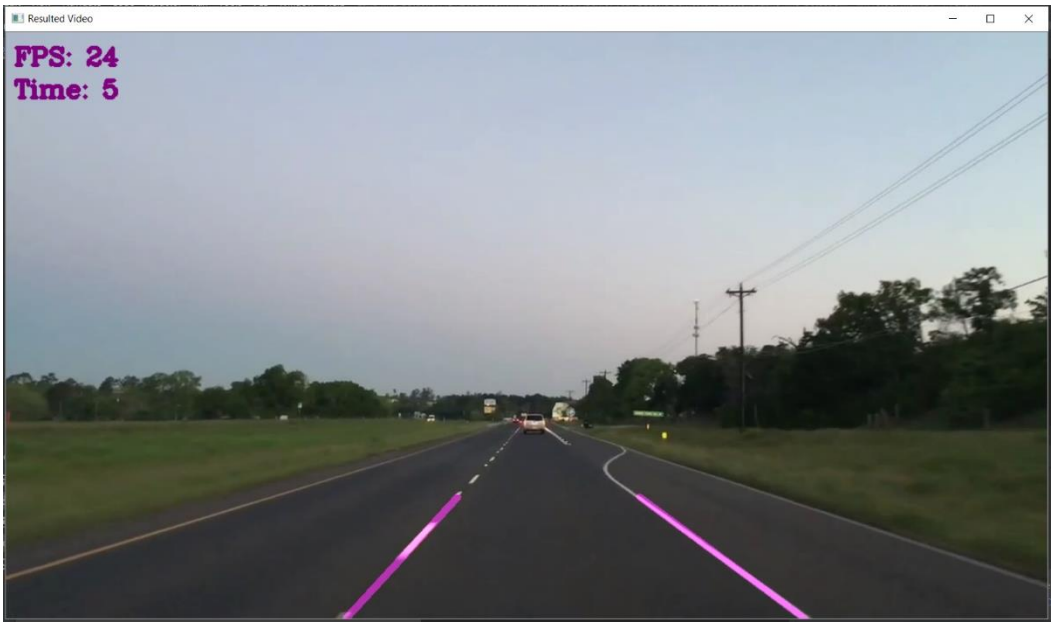


Figure 13: Result image

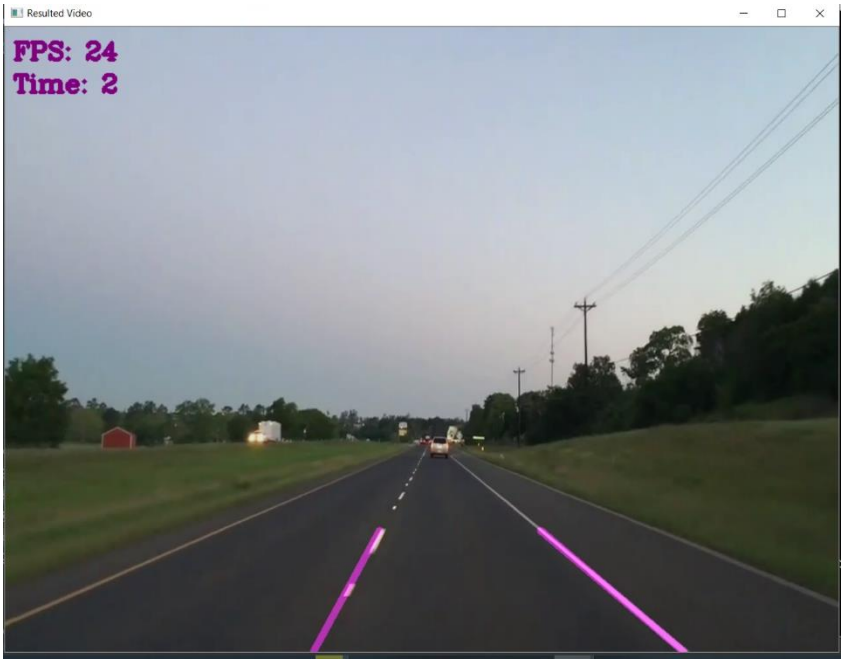
Comparison of different video sizes.

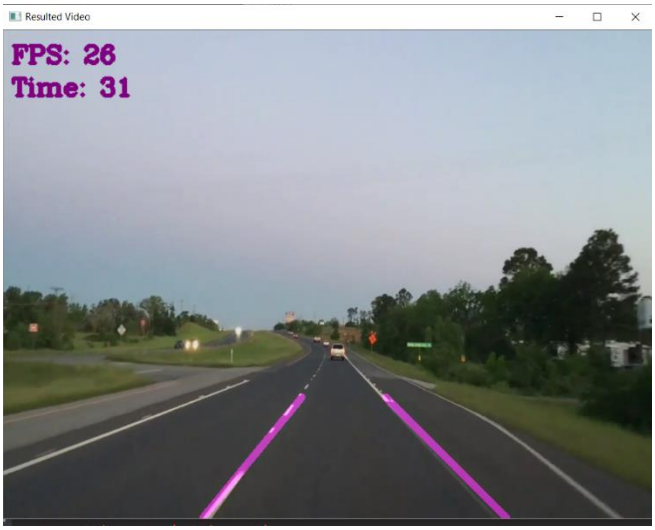


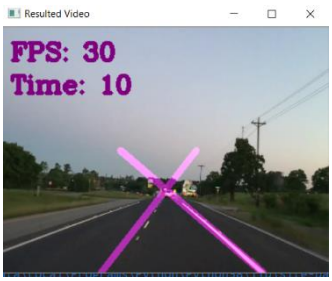
As it can be seen from images, the FPS is increasing for smaller resolutions, so the

1. 1280x720: 24
2. 1080x768: 24
3. 800 x 600: 26
4. 640 x 480: 27
5. 400 x 300: 30

Seeing the results, we can conclude that decreasing the resolution size, will increase the FPS rate







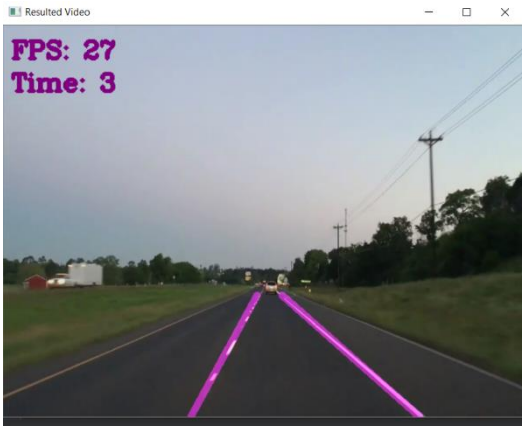


Figure 14: Different frame size images

Conclusion

In the project structure part, we have seen the directories and subdirectories of the project.

In the methodology part, we have used the OpenCV library and its functions such as the Canny Function through which we achieved edge detection.

After that, we prepared a mask of zero intensity and mapped our region of interest by performing the bitwise operation.

Then we used the Hough Transform technique that detected the straight lines in the image and identified the lane lines. We made use of the polar coordinates since the Cartesian coordinates don't give us an appropriate slope of vertical and horizontal lines.

Finally, we combined the lane image with our zero-intensity image to show lane lines.

Additionally, we have compared FPS (frame per second) of different sizes of video (1024x768, 800x600, 640x480, 400x300 and 1280x720)

References

OpenCV. (n.d.). *https://opencv.org/about/*.

-
- https://www.researchgate.net/publication/323406613_VEHICLE_NAVIGATION_USING_ADVANCED_OPEN_SOURCE_COMPUTER_VISION
- Jay Hoon Jung, Yousun Shin, YoungMin Kwon (2019). "Extension of Convolutional Neural Network with General Image Processing Kernels".
- Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2016). "You Only Look Once: Unified, Real-Time Object Detection".
- Mayank Singh Chauhan, Arshdeep Singh, Mansi Khemka, Arneish Prateek, Rijurekha Sen (2019). "Embedded CNN based vehicle classification and counting in non-laned road traffic".
- Tejas Mahale, Chaoran Chen, Wenhui Zhang (2018). "End to End Video Segmentation for Driving: Lane Detection for Autonomous Car".