Overview

The goal of this project is to make a pipeline that finds the lane lines on the road using Python and OpenCV. We need to use the images and video that is provided by Udacity

Assumptions made by me

1. No environmental dependencies
2. The performance of algorithm is subject to quality of the image
3. Simple basic algorithm to detect the lanes in straight road

Reflection

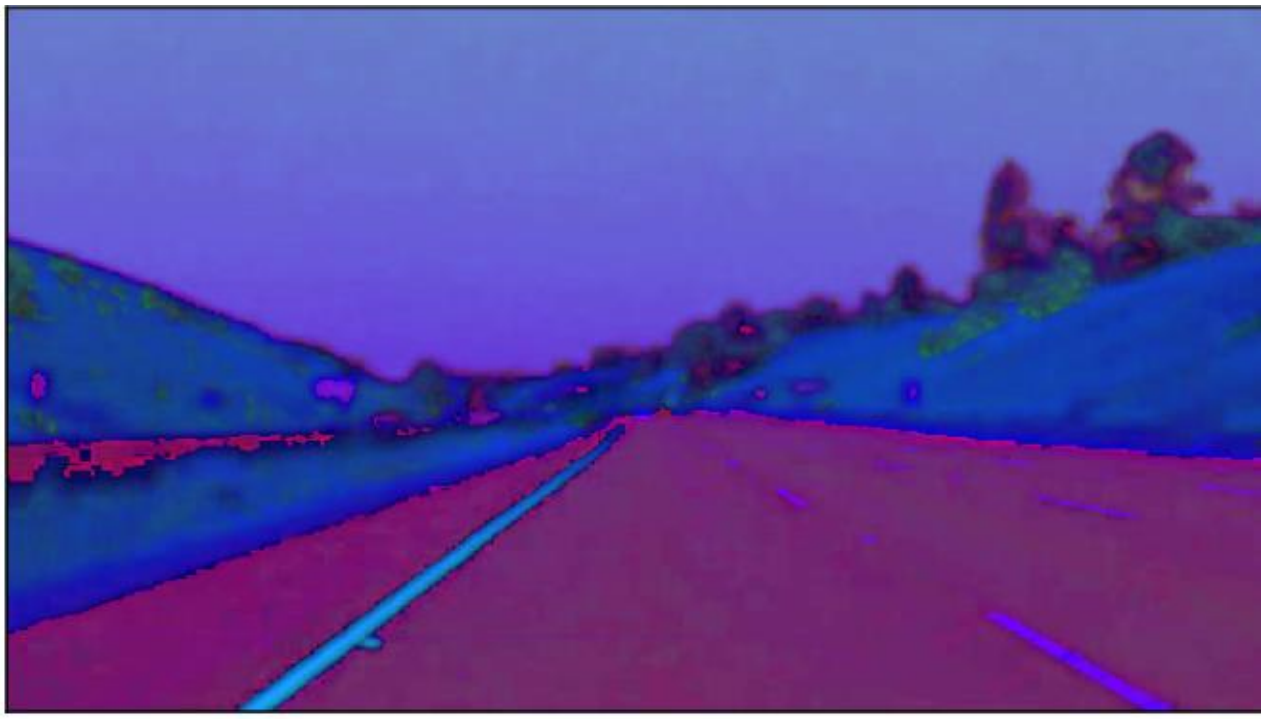
I will be using an example image and appropriate video to explain the approach (SolidYellowLeft.jpg)

Color spaces and Conversion

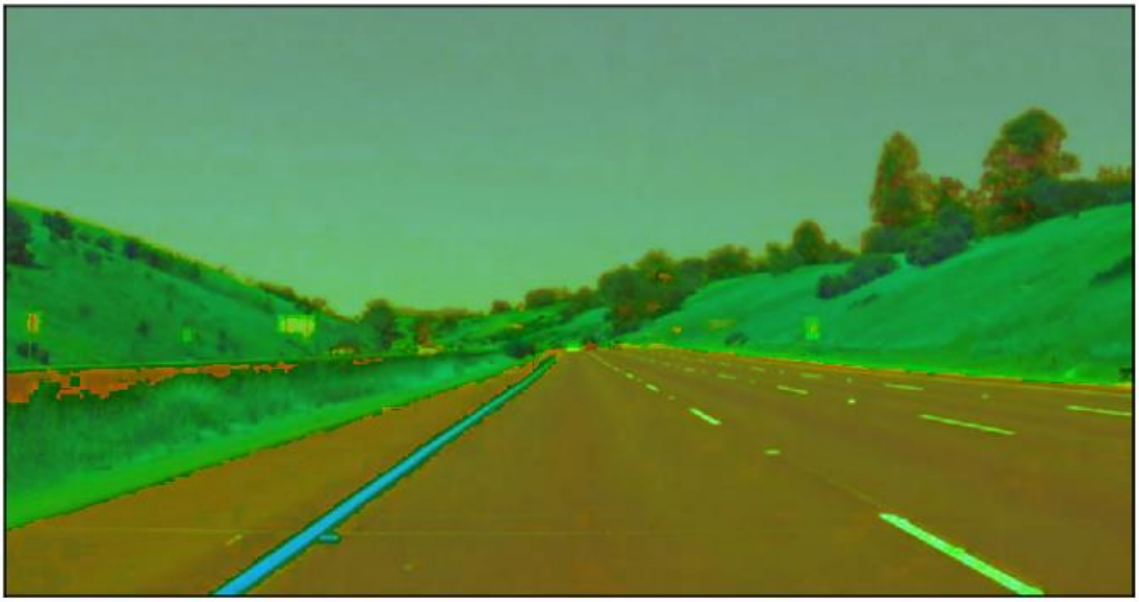
Our current image is in RBG Format and I had explored to see whether it is better to visualize in the HLV or HLS format?

On the HLS format, it seems to have better command over the images in identifying the lanes. This is visualized in the below images

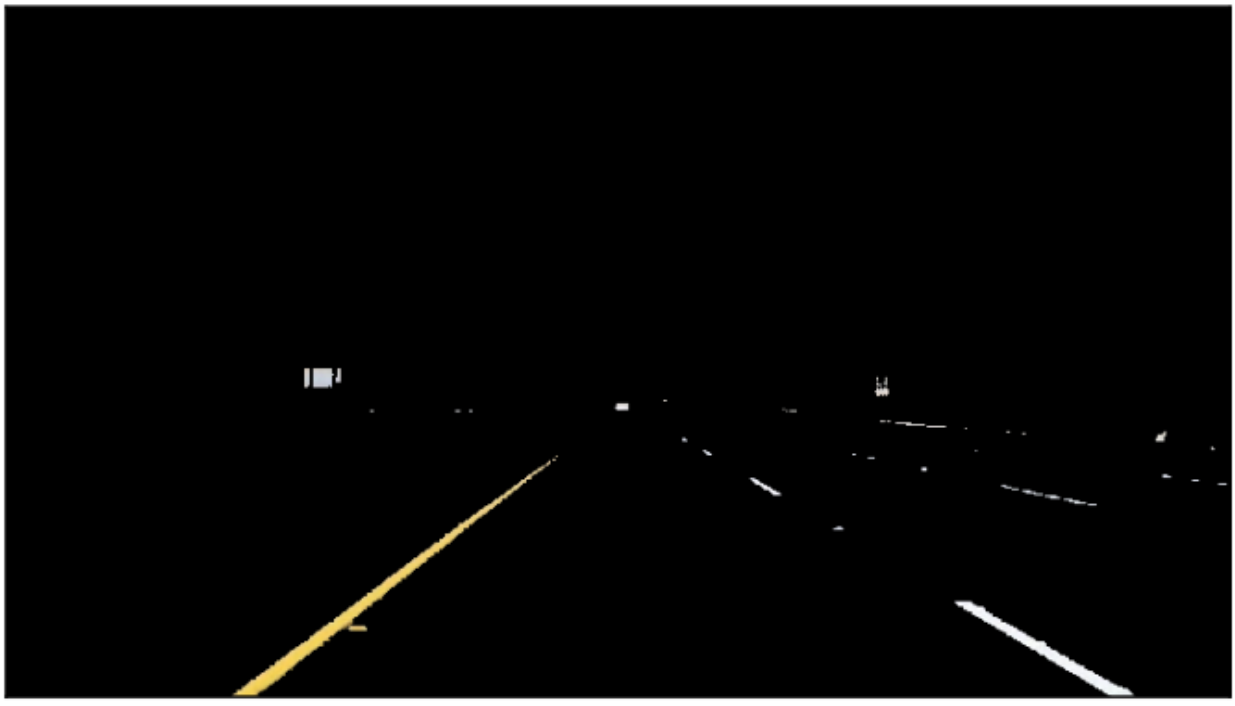
1. Convert the original image in to HSL



2. Convert the image in to HSV



3. I have applied Yellow and White Mask and filtered the image using trial and error combination



The above figure shows clearly the white lane lines and yellow lane lines identified

Isolating yellow and white lanes from the image is done by applying individual masks for white and yellow and by ORing them

Gray Scale Conversion

The images should be converted into gray scaled ones in order to detect shapes (edges) in the images. This is because the Canny edge detection measures the magnitude of pixel intensity changes or gradients

Gaussian Smoothing

When there is an edge (i.e. a line), the pixel intensity changes rapidly (i.e. from 0 to 255) which we want to detect. But before doing so, we should make the edges smoother. As you can see, the above images have many rough edges which causes many noisy edges to be detected.

Gaussian smoothing is a preprocessing technique used to smoothen edges of the image to reduce the noise.

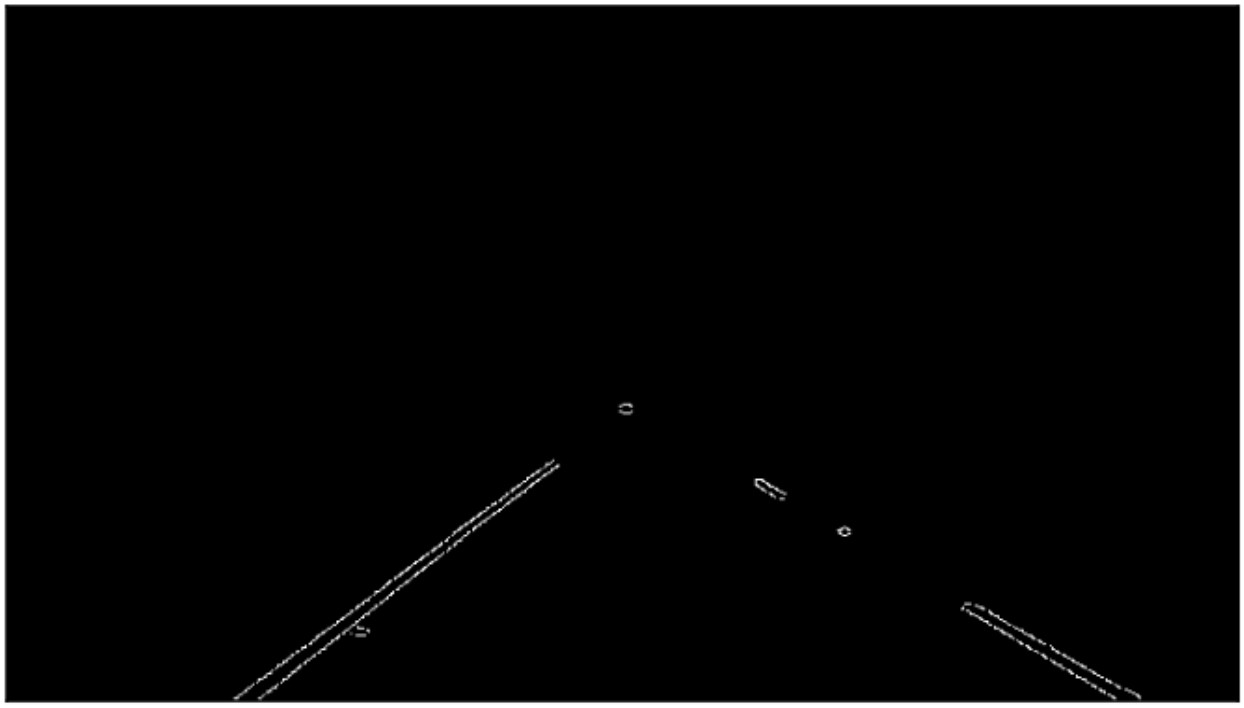
The OpenCV implementation of Gaussian Blur takes a integer kernel parameter which indicates the intensity of the smoothing. For our task we choose a value of *5.*

Canny Transform

4. By using Canny transform, we will identify the edges
   Canny edge detector focuses on identifying the edges (lines) based on gradient changes.
   https://en.wikipedia.org/wiki/Canny_edge_detector

5. The other steps in Canny Edge detection include: Finding Intensity Gradient of the Image, Non-maximum Suppression and Hysteresis Thresholding

6. The OpenCV implementation requires passing in two parameters in addition to our blurred image, a low and high threshold which determine whether to include a given edge or not. A threshold captures the intensity of change of a given point (you can think of it as a gradient). Any point beyond the high threshold will be included in our resulting image, while points between the threshold values will only be included if they are next to edges beyond our high threshold. Edges that are below our low threshold are discarded. Recommended low:high threshold ratios are 1:3 or 1:2



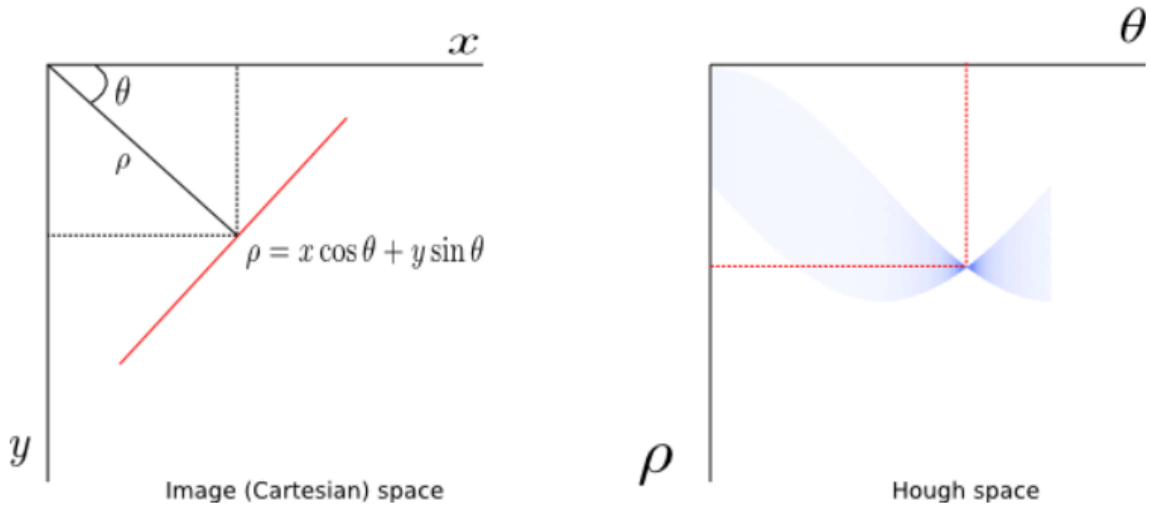
Hough Transformation Lines Detection

- Hough Transform is the technique to find out lines by identifying all points on the line. This is done by representing a line as point. And points are represented as lines/sinusoidal(depending on Cartesian / Polar co-ordinate system). If multiple lines/sinusoidal pass through the point , we can deduce that these points lie on the same line.

- Hough Transform is a popular technique to detect any shape, if you can represent that shape in mathematical form. It can detect the shape even if it is broken or distorted a little bit. We will see how it works for a line.

- A line can be represented as $y = mx + c$ or in parametric form, as $\rho = x \cos\theta + y \sin\theta$ where $\rho$is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise

7. We have applied Hough Transform technique to extract the lines and color it. The Hough transform identifies the points and find the lines. Axes in Hough coordinates are Y = mx + B where m and b are the axis
8. In this Hough transform, lines are represented as points and points are represented as lines, intersecting lines means the same point is on multiple lines
9. Hough transform use polar coordinates



Hough Space

10. So if we want to find the lines in the cartesian coordinates, we can trace the sinusoids in the polar coordinates

Slope Intercepts and Lane Interpolation calculations

11. Identify the positive and negative slopes. If the slope is negative, that is left lane(As X axis increase, the height becomes smaller) and the slope is positive, then the lane is right. This is in contradiction to normal mathematics because here the origin of the image is in top left corner
12. From the Hough lines, identify the slope and intercept and thereby find the lane lines by weight averaging method
13. Find the relative dots over the image by using the slope and intercept
14. Connect the dots and extrapolate
15. Parse over the image and draw the lines

Processing Videos

Setup

Udacity has given 3 videos

The first implementation seems reasonably worked in first 2 videos but when the lane is becoming curvy its performance is but not good

Therefore tried to interpolate the lanes and using deque technique from python to store the frames and average it. Configured the maximum queue as length as 30 and then the output becomes reasonable

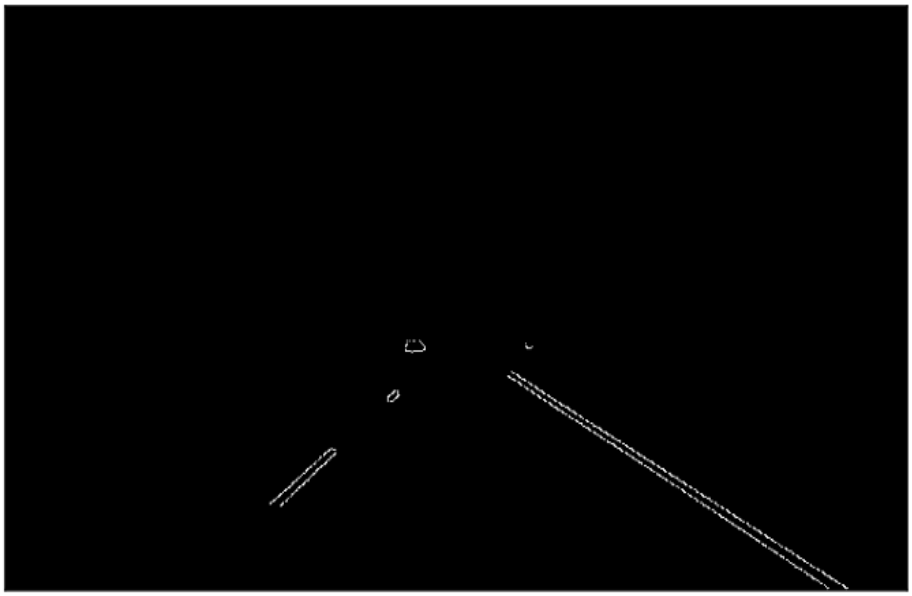
The order of polynomial is first order since y = Ax + B works for straight roads but not for curves.

Shortcomings and fix

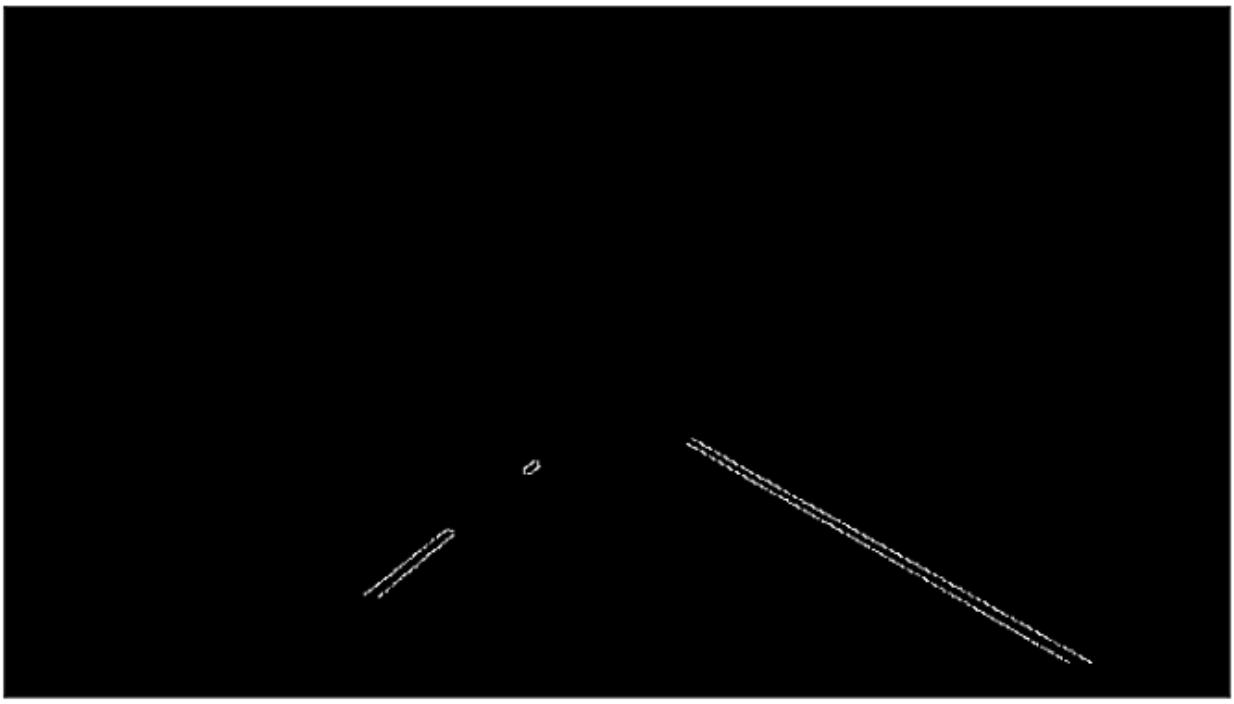
In the Initial algorithm which I developed without HLS, I saw the yellow lanes are not detected. But after color masking and conversion, we are able to detect the yellow and white lane lines.

The Kernel size is tried for 21,11,5 etc and for me the Kernel 5 took less time than Kernel 11 and 21 configurations.

When selecting the region of interest, I first tried using a triangle approach but seems to be cluttering some noise factor in the middle of images. So tried reducing the area of interest and it fitted well to detect only the lanes



With reduced fit of vertices, same image is appearing as



Future improvements

- Lane curvature needs to be estimated accurately
- More adaptive algorithms which employ higher degree polynomials such as two degree or three degree
- Curve fitting algorithm development which adapts as per the shape of the curve

Note: This project is found in git hub under the link

https://github.com/pradhapmoorthi/Project1-LaneFinding.git