Finding-Lane-Lines--Project-1
First Project in the Self Driving car Nano degree program

We use our eyes to see the road while driving. In this project we give vision to the car, Computer vision. The lines, signs and signals on the road shows us to drive according to the rules. Here in this project, we take only lines (in proper words lanes). In this project, an algorithm is made to detect the lane lines automatically.

There are lots of tools used in this project. They are color selection, region of interest selection, grayscaling, Gaussian smoothing, Canny Edge Detection and Hough Tranform line detection.

Reflection

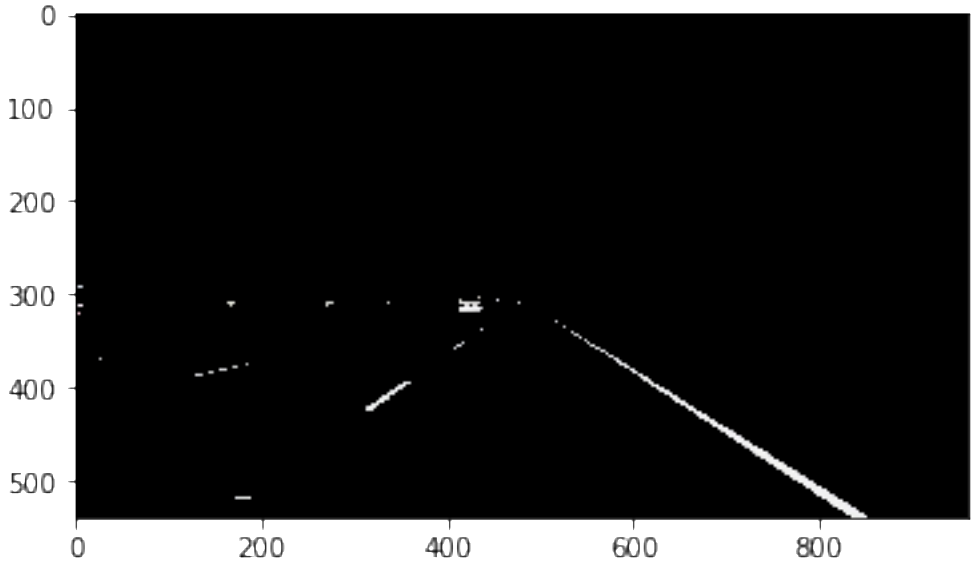
Firstly, read the Image and print out some necessary stats so that we'll get the dimensions and then visualize (plot) the image.

Grab the x and y sizes and make two copies of the image With one copy we'll extract only the pixels that meet our selection, then we'll paint those pixels red in the original image to see our selection overlaid on the original.

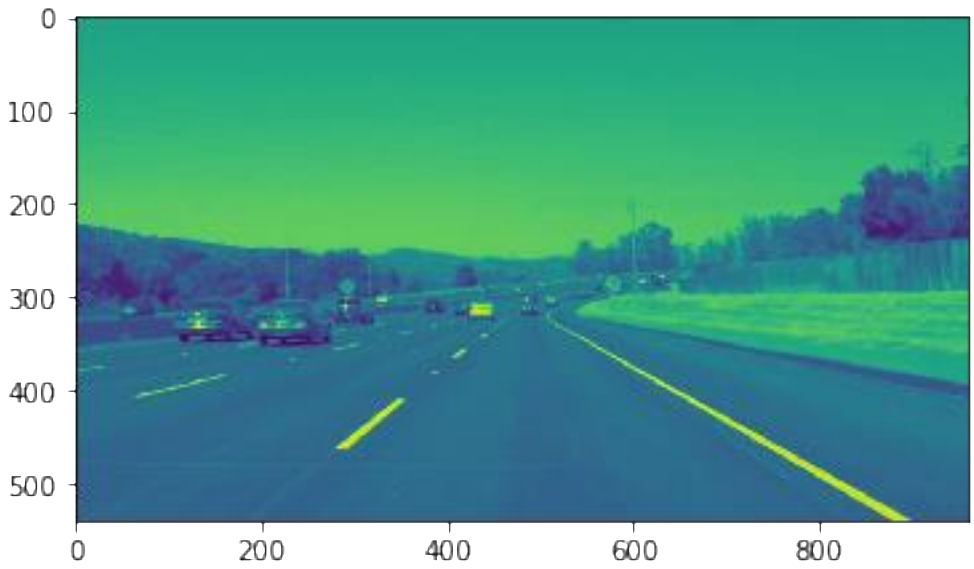
Color criteria is defined and the variables are modified to make the color selection.

Next, what we need is the triangular kind shape. So now Let's let's get the region of interest. Here, we introduce three new parameters which are left_bottom, right_bottom and an apex. So now, we'll modify these values according to our need to isolate the region. Key Point: As per image processing, the origin (x=0, y=0) is in the upper left corner.

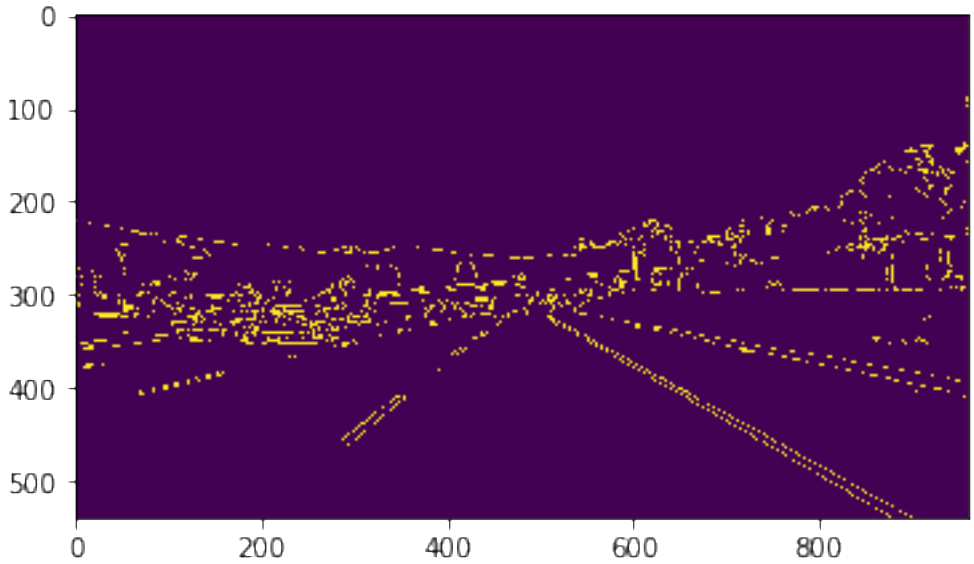
Now let's Perform a linear fit which is y=Ax+B to each of the three sides of the triangle. Key point: np.polyfit returns the coefficients [A, B] of the fit



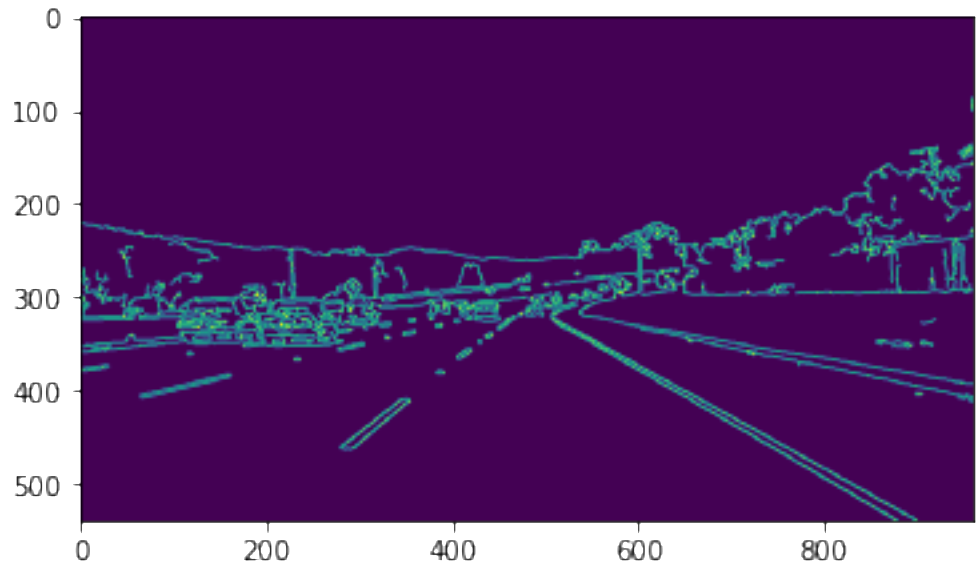
First step in the pipeline is to convert our original image into grayscale image. Module named OpenCV will be helpful to do this.



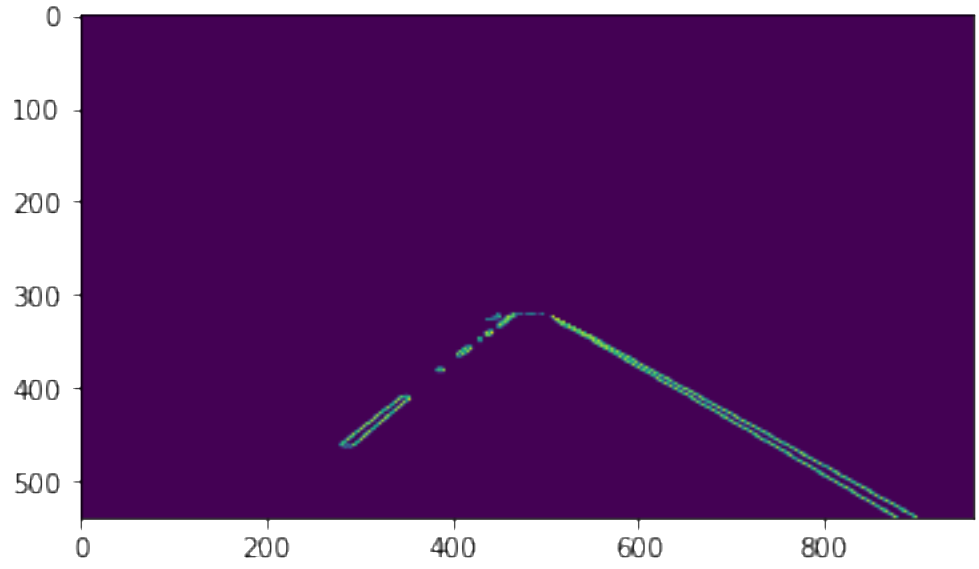
Second step in the pipeline is using Canny edge detection. Here I'm applying the canny function to the grayscale image. Low threshold and High threshold determines how strong the edges must be detected. Strength of the image is proportional to the pixel in the image. The algorithm will first detect strong edge (strong gradient) pixels above the high_threshold, and reject pixels below the low_threshold. Next, pixels with values between the low_threshold and high_threshold will be included as long as they are connected to strong edges. The output edges is a binary image with white pixels tracing out the detected edges and black everywhere else.



Thirdly, let's apply Gaussian smoothing.Gaussian smoothing is a way of suppressing noise and spurious gradients by averaging. Kernel_size for Gaussian smoothing should be any odd number. A larger kernel_size implies averaging, or smoothing, over a larger area.

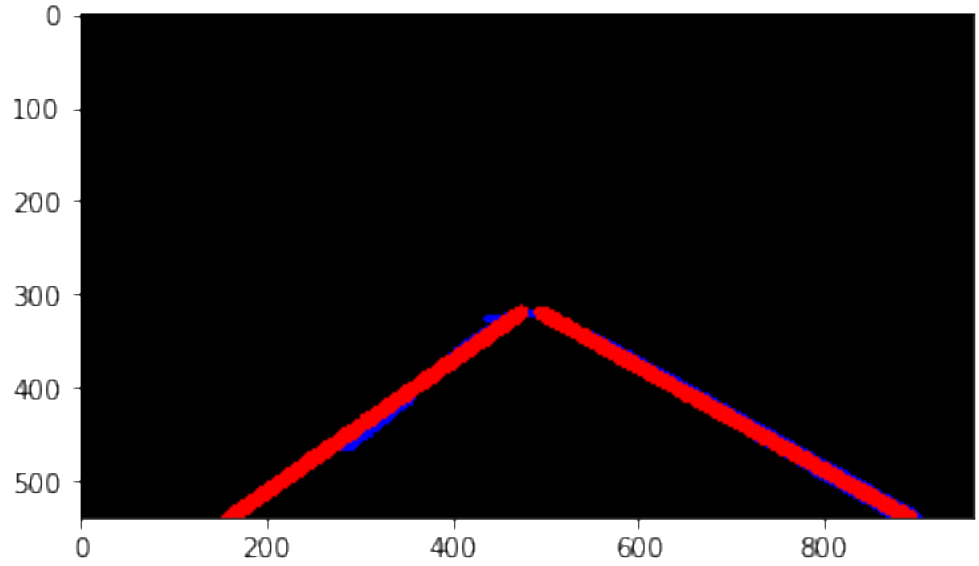


Fourth step, we have only dots in the image. Now let's make into line by connecting the dots. The Hough Transform is just the conversion from image space to Hough space.Example, dot in image space is line in hough space. If many line in the hough space intersect, we can conclude that we found the line. cv2.fillPoly() function is used to mask the quadrilateral region.

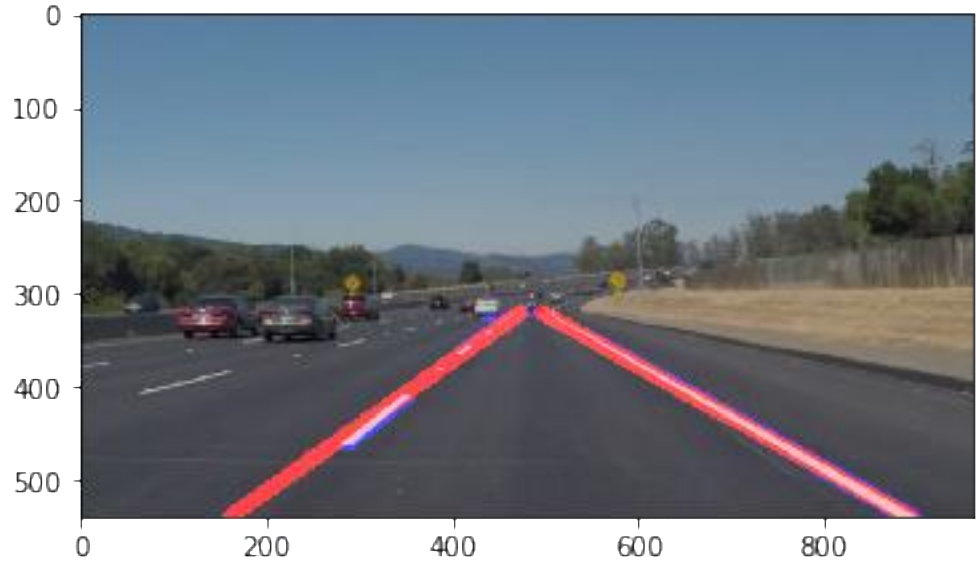


Fifth step, As of now, we have a only the line. Now let's map out the full extent of the lane using average/extrapolate of the line segments. get_slope fuction helps in separating line segments by their slope. draw_lines function draws lines with color and thickness. Lines are drawn on the image inplace (mutates the image).

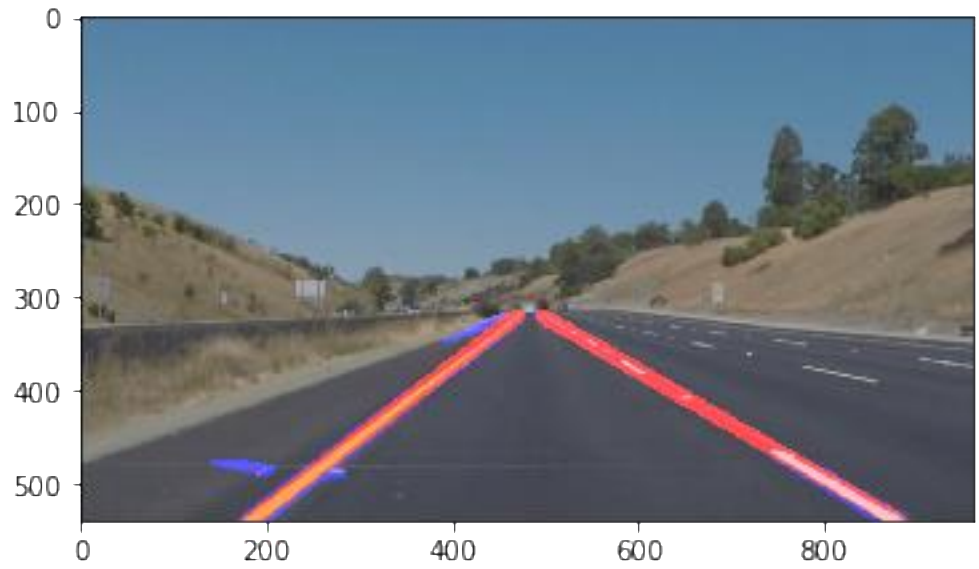
Sixth step, Here rho is the distance resolution in pixels of the Hough grid, theta angular resolution in radians of the Hough grid , threshold is the minimum number of votes (intersections in Hough grid cell), min_line_length is the minimum number of pixels making up a line, max_line_gap is the maximum gap in pixels between connectable line segments.These values are adjusted in such a way that it constitutes a line.
The output from HoughLinesP will be lines, which is an array containing the endpoints (x1, y1, x2, y2) of all line segments detected by the transform operation.



Seventh step, we should combine the above image with our original image using cv2.addWeighted function. And the output is the combo image of the both.



Here comes the pipeline implementation for Series of 5 original images. Whatever we did from the first step to fifth step, let's put all together in the same order and iterated it for all the 5 images one by one.



Videos are made up of series of images. Lets make two videos one with continous white line on left side of the car and other with yellow line on the right side of the car and named the videos as `solidWhiteRight.mp4`, `solidYellowLeft.mp4`
Now lets import the necessary packages to make the video and to view the video on the webpage. Module named VideoFileClip and HTML are used to do this.

Let's try the one with the solid white lane on the right first.

https://www.youtube.com/watch?v=P67niS1o2WU

Now for the one with the solid yellow lane on the left.

https://www.youtube.com/watch?v=BU-DZKNNPC4

Potential Shortcomings in the pipeline:

This algorithm wont work properly if the shades from the tree interrupts the lane. Also, with the noise as well.

Possible improvements in the pipeline:

One possible improvement would be to play with the values to find out more accuracy.

Another potential improvement could be to change the method in draw_functions.