Finding Lane Lines on the Road

Finding Lane Lines on the Road

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Reflection

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

I had created a threshold values for the slope , which would in turn decide which side of the lane ,the line should be. Having calculated the Slope and the intercept, it was easier to find the positions at which the lines had to be drawn.

Main pipeline

My pipeline consisted of 5 major steps namely:

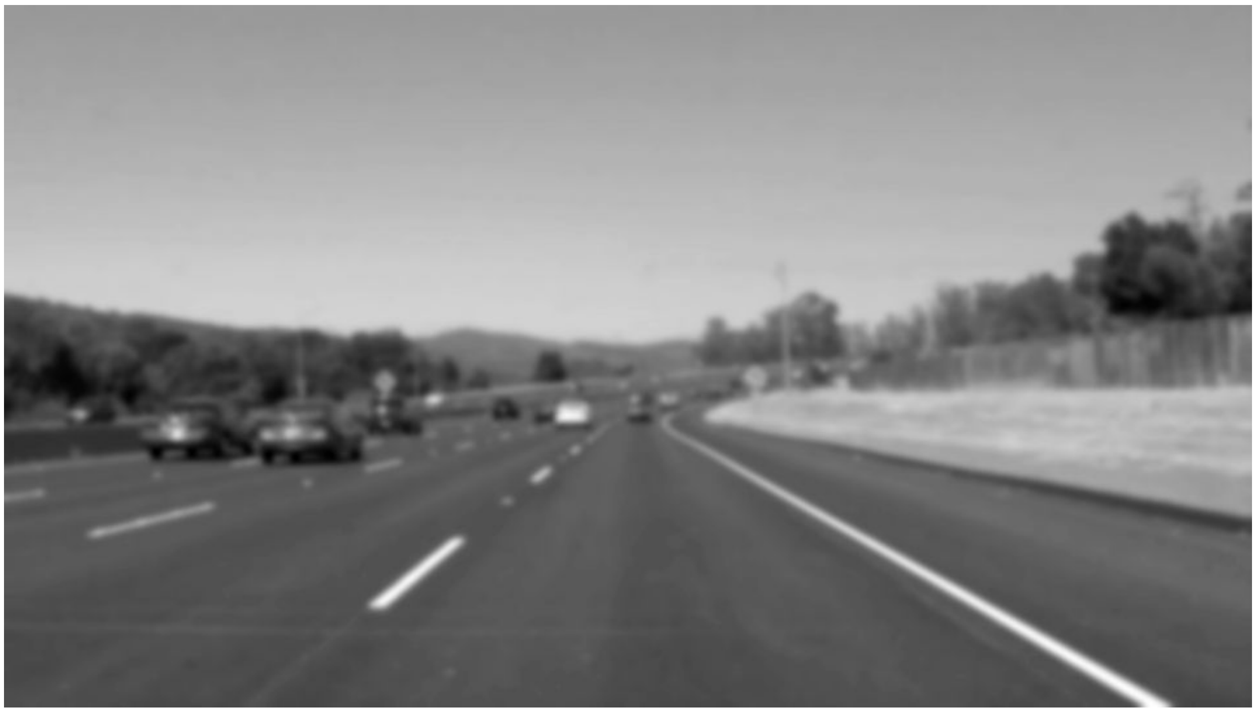
- Grayscale
- Blur
- Canny Edge Detection
- Masking
- Hough Transform

Grayscale

I also had to used some functions for displaying the Grayscale images. But I had not used this gray scale image as I faced performance issue in challenge video file. The brightness and contrast change makes the car go blind for when I use the grayscale image .This is my general Grayscale function used from the OpenCV library :

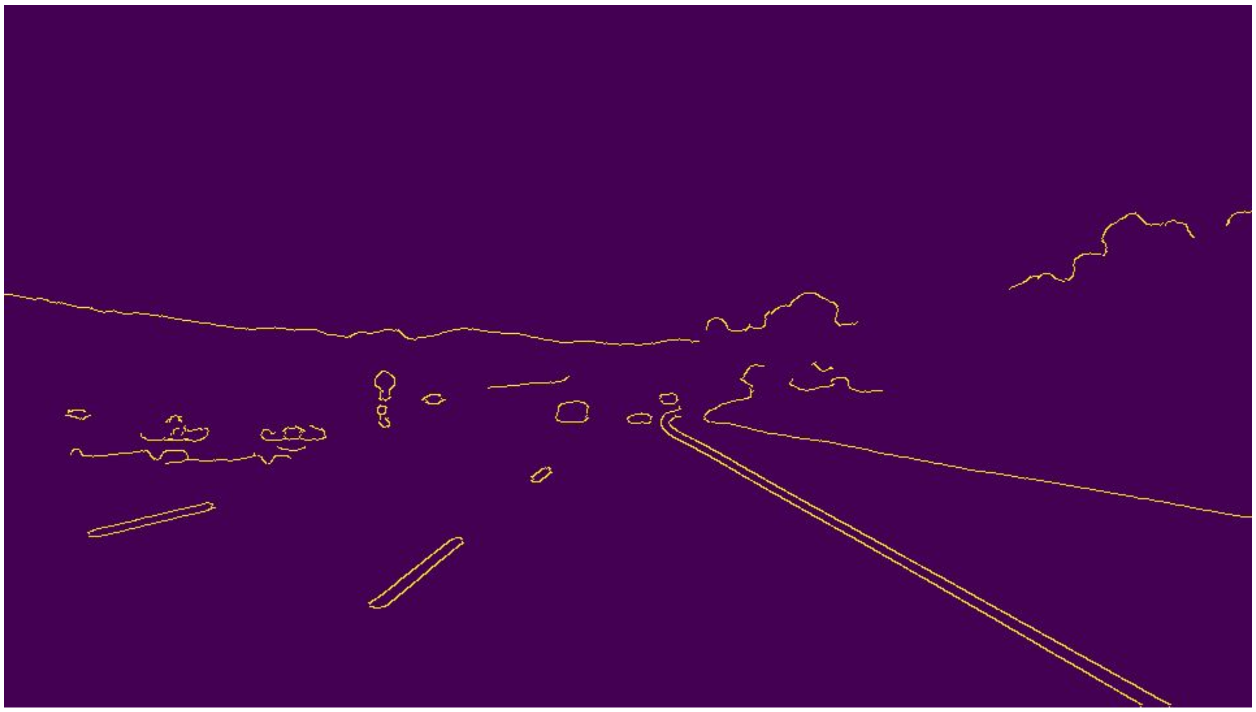
- cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
- gray_image = grayscale(image)

Blur



The Gaussian Blur is simple and uses the function to produce a blurred images from the Grayscale image. Here is my function, I used a kernel size of 15.

Canny Edge Detection

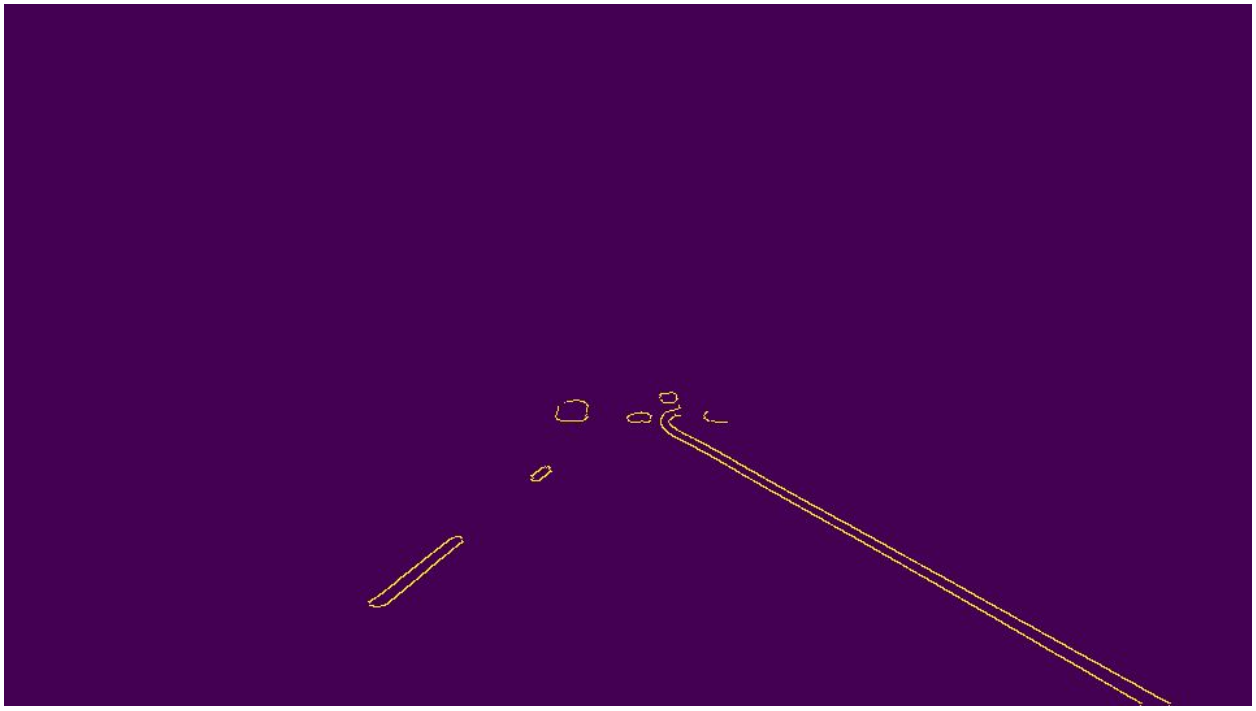


My Canny Edge Detector had a the following Threshold values:

1. low_threshold = 50
2. high_threshold = 150

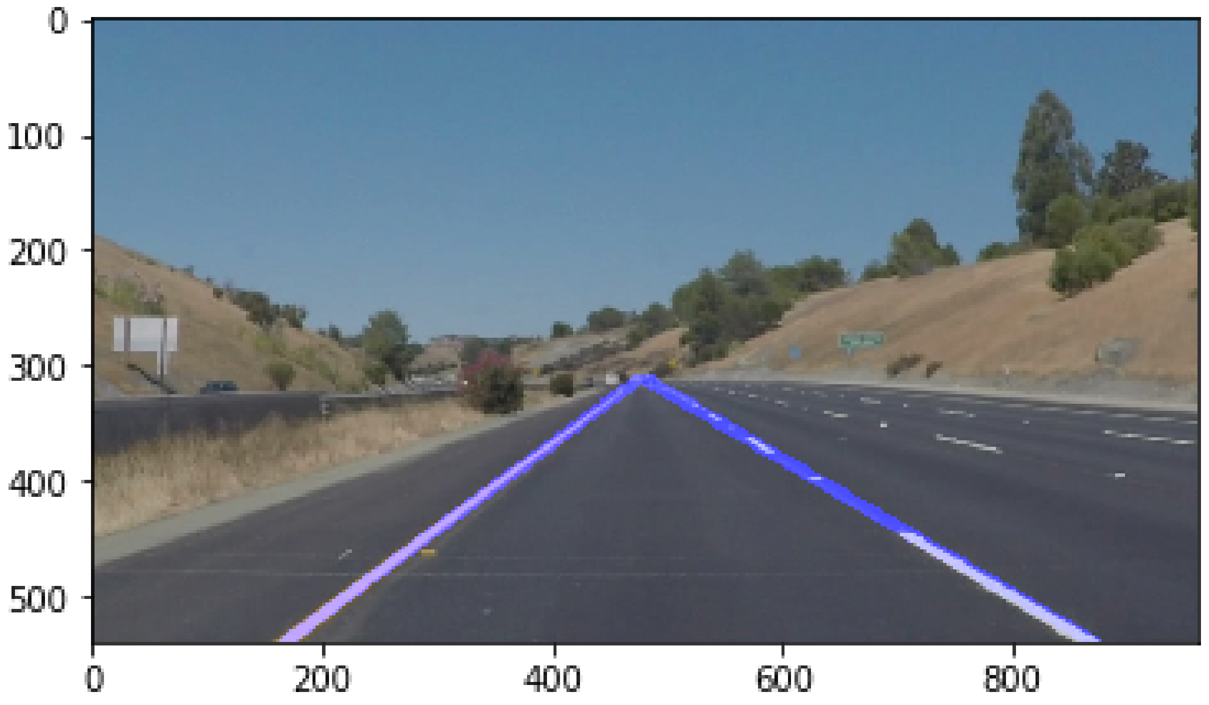
Edge detection using Canny Filter - kernel_size = 11.For kernel values less than 11 noisy lines were detected, and for values greater than 11, the filter was aggressively removing the lanes.Isolating the region of interest using a polygon - trapezoid with apex approximately halfway in the y direction (0.58 times y).

Region of Interest



My region of interest worked pretty well. I used the values that we had for the examples and test.

Hough Transform



I used a hough transform with the following parameters:

- rho = 1
- theta = np.pi/180
- threshold = 5
- min_line_length = 100
- max_line_gap = 160

Hough Transform for line detection - rho = 1, theta = np.pi/180, threshold = 20, min_line_length = 40, max_line_gap = 300. These settings ensure that even the faint lane lines are detected. And the result is the image above. I managed to merge too. Lastly I tested it on videos after modifying the draw_lines function.

2. Identify potential shortcomings with your current pipeline

One potential shortcoming would be what would happen when the test images were changed with other images whose contact is different. No matter how much I tweaked my Canny edge detector, I still has issues detecting the lanes. In the Challenge Video, the issue escalated as there were shadows which would naturally alter the brightness of the image.

Corners: When the lane takes a corner, my pipeline gets 'lost' especially in the last challenge. Either the left lane line or the right laner line would go off.

3. Suggest possible improvements to your pipeline

A possible improvement would be to add more code to handle different types on image parameters i.e Dynamic threshold value. The code could have also included Temporal Filtering where the edge detection and the lines drawn would have been much smoother.

Several thresholds and filters in the approach are hardcoded, and with changes in conditions such as lighting and road slope, these may not work as desired. Another issue is that curved lanes are detected as straight lines. It might be interesting to capture the curvature. Further, the lane lines are jittery, introducing additional complexity in the process.