Self Driving Car - Finding Lanes using OpenCV

Udacity

Self Driving Car - Nanodegree

Robin Redhu

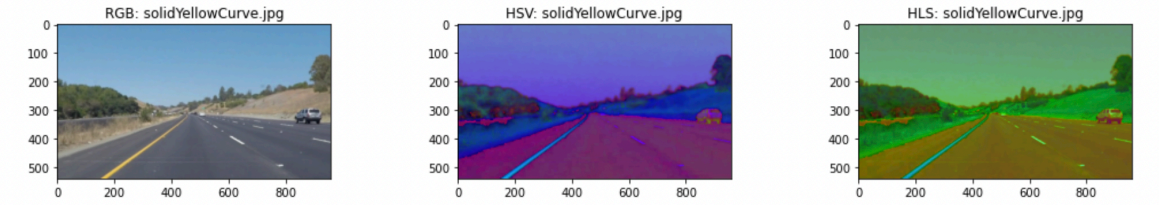
13 June 2021

We humans drive our vehicle with our vision and brain, and so do autonomous vehicles, vision obtained from Camera / Radar / Lidar and intelligence from machine learning. This article will focus on creating a simple pipeline to detect the lanes marked on roads which will help the vehicle to determine its position on road using python and OpenCV (computer vision) . At first, we will go through the pipeline steps and then we will talk about the limitations and some suggestions to improve the below algorithm.

Lane Detection Pipeline -

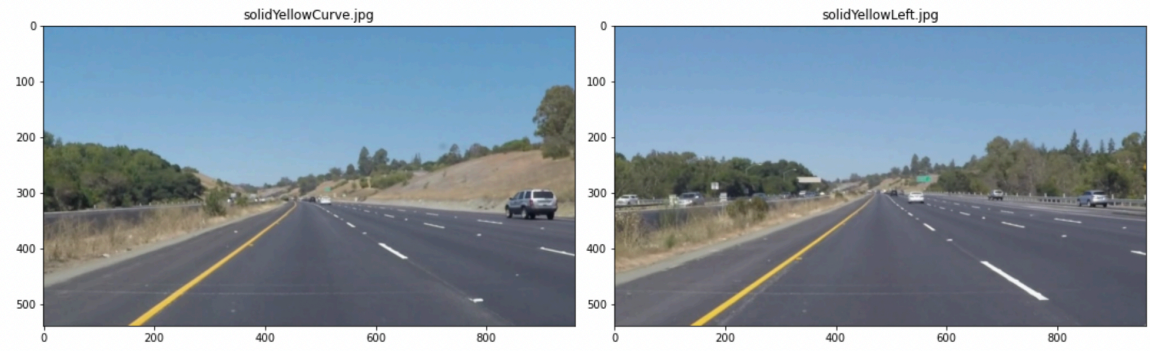
1. Selecting Color space (RBG/HSV/HSL)

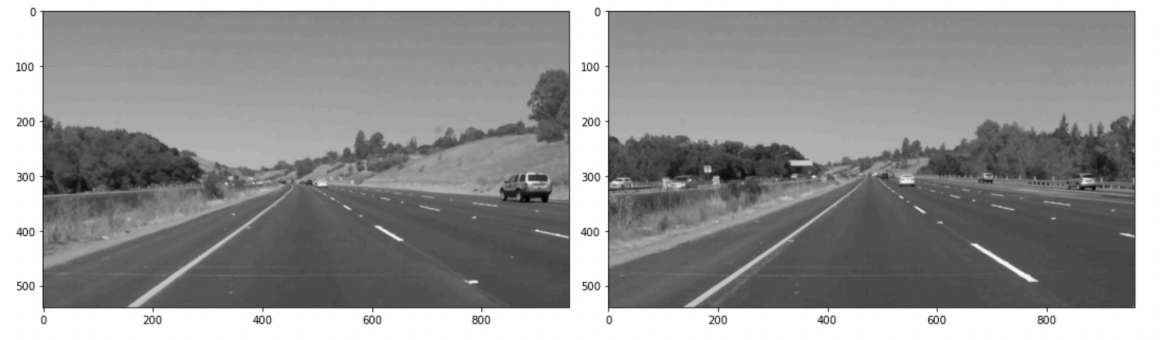
We have variety of color spaces and in this project I have tested with RGB, HSV and HLS color models. Out of which we can clearly extract the lanes from image using HLS color space.



2. Read the image and convert it to grayscale.

This is necessary because it will make things easier, since it is hard to extract the objects from multi dimensional color space and in grayscale we only have 1 channel, it will be easy to detect edges.





3. Adjusting Gamma value to darken the image to clearly distinguish between the lanes and other objects.

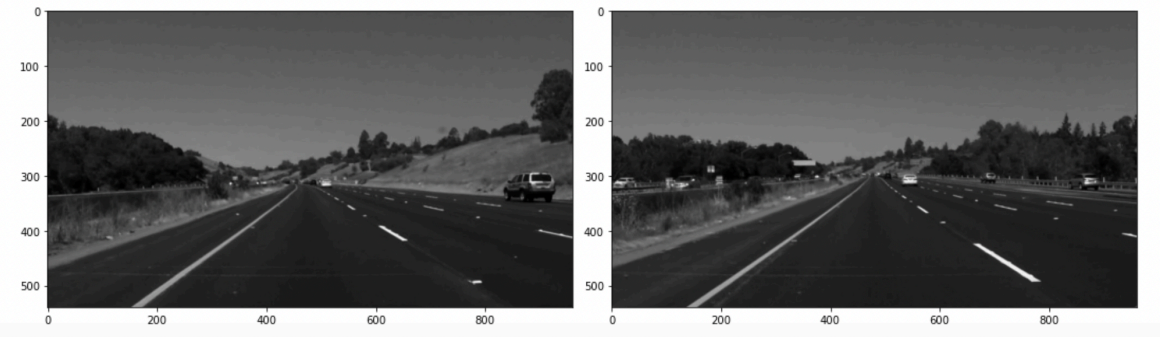
Gamma correction is also known as Power Law Transform. First our image pixel intensities are scaled from [0, 255] —> [0, 1]. Then we apply the gamma function and correct the gamma output value and finally again the intensity pixels are scaled back to [0, 255] range.

Note that:

If gamma values are < 1, then image is shifted towards darker spectrum

If gamma values are > 1, then image is shifted towards brighter spectrum

And, if gamma values are = 1, then no change in image.



4. Isolate lanes from image(Creating masks for yellow or white lanes).

Here we have to select the threshold range of the colours which we want to extract from image. For example in our case we have to extract yellow and white lanes from the road. For that, my thresholds were :

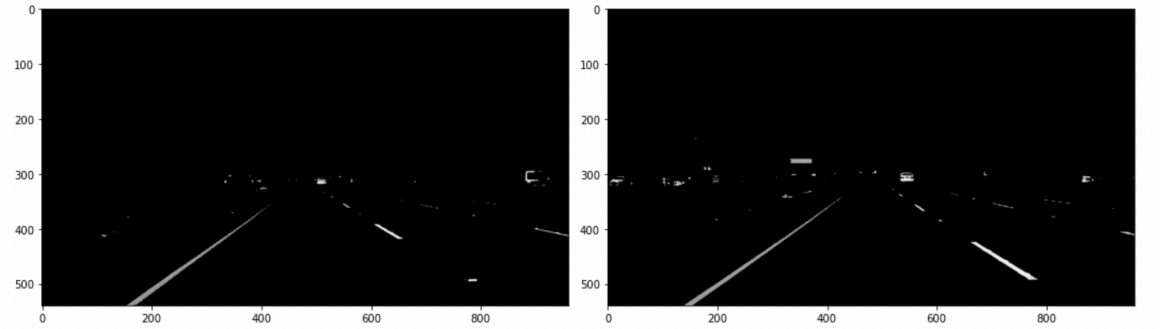
white_low = [210, 210, 210]

white_high = [255, 255, 255]

yellow_low = [190, 190, 0]

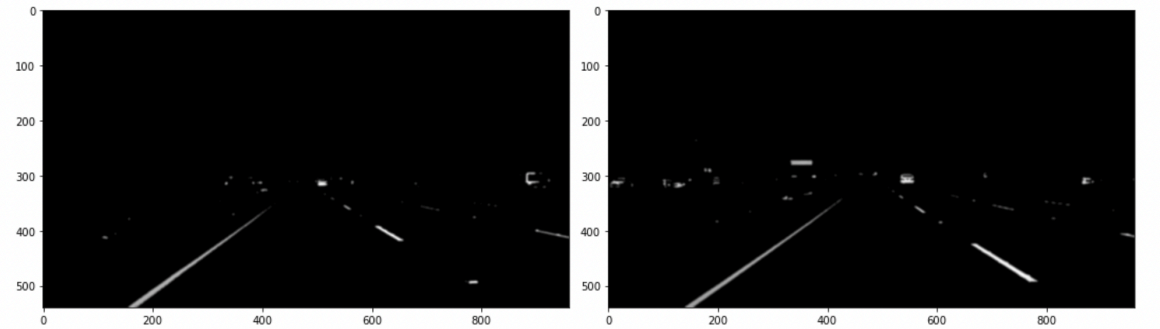
yellow_high = [255, 255, 255]

PS: All were found by hit and trial method 😬



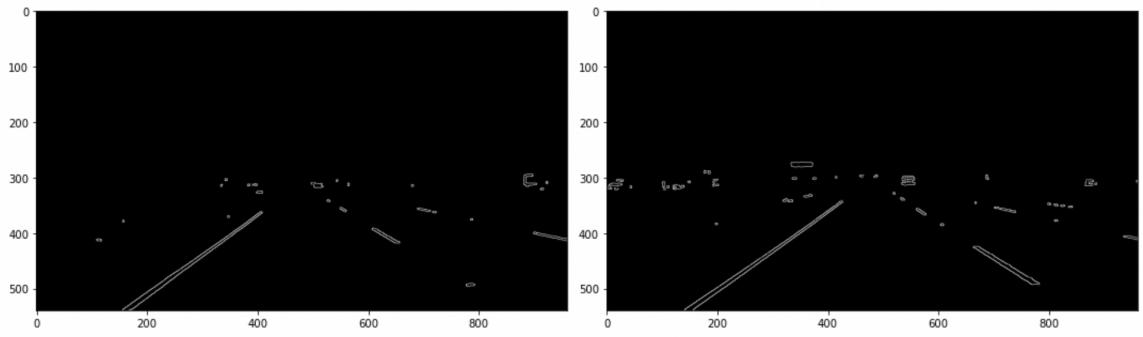
5. Apply Gaussian blur for smoothening(Select odd Kernel maybe 3, 5 or 7)

This is again a pre processing step for smoothening (Reducing Noise) of images which helps to only keep prominent edges. Here I have used kernel size as 7.



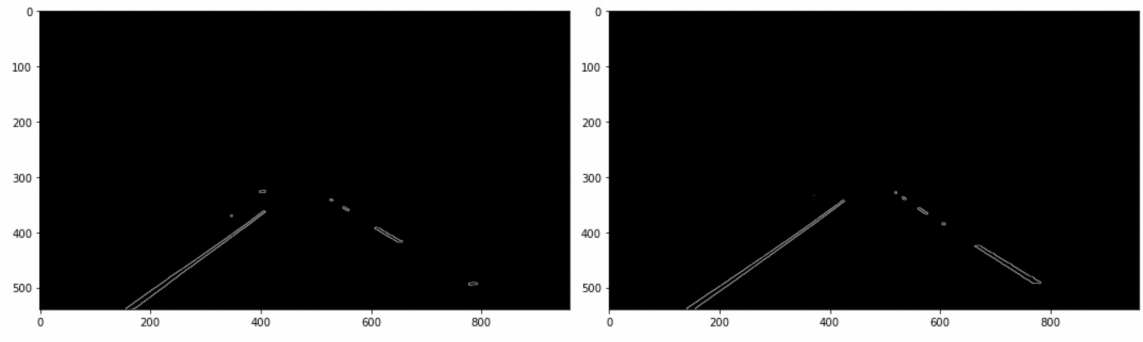
6. Select edge using Canny Edge detection algorithm.

Canny Edge detection algorithm works on gradient change principle. It detects the edges where there is a sudden change in intensity/ gradient in the pixel values. Although this algorithm applies blurring to the image however in the previous step we have explicitly applied gaussian blur for better results.



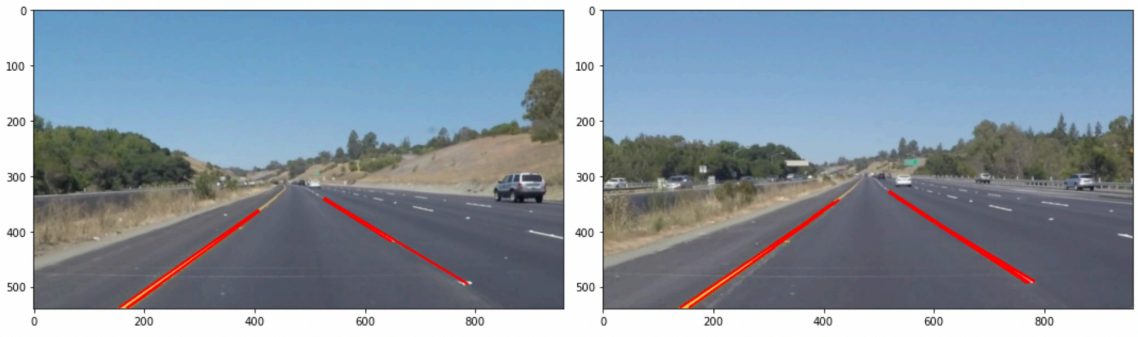
7. Define area of interest(For vehicles we will define a quadrilateral just infant of it so that we can neglect unnecessary edges.)

Even though we have applied canny edge algorithm to remove the unwanted noise, we have still a lot of prominent edges left (Sign boards, trees, bridges, hoardings etc..). So to get rid of all these unwanted objects we will define our our region on the road where our algorithm will be applied and rest will be ignored.



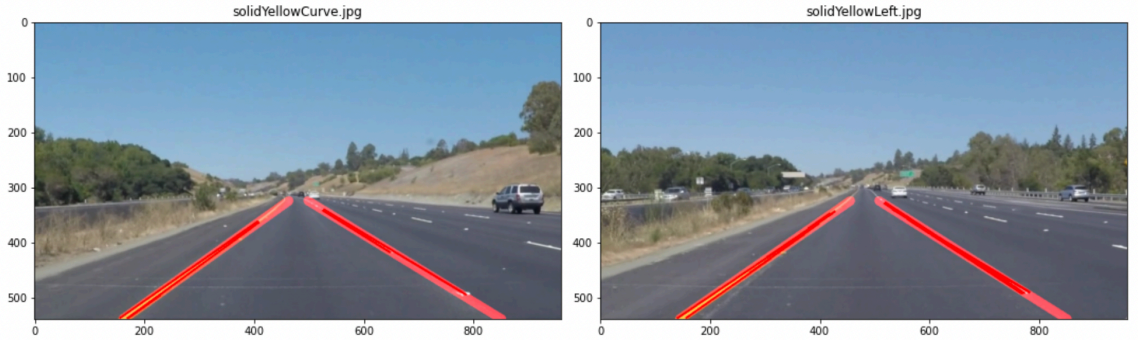
8. Define Hough lines.

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.



9. Averaging and extrapolating the hough lines.

Averaging is needed for multiple lines that are detected to come up with averaged line. For instance, suppose we have 7 lines detected on the left lane, then, we will calculate average of all the slopes and intercept of all lines on left lane. And then we will generate 1 final line with the help of averaged slope and intercept and a starting point which will always be fixed near the car on the left side. And, similarly we do this for right lane. This helps us to fill the gaps between the detected lanes on the road. (We can see in above images there is small gap in the below right corner which is filled below)



Limitations :

Since this is just a very preliminary approach to detect the the lanes there are few limitations to it.

- This only works well for roads having straight lanes. Why? Because this is build upon linear equation of straight line

  $$y = mx + c$$

  However in real world, we have tortuous roads so that will be a problem.

- Moreover there are certain environmental factors as well like shadows which could create further problems in lane detection.

Enhancements :

- In this pipeline we can implement higher degree polynomials instead of just a straight line which could solve our first limitation.
- Moreover we have just taken a simple average for extrapolating which can be improved using weighted average for each line. For instance longer the detected lane more will be weight