Autonomous Driving Project # 1- Lane Detection

Himanshu Kumar

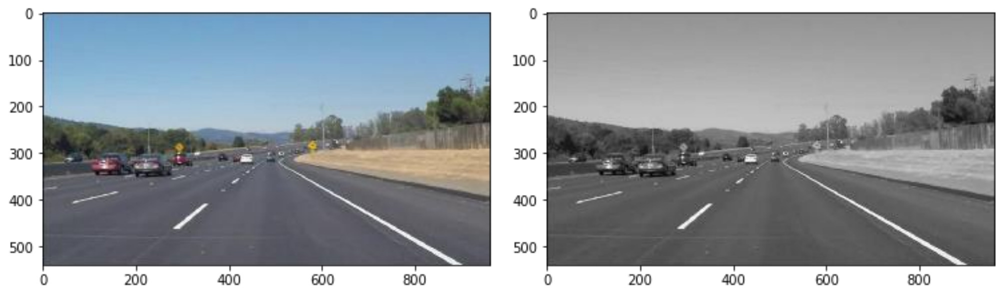
Project Definition:

This project has following requirements:
1. Detect lanes in images, and annotate image with lane marking
2. Detect lanes in video as an input and output video should have annotated lanes marking
3. Use helper functions to identify lanes lines
4. Detected line segments should be filtered/averaged/extrapolated to map out full extent of the left and right lane boundaries
5. Describe the implemented pipeline and future enhancements

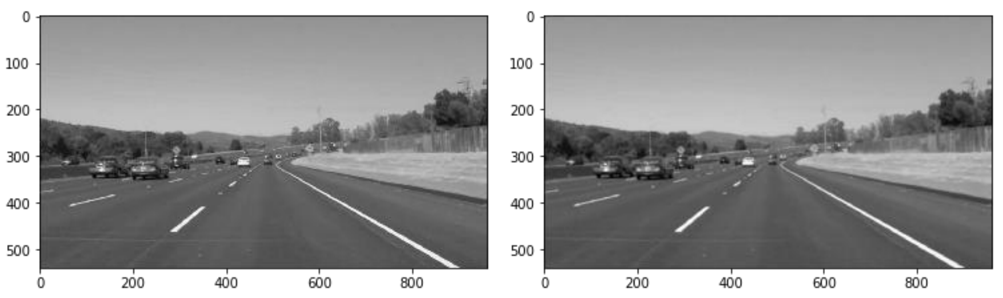
Lane Detection Pipeline:

The implemented pipeline has 6 steps:

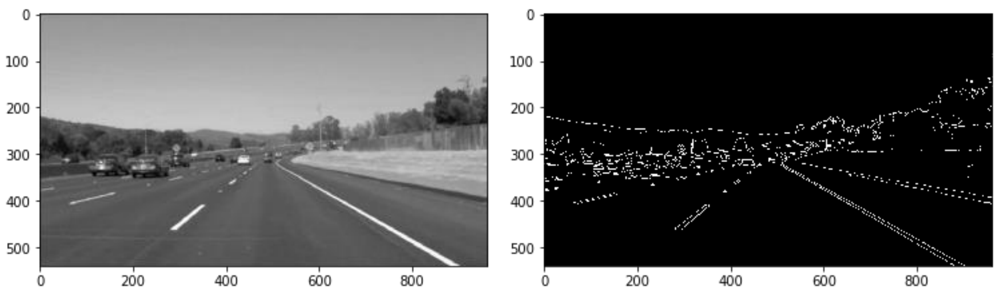
1. Convert RGB image to GRAY image using helper function "*grayscale(image)*"



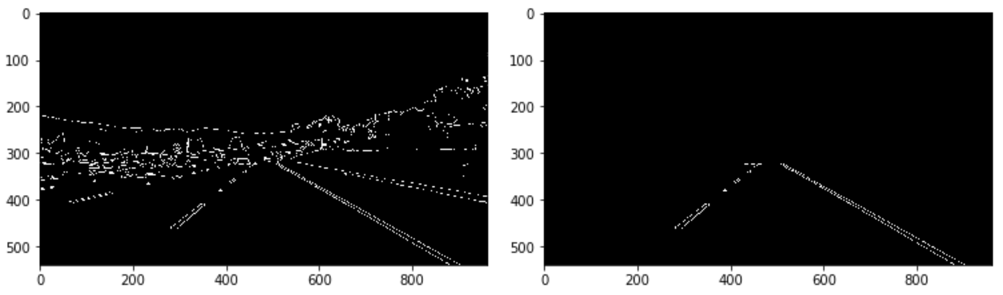
2. Apply Gaussian Blur on Gray image to reduce noise (smooth image) using helper function "*gaussian_blur(gray, kernel_size)*". The chosen kernel size is 5.



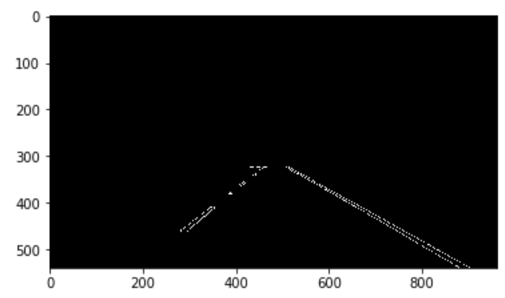
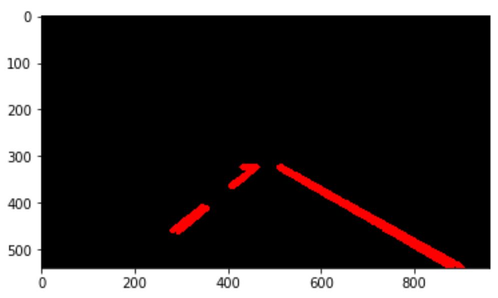
3. Apply Canny Edge detector in Gaussian blurred image (smoothed image) with minimum threshold of 50 and maximum threshold of 100.



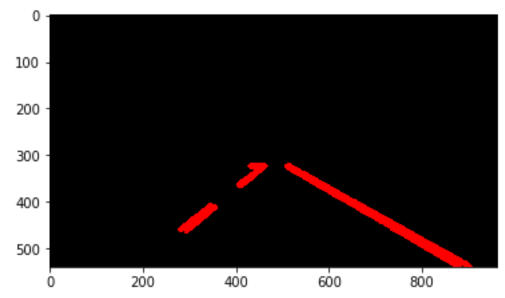
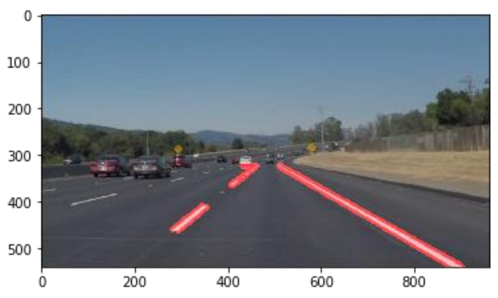
4. Select region of interest using parameter tuning. And mask the edge detected image using region of interest to filter unwanted image parts.



5. Then output of previous state is feed for Hough Transform using helper function "*hough_lines(…)".* Tuned the parameter for Hough Transform. The output will be lines drawn on masked image.

6. Apply weight image helper function for putting lane detected image on original image.

Modifying Draw Line Function

This was bit tricky to join segment lane lines. For this, first separate lines (output of lines after Hough Transform) into left lane lines and right lane lines. To do this, I used slope of line. If slope was negative then line is LEFT lane otherwise for positive (or 0 slope") its RIGHT lane.

While classifying lines into left or right lane, there were few bad points. To filer these bad points, I used X_CENTER of region of interest. If X coordinate of points does not meet these requirements then the point will be discarded for extrapolation:

For Right lane:
If x1 & x2 < x_center, then discard it

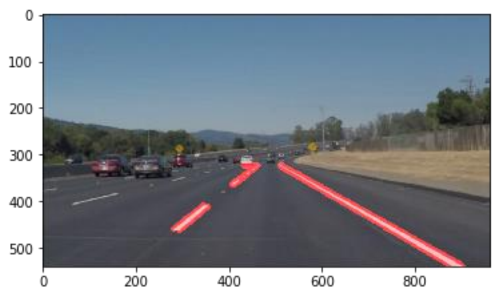
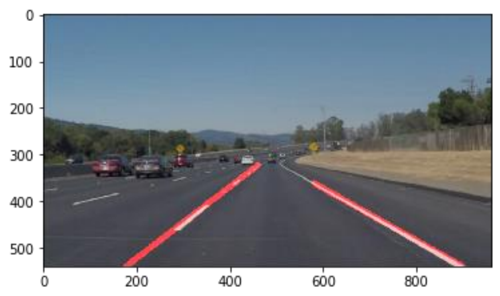
For left lane:
If x1 & x2 > x_center, then discard it

Now after filtered points for each lane, calculate average slope with highest weight for longest line segment. Also use intercept for the longest line segment.

To calculate X coordinate for bottom of image (left/right lane), I used the assumption that lane always starts at BOTTOM of the image. So We already have Y coordinate (image.shape[0]). With calculate average slope (for left and right lane), intercept and known Y coordinate, we can find X using

X = (Y – C)/M , where C is intercept, M is average slope , X,Y are coordinate of bottom points

For top points, I use region of interest top points.

Shortcoming of implementation:

There are few caveats of the implementation:
1. Draw line does not work for some image. This may be due to not finding X,Y coordinate of the points correctly.
2. This pipeline might not work on new image set as I have made some assumptions on image size and lane position (for region of interest).
3. The parameters for different filter might not have same performance on different image set.

Future Enhancements:

1. I think we can get rid of region of interest by using color selection on image. Maybe we can apply yellow and white color mask (lanes are either yellow or white) on image to find out lane.
2. I think, we can further improve draw line function. We need to investigate how to get better line draw.
3. This implementation is heavily dependent on using correct value for all parameter. This make it vulnerable to new images. We should design our pipeline in such a way that parameter value chosen should work on all image set.