PROJECT 1: FINDING LANE LINES

INTRODUCTION:

An autonomous car, unlike conventional human-driven types, are without the luxury of human sensory intelligence to guide their trajectory. It has to be equipped with its own sensory devices to carry itself and its passengers around. A suite of vision sensors installed onto the vehicle can provide the vehicle with perception ability, however, it also needs to understand what it sees and how to act upon receiving this information. The most intuitive way is mimic human behavior. The lane lines on the road act as a constant reference for a human driver to maneuver the vehicle and an autonomous vehicle can use these same lane lines to guide them.

In this project, we simply try to detect these lane lines, precisely using cameras. Detecting lane lines, albeit sounds straightforward, in reality can be made difficult due to varying surrounding conditions. These include change of lighting, shadows, adverse weather conditions, road color changes, and so on. Through this project we get a feel of detecting lane lines using Python and OpenCV library and prepare ourselves to detect lanes in various scenarios.

The goal of this project is to make a pipeline to find lane lines on a road, reflect on the potential shortcomings of this pipeline and suggest improvements.

PIPELINE DESCRPTION:

The output after detecting lane lines and final output should resemble the below pictures.

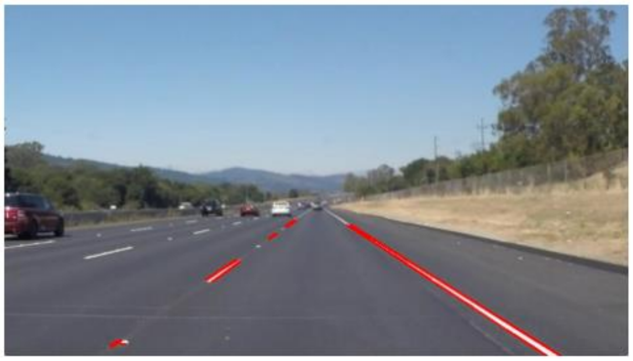

Figure 1 Lane Marking Detection

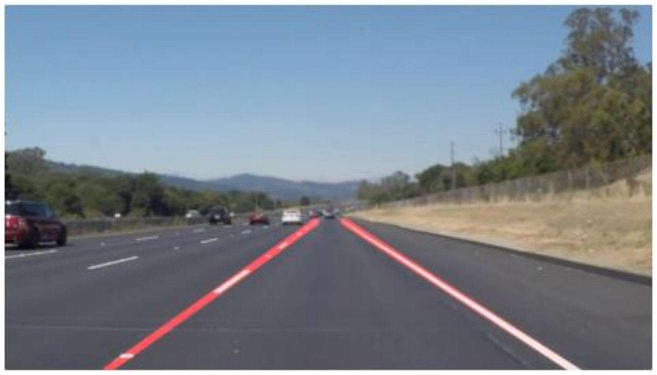

Figure 2 Final Lane Detection

Setup: Importing useful packages including ‘numpy’ for numerical operations, matplotlib for reading images and plotting images, cv2 for image processing, and math for mathematical operations. Then read test images to test and improve different helper functions, and finally the pipeline. OpenCV uses BGR as its default color order for images, whereas matplotlib uses RGB. When you display an image loaded with OpenCV in matplotlib the channels will be back to front. The easiest way of fixing this is to use OpenCV to explicitly convert it back to RGB.

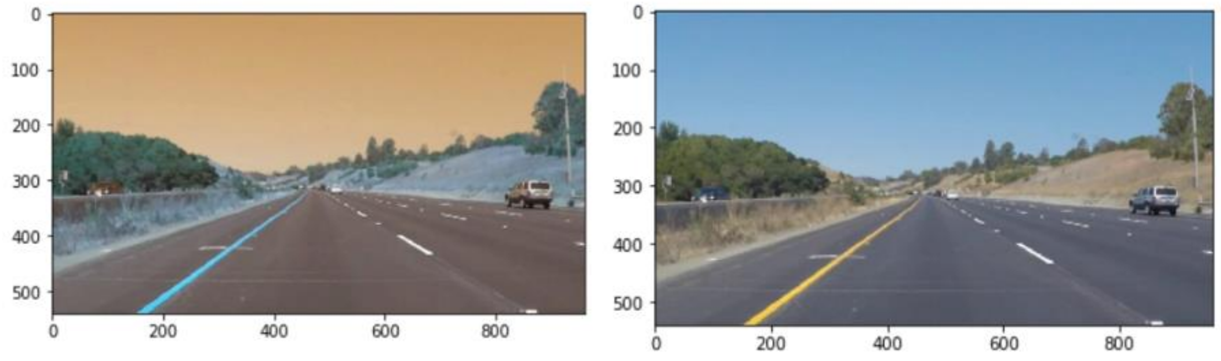

Figure 3 Sample Test Image – (a) BGR , (b) RGB

Step 1: Smoothen the image using GaussianBlur cv2 function. This reduces image detail and noise.

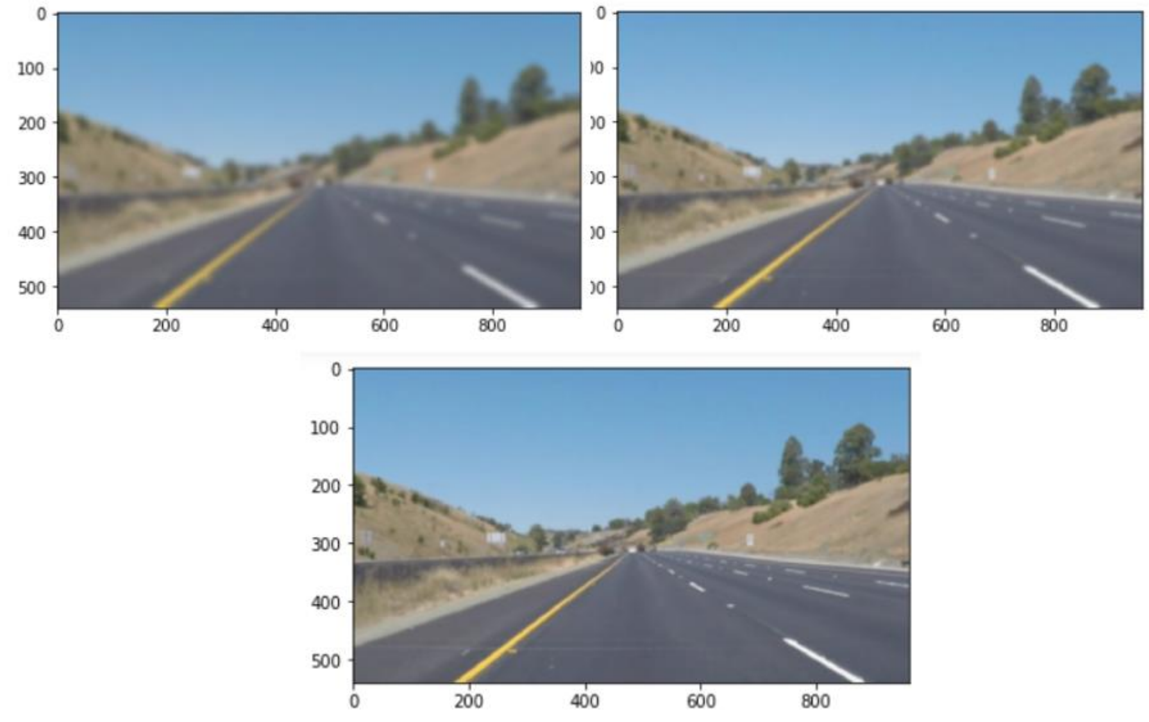

Figure 4 Varying image detail by varying kernel size - (a) 33 (b) 13 (c) 7

Step 2: Convert image to gray scale using cv2 cvtColor function. The consequent steps require a gray scale image to process, hence this step. A color image is a 3D matrix. But a grayscale image is a 2D matrix in which there are no colors. So, matplotlib applies a colormap by default. Therefore, applying a 'gray' cmap while plotting will do the job.

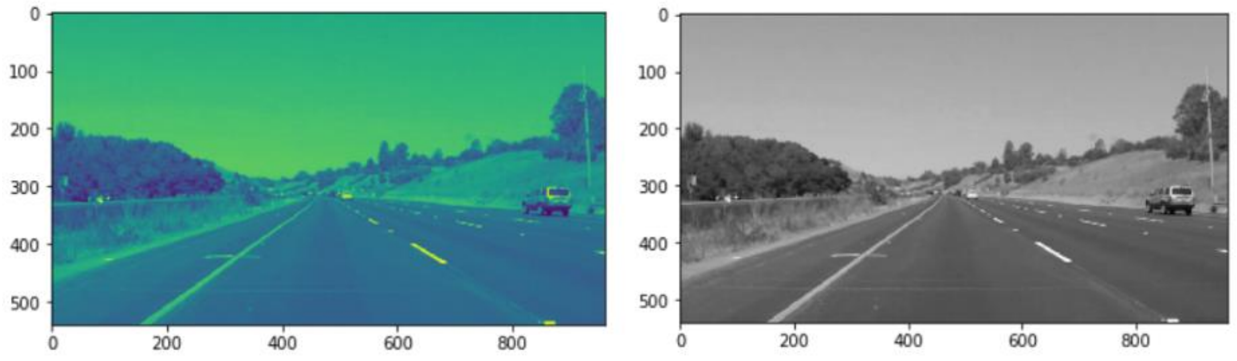

Figure 5 Gray scale image - (a) default (b) gray cmap

Step 3: Canny edge detection using cv2 Canny function. The Canny function takes a gray scale image as its input and detects edges based on the gradient threshold. It first detects strong edge pixels above the high threshold and rejects pixels below the low threshold. Then, pixels with values between the low threshold and high threshold will be included as long as they are connected to strong edges. The output is a binary image with white detected edges.

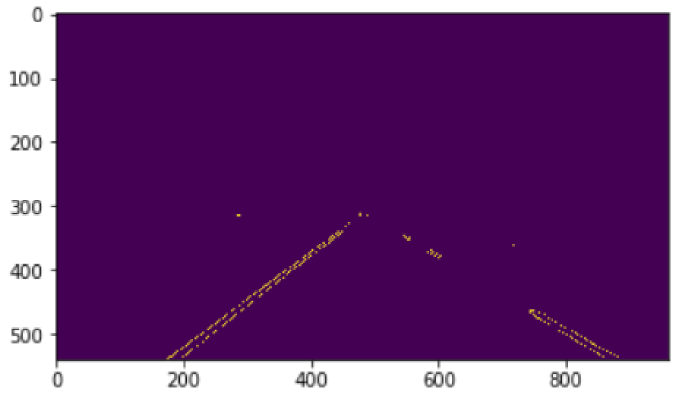

Figure 6 Canny Image with low and high thresholds as 150 and 255 respectively

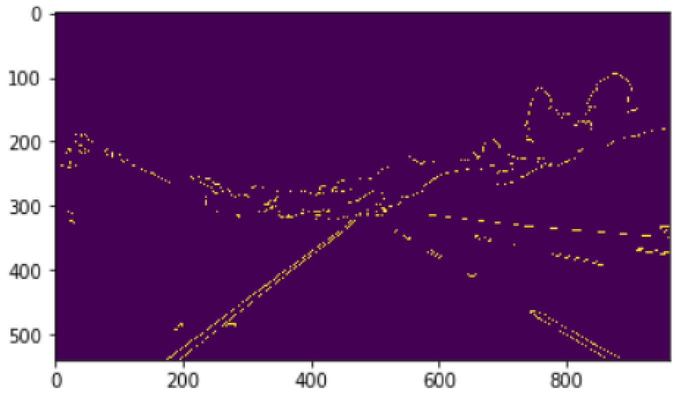

Figure 7 Canny Image with low and high thresholds as 50 and 155 respectively

Step 4: Applying a mask. A mask converts all the irrelevant pixels on the image to zeros, thus passing only required information forward. This is done by defining a region of interest, passing its vertices to a cv2 fillPoly function and returning the image only where this mask pixels are non-zero using a cv2 bitwise_and function.

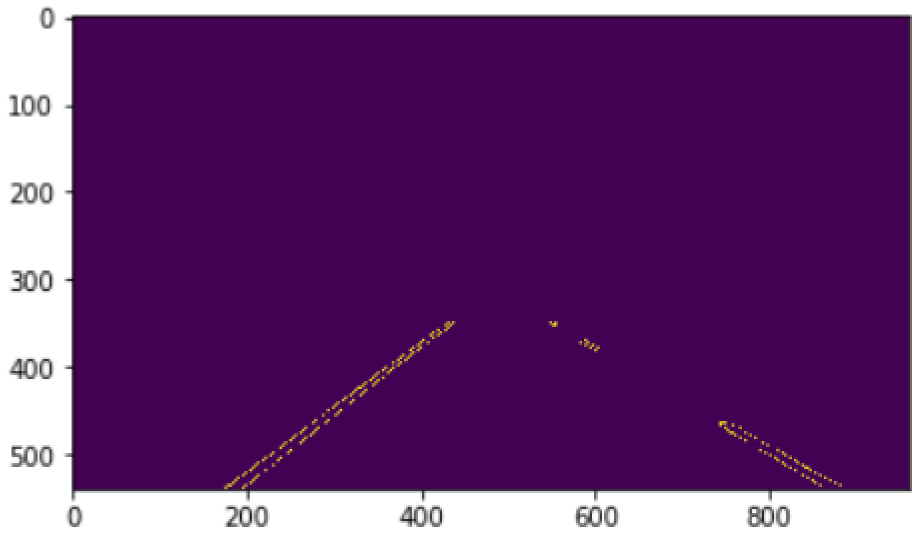

Figure 8 Mask Region of 1s and 0s everywhere else

Step 5: Detecting lane lines using cv2 HoughLinesP function. The HoughLinesP returns an array containing endpoints of all detected line segments. It transforms the conventional current co-ordinate system into Hough space, distance and angular resolution as each co-ordinate. These are specified based of which lines with specific alignment and distance from origin are detected. The threshold parameter specifies the number of intersections, called votes, signifying a line passes through the pixel points transformed to lines in hough space. Minimum line length and maximum line gap between segments that are allowed to be connected into a single line are also passed as arguments.

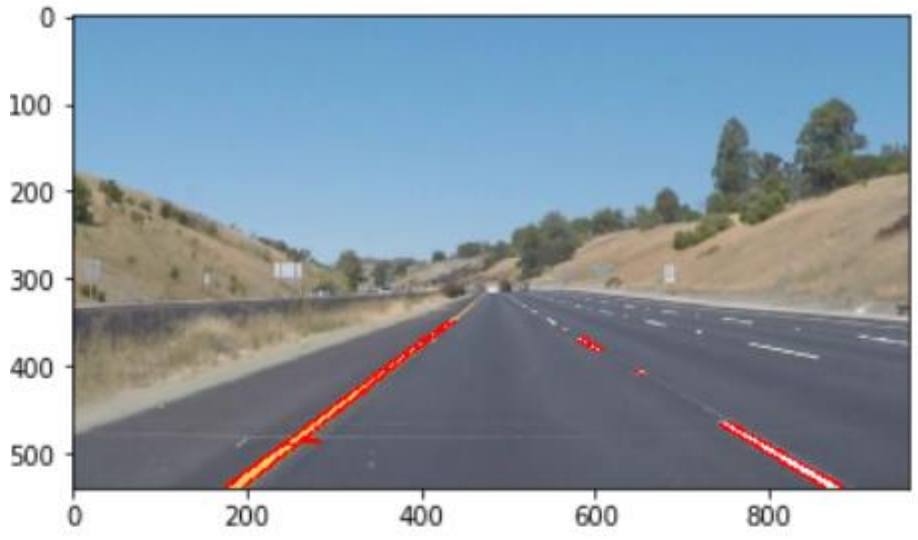

Figure 9 Detected Lane Markings in Red

<u>Step 6:</u> Interpolating detected line segments into lane lines. Using the line segment data returned from the previous step, we can differentiate left lane and right lane lines by computing slopes. The left lane has positive and right has negative slopes. After separating both the lanes, we can compute array of slopes and intercepts. Filtering this data by taking averages of slopes and intercepts within allowable range of their median for eliminating outliers, and selecting start and computing end lane points, by slope intercept form, we can draw lane lines using cv2 Lines function.

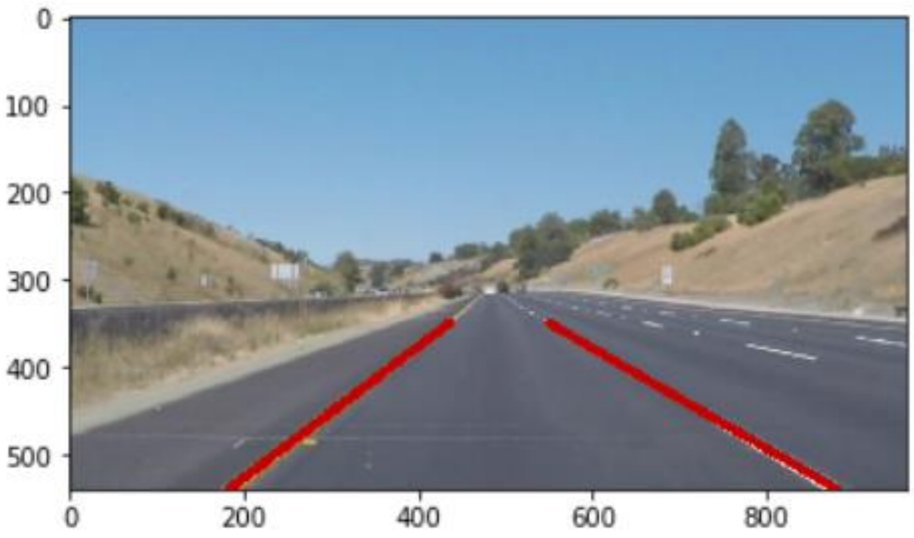

Figure 10 Detected Lane Lines by extrapolation

Additionally, the detected lane lines can be made translucent and can appear in different colors by adding weights using cv2 addWeighted function.

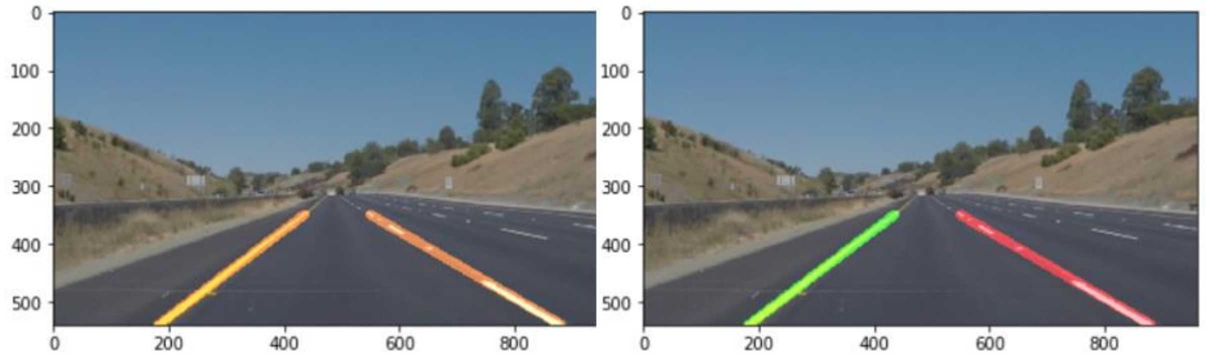

Figure 11 Adding transparency and color schemes

This pipeline is the foundation for the process_image function which takes each frame of a video as an input argument and returns the processed image with lane lines marked on them.

EXPECTED SHORTCOMINGS:

1. The pipeline is not robust to changes as the hough, canny threshold values are often specific to scenario. By changing road conditions, as experienced to an extent in the second challenge and considerably in the third challenge, the pipeline would need tailored attention.

2. The pipeline does not perform well in large curvatures and sudden curvatures as the current manually applied mask might not be applicable. Also, for straight roads, the slopes and intercepts remain consistently constant. So, the current extrapolation logic may not work.
3. Large road gradients would also pose a problem for a manual mask.
4. In the presence of shadows and road color changes, the current pipeline might not be effective as canny threshold modifications might be required.
5. Adverse weather conditions might disrupt the input image quality, making lane detection almost impossible.

SUGGESTIONS FOR IMPROVEMENT:

Most shortcomings can be overcome by changing the approach to image processing adopting advanced lane line detection strategies that are explored in the next project. These include:

1. Perspective transform for changing to a birds-eye-view for ease and improvement of masking and tackling curvature issues.
2. Applying sobel thresholds, an advanced gradient thresholding method to canny, in various orientations, directions and combinations. This gives more options to tackle diverse problems.
3. Exploring color spaces is important in tackling varying lighting, colors, and presence of shadows.

Apart from these, additional sensors like LIDAR, RADAR, etc., and sensor fusion will help overcome shortcomings of cameras.