Finding lanes on the road

The goal of this assignment is to find lanes on the road. Given a jpg image of a road, this program finds the lanes in the road (both solid and dotted lines) and tries to extrapolate this for the entire length of the road.

In this document, I discuss the pipeline I applied to solve this problem. During this course, I also looked at some other opencv algorithms used in smoothing the images. The results of my experiments are also documented here

Pipeline:

The following is the pipeline used given a color image (the same idea is applied to videos where these steps are done for each frame):

1. Given a color image, the first step is to convert it to grayscale.
2. The gray scale image goes through a noise reduction process and smoothing of image. Gaussian blur is used for this process with a kernel size of 5. I tried other different smoothing functions like, average, median and bilateral. The results of them are provided in the discussion below.
3. Next step is apply Canny edge detection with lowthreshold= 50 and high threshold=150
4. The region of interest (the lanes in case) is determined using a polygon in the picture. Hough transformation is applied on this area of interest to determine the lanes.
5. Step 4 returns segments of lines as identified from the jpg image. This is then extrapolated using the equation of a line y=mx+c to extend to the region of interest.

Discussion:

Converting to grayscale is straightforward. I will discuss below the results of applying varying smoothing algorithms on the grayscale image.

1. Median (kernel size = 5)

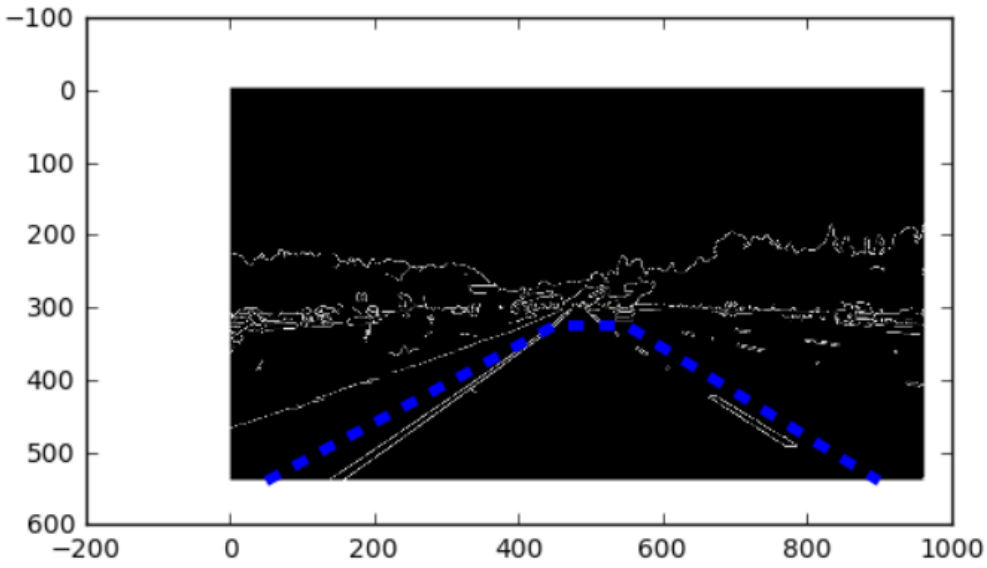

Figure 1: Median Blur in grayscale

Figure 2: Hough Line for Median blur

2. Average (kernel size = 5x5)

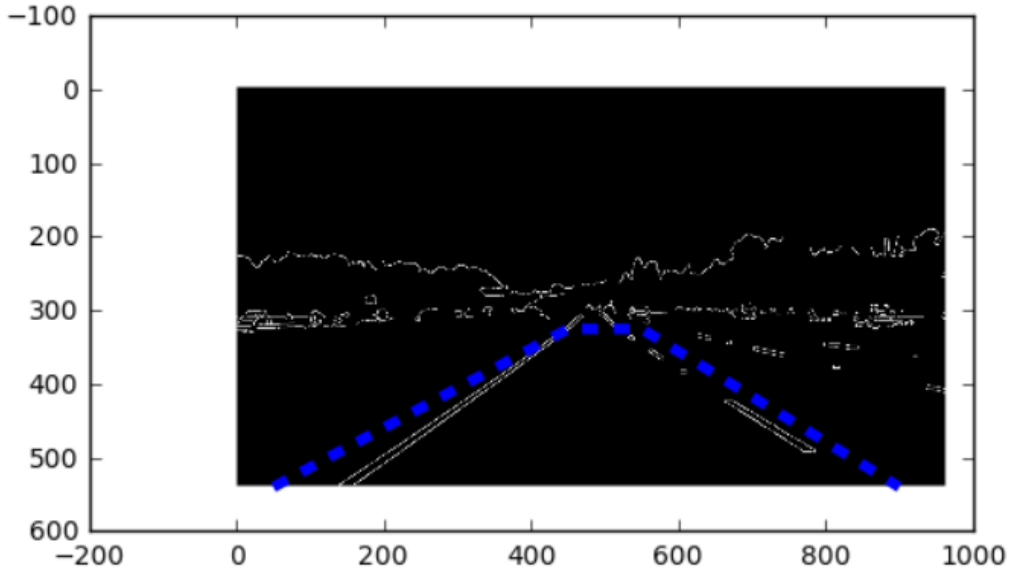

Figure 3Average Blur filter in grayscale

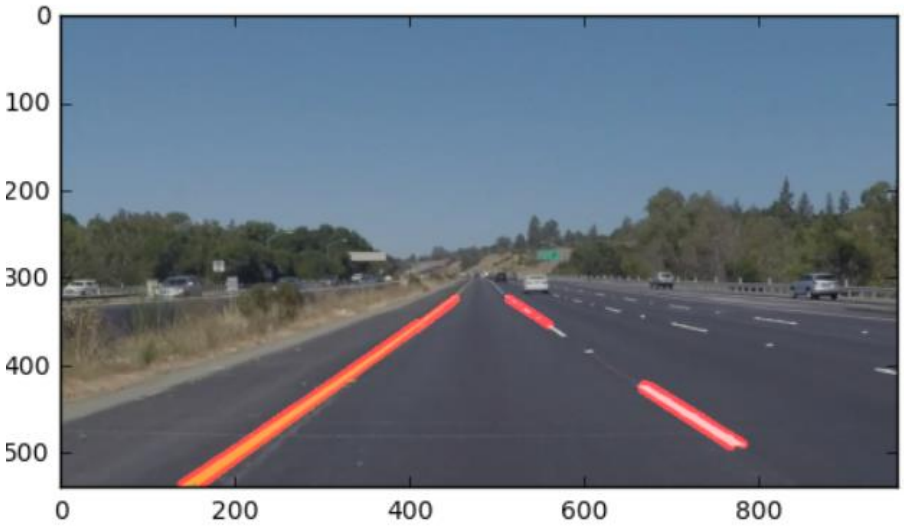

Figure 4Hough Lines for Average Blur Filter

3. Gaussian (kernel size = 5)

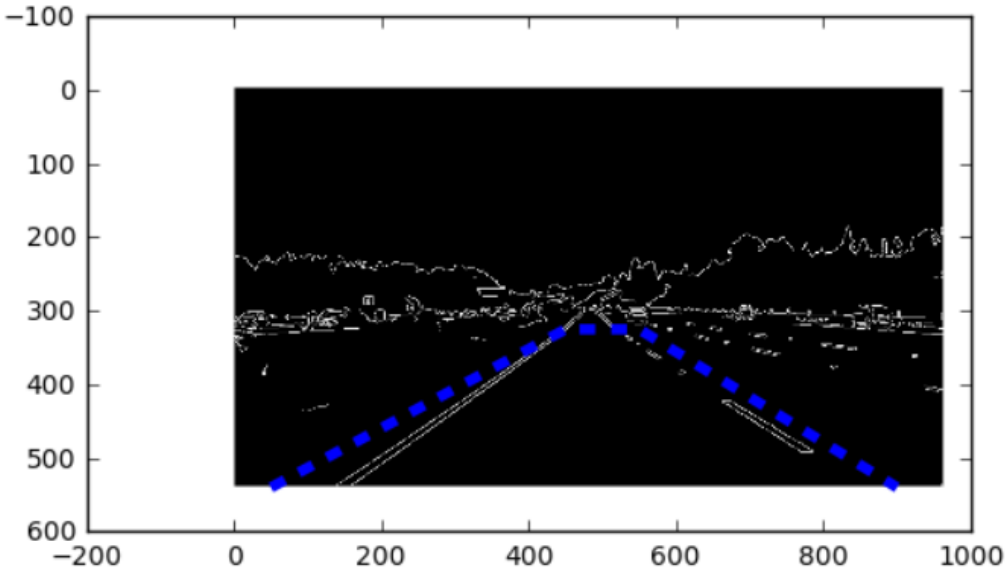

Figure 5: Gray scale image with Gaussian blur

Figure 6: Hough lines for Gaussian blur

4. Bilateral (d = 5, sigmaColor=sigmaSpace=75)
   This yielded results that were similar to Gaussian, but took too much processing time.

Out of the 4 smoothing functions, as can be seen Gaussian offered best results. Average function was a bit faster than Gaussian, but when applied to the video causing a lot of image jumping.

Maybe for the challenge problem exploring a smoothing function that removes a lot of noise on the road caused by shadows, road condition etc. will be worthwhile.

Results:

With a straight forward linear extrapolation, I got the following results:

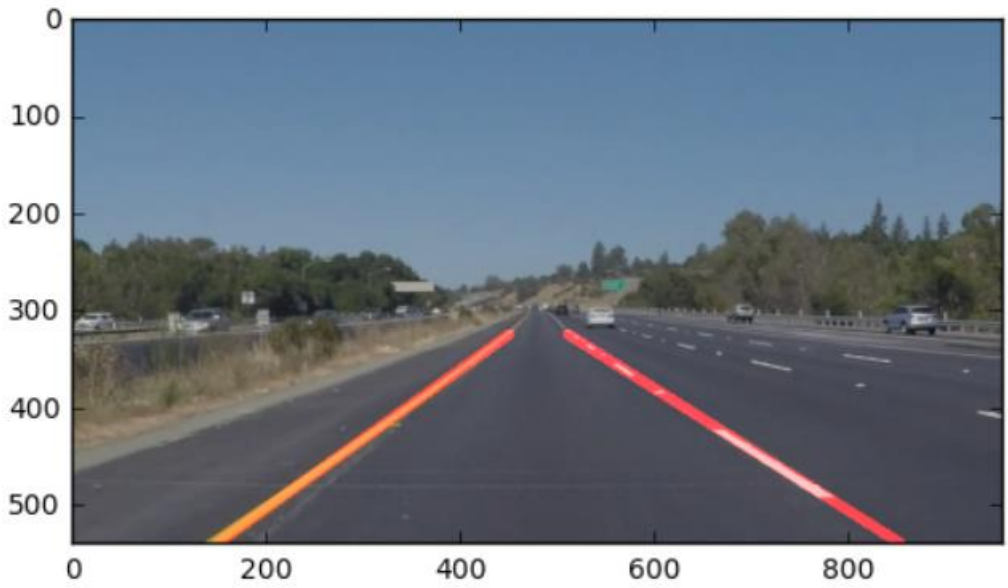

Figure 7: Final result on static image

Areas of improvement:

There is a lot of scope for improvement. I think if the raw image is clean and the noise is removed, a lot of this process becomes much easier.

1. Improving the quality of raw image. Based on my readings, usually the images generated by the camera have an inherent distortion to them and there are some calibration functions available in opencv to correct them. (Ref1). Would like to explore if this would make a difference.
2. I would like to understand the mathematics behind smoothing and edge detection algorithms so as to manipulate or choose a better algorithm that finds the edges (of only the lanes) clearly and reliably.
3. Instead of dealing with gray scaling RGB images, I would also like to explore if better results are obtained with other formats of images like - HSV. Is there any advantage/difference in the color formats between the different formats (Ref 2)
4. In my current computation, I have computed slope for each frame. This is not the right way of computing as I will have to apply a tolerance to the slope and intercept parameters and allow them to change between frames only if there are within the tolerance. This will help to correct some sudden distortions seen in video #2 due to the presence of lines other than the lanes found on the road. I would like to convert the code to classes so I can store the slope and intercept values for each frame and compare them easily. I did not want to implement a static variable computation in the current code.

References:

1. https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html
2. https://en.wikipedia.org/wiki/HSL_and_HSV