Self Driving Car Nano Degree

Project -1 Lane Line Detection Report

Jaganadh Gopinadhan

March 24, 2017

jaganadhg@gmail.com

Introduction

Abstract

Autonomous vehicles are are going to play an integral role in human life in near future. Rapid developments in technology and applied sciences accelerated development of autonomous systems. It is somewhat like yesterday's science fictions came to reality today. Computer Science, hardware technology such as advanced sensors, Machine Learning and Mathematics, Physics and Artificial Intelligence is rightly baked to build such solutions. The path towards building an autonomous vehicle is challenging. Step by step research and development is required to achieve the goal. One of the basic requirement in this area is to find Lane Lines in the road for safe driving. It is a quite exciting problem to solve. There are standard Open Source Softwares to help us in developing a basic system. But there are unique challenges in this and it has to be resolved by a systematic investigation. The current project aims to develop a lane line detection system in normal conditions with minimal complexity in the images.

Introduction

Objective and Data

The objective of this project is to develop a basic lane line detection system. A set of six sample images is provided to build and test the methodology. The following technology environment is used for developing the system.

- Programming Language - Python 2.7
- Image and Video Processing - OpenCV- Python 3.0
- Statistical Computing - Python - Numpy
- Video IO - MoviePy

The image samples covered following patterns of line on the road.

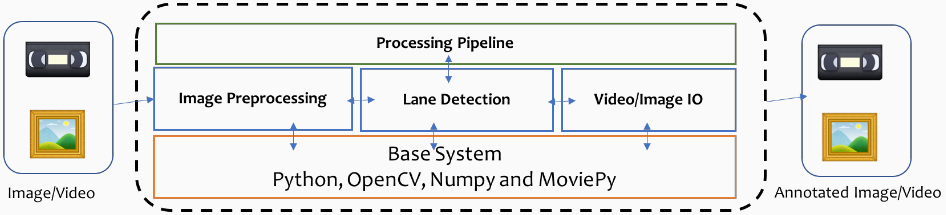
- Curve with solid white/yellow lane mark
- Left/Right solid yellow/white line mark and dotted line

A set of three videos are also supplied to test the system. The video covers above mentioned phenomena and a challenging one. The challenging one has curve and shades of trees in the image.

Approach

Approach

The overall solution view is described in the diagram below.



The logical steps involved in the solution includes:

- Image Preprocessing
- Lane Detection
- Video/Image IO utility
- Pipeline

Approach - Preprocessing

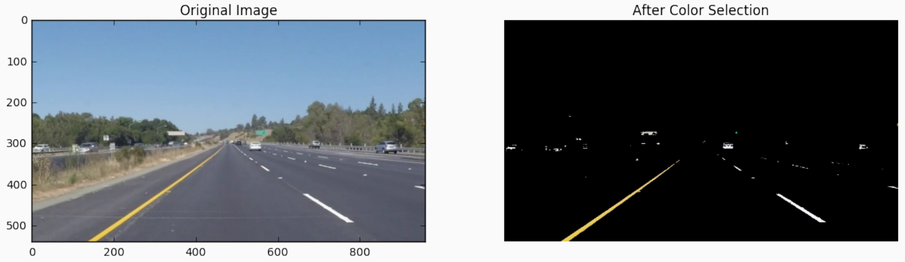
The objective of preprocessing is to detect edge lines in the image. The preprocessing of an image requires following steps.

- Color selection
- Image Enhancements
- Parameter Estimation for Edge Detection
- Edge Detection
- Region of Interest Cropping
- Edge Detection

Approach - Preprocessing

Color Selection

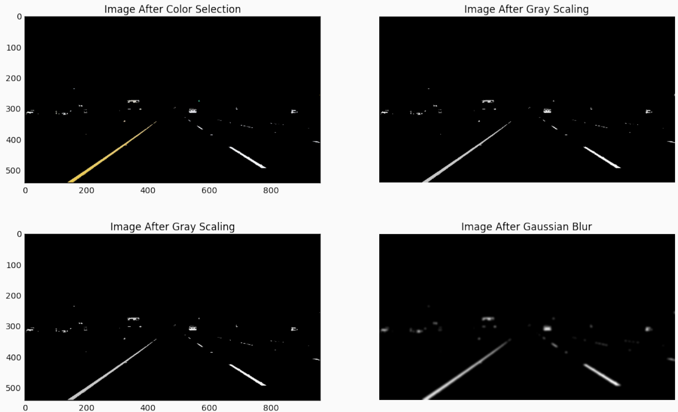
The lane lines in road images/video frames is either yellow or white. In a single image, both yellow and white will be available. So we have to extract everything in white and yellow in the image. The OpenCV inRange API with yellow and white color minimum pixel threshold is used to filter the color. Each color pixels are extracted separately extracted and combined to get the color selected image. The result of color selection is shown below.



Approach - Preprocessing

Image Enhancement

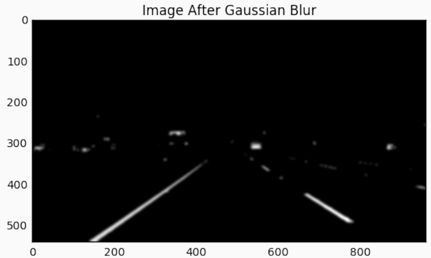
For enhancing the image before it goes to canny edge detection we apply a gray scaling and Gaussian Noise addition. Results of the gray scaling and Gaussian Noise addition is shown below.

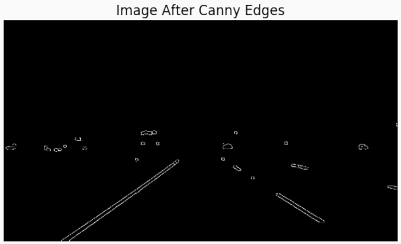


Approach - Preprocessing

Edge Detection

In image processing edge detection is the process of finding boundaries of objects with-in an image. The Canny Edge algorithm is one of the most popular edge detection methods. The canny method requires two parameters low_threshod and high_threshold. We can either give a trial and error or best guess approach. But this can fail sometimes. So it is better to find it by another algorithm. We used the Otsu Threshold methodology estimate the high threshold and multiplied the value bu 0.5 to obtain the low threshold. This is a popular trick in image processing to get the threshold for the canny edge.

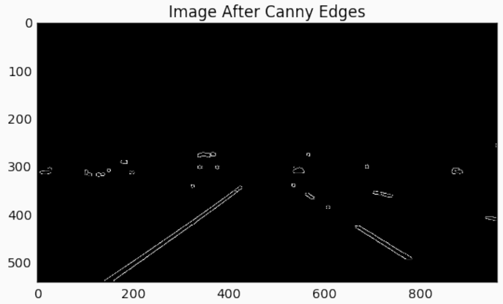


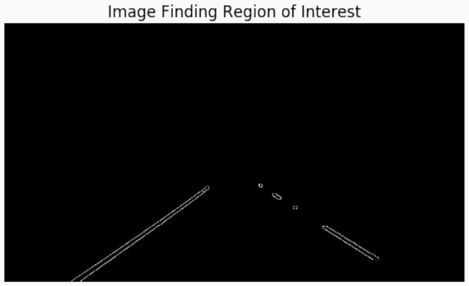


Approach - Preprocessing

Region of Interest Cropping

The edge detected results have both lane line and other edges marked there. It is better to remove those elements before we estimate a smooth lane line in the image. From the bottom of the image to a visible horizon in the image has to be cropped to avoid other edges like makes of cars in the road, or traffic signs or bridge lanes etc..

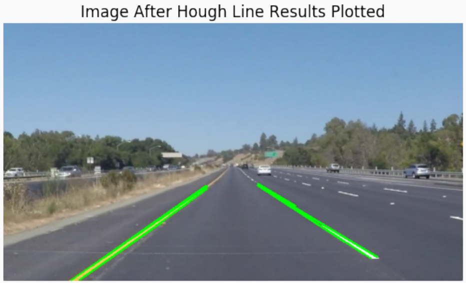




Approach - Line Detection

Hough Transformation

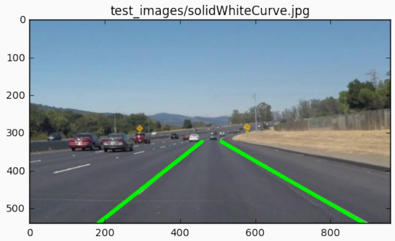
To identify potential lane lines from the image, now we will apply a new technique called Hough Transformation. This techniques is used to isolate feature of a particular shape within an image. It is useful to detect straight lines.

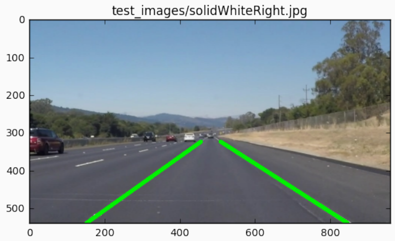


Approach - Line Detection

Hough Transformation - Post Processing

The analysis of hough transformation reveals that it detect lines. But there is some noise such as multiple adjacent straight lines are detected. It is due to edges in the thick line marking. The dotted lane line which is a common phenomenon will not be well captured in the output. It will be still marked as dotted lines in the hough result some times. To avoid that we have to use a simple line fitting technique such a ”y = mx + b” .
This implementation is adopted from another note in Github. Reference provided in the code.





Approach - Video and Image IO

Video IO
The video IO process is handled using the MoviePy library in python. The video is opened by the library and a function is passed to the object to apply transformation in each frame and reconstruct the video.

Image IO is handled using matplotlib image reading, which has to be later migrated to OpenCV or Nmpy functions.

Approach - Pipeline

Pipeline

A pipeline in data processing system is used to perform a series of sequential operations starting from preprocessing to postprocessing. To make the concept completely functional we have developed a pipeline. The pipeline API has two methods one to handle single image/frame and the other one to handle video input. The video input expects a vide file name as inPut and the image API expects image as numpy array.

The results of video processing is available at:

`https://www.youtube.com/watch?v=Lw47Zfltjro` solidWhiteRight

`https://www.youtube.com/watch?v=M5zzMs5xLF8` solidYellowLeft

`https://www.youtube.com/watch?v=aL_uIito0_4` Challenge

Discussion

Special Recipe and Future

Special Recipe

In this project, we tried to implement the automatic threshold estimation for Canny Edge detection. The two methods tested is Otsu Threshold based and median pixel based one. In the final pipeline, we used the Otsu threshold-based approach.

Future Scope

Estimating the kernel size for Gaussian Blur and parameter estimations for hough transform will be a potential enhancement for the system.

Trying Random sample consensus (RANSAC) for final line estimation is a good experiment. This will add more robustness as we are trying to find a straight line most of the time. At the same time, typical regression techniques can be tested to finalize the methodology.

Limitations

The current system provides a very basic lane line finding system. It operates on very normal camera conditions. It has to be tested on extreme weather condition images, the road with shaded or urban roads, roads with bumper to bumper traffic and parked vehicles.

In the video output, a jitter is visible in the result when a curve is approaching. It is a dangerous technical glitch when we depend on this line for determining steering angle. Frame to frame transition with previous frame context and averaging has to be performed to resolve the same. Else the system will be subject to DUI ;-(. The notable key limitations and improvement area include:

- Tune for adjusting with extreme road conditions
- Avoiding jitter in the video frames
- Automatic threshold estimation for Hough Transformation
- Automatic kernel size estimation for Gaussian Blur

Honer Code Declaration

Declaration

For the development of this project various open source libraries and its documentation and forum discussion are referred along with the Udacity lecture materials. The OpenCV Python documentation is the primary reference point. For the smooth line drawing the code has been adopted from a GitHub repository which is duly acknowledged in the code.

The opinion or point of view expressed in this document are explicitly associated with me and have no relation to my employer.

Thank You