**BIOMETRICS**

**CSCI 6970 (CRN 25886)**

**Spring 2019**

Final semester project (Option-C (Python):)

**Joseph Kwame Apprey-Hermann**

**Lane detection with open-CV using python**

OpenCV (Open Source Computer Vision Library) is an open source software library for computer vision and machine learning. The library was built originally by Intel with the support of Willow Garage to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

The library has over 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

OpenCV is used extensively in companies, research groups and by governmental bodies. With big names like google, yahoo, Microsoft and the likes.

OpenCV is written natively in C/C++ but interfaces for C++, Python, Java and MATLAB it is also compatible with Windows, Linux, [Android](https://opencv.org/android/) and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured [CUDA](https://opencv.org/cuda/)and [OpenCL](https://opencv.org/opencl/) interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

Machine learning, deep learning and artificial intelligence play very vital role in modern computing. As learners of this field and concept of computing it became very necessary to have a practical experience of how these concepts work in real life.

Using OpenCV for lane detection in this project provided three great opportunities, the first one is the opportunity to use and experience OpenCV. A situation that has arouse so much interest and has provided a great inspiration to look and dive further into.

This project has provided a great opportunity to apply machine learning in a practical way. And has enhanced my understanding of the machine learning concepts. The final major opportunity to create a software that can detect lanes is very fulfilling, as this system is used in modern self driving automobile.

It was a great challenge with some compatibility issues from the installing of OpenCV and other API’s to the writing of the code but at the end it was a very interesting one that I will always be proud of.

**System Requirements:**

Software (APIs)

* Opencv: Open cv version 4.0 was used for the project even though at the the latest version 4.1 has been realesed, it had some compatibility issues with computer.
* Anaconda version 3.6
* Text editor: sublime text the unregistered version.

Hardware

* Window 10, 64 bit with 4GB RAM (intel core) i3 cpu 2.21Ghz.

Development/Implementation

The “feature detection module” was adopted and used for this project to detect lane in an image.

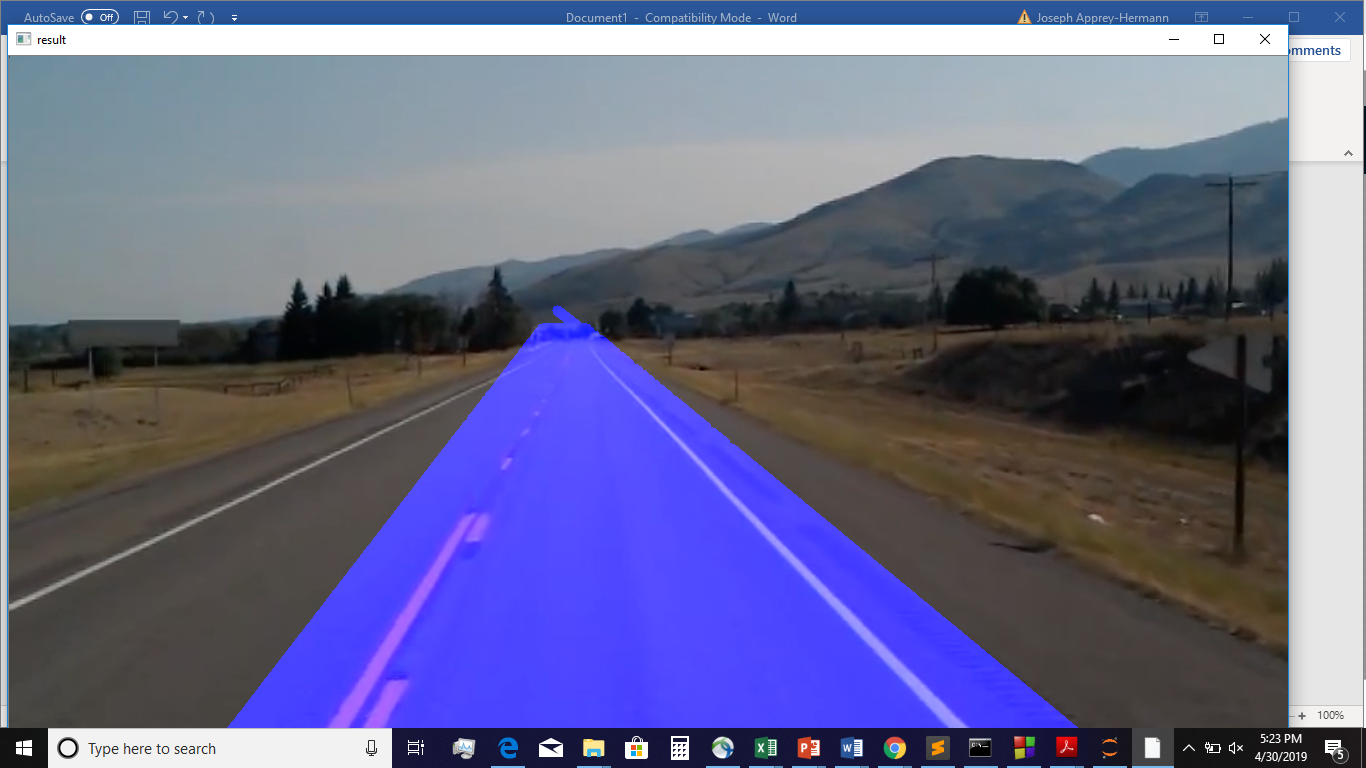
A total of about 50 lines of codes are used in this project. This might be a plus for OpenCV it took a great deal of time and work to get those codes together. Few codes but took several weeks to get the project to it current state. Most of the methods and functions are new and I had to get around them a couple of time to attain the right output, more over there was the need to take some time out to understand how most of the methods work and that took most of the time as I had to do other course work alongside.

*The image below was downloaded and used for this project.*



*Image used for the lane detection project. A jpg image.*

*After running the program the following is displayed*



The program was tested on other similar images. During the test it was realized that, images that are different sized from the image used for the initial for the writing of the programs couldn’t be detected.

Future enhancement.

I am still working on this program in attempt to get it to detect lane in a video feed or from the live camera.

The source code together with the test image are presented in a zipped file.

BY

JOSEPH KWAME APPREY-HERMANN

Reference:

<https://opencv.org/about/>

<https://en.wikipedia.org/wiki/OpenCV>

source codes:

*import cv2*

*import numpy as np*

*import matplotlib.pyplot as plt*

*def canny(image):*

*gray = cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY) #to change the image from colored to to gray for easy separation of the lane features.*

*blur = cv2.GaussianBlur(gray,(5, 5), 0)*

*canny = cv2.Canny(blur, 50, 150)*

*return canny*

*#function to show the lanes*

*def display\_lines(image, lines):*

*line\_image = np.zeros\_like(image)*

*if lines is not None:*

*for line in lines:*

*x1, y1, x2, y2 = line.reshape(4)*

*cv2.line(line\_image, (x1, y1), (x2, y2), (225, 0, 0), 10)*

*return line\_image*

*# function to demarcate the side of the right side of the road/ the lane of interest and cover it up with a mask for easy identification.*

*def region\_of\_interest(image):*

*height =image.shape[0]*

*polygons = np.array([*

*[(200, height), (1100, height), (550, 250)]*

*])*

*mask = np.zeros\_like(image)*

*cv2.fillPoly(mask, polygons, 255)*

*masked\_image = cv2.bitwise\_and(image, mask)*

*return mask*

*image = cv2.imread('test\_image.jpg')*

*lane\_image = np.copy(image)*

*canny = canny(lane\_image)*

*cropped\_image = region\_of\_interest(canny)*

*lines = cv2.HoughLinesP(cropped\_image, 2, np.pi/180, 100, np.array([]), minLineLength=40, maxLineGap=5)*

*line\_image = display\_lines(lane\_image, lines)*

*combo\_image = cv2.addWeighted(lane\_image, 0.8, line\_image, 1, 1)*

*cv2.imshow("result", combo\_image)*

*cv2.waitKey(0) # to let the image staty until a key is pressed*