Finding Lane Lines

Seymur Dadashov

July 25, 2020

Abstract

The purpose of the project is to become familiar with the OpenCV library and development of a pipeline that can be used to recognize lane lines of general highway road images and videos.

1 Task 1 Pipeline Procedure

The pipeline implementation followed a similar flow as was discussed throughout the course lessons. The image was initially converted to gray-scale from RGB using `cv2.cvtColor()` and the parameter responsible for the conversion is `cv2.COLOR_RGB2GRAY`. Since we used `mpimg.imread()` the image was already in RGB format, since `cv2.imread()` would result in BGR image import.

Following the gray-scale conversion, a Gaussian Blur was passed over the image with a kernel size of 3. The kernel size was 3 rather than something such as 5 or higher mainly due to the size of the dashed white lines appearing smaller around the center of the frame and I did not want to further distort the quality of the area, only suppress the noisy aspects of the image.

The `cv2.Canny()` function was used to scan the image for areas of rapidly changing pixel intensity, most likely using the $Sobel_x$ and $Sobel_y$ operators. A low-threshold was set to 50 and high-threshold set to 150 with the 1:2 or 1:3 ratio between low and high in mind.

The image was then cropped for the Region of Interest (ROI) using `cv2.fillPoly()` in combination with `cv2.bitwise_and()` to set everything outside the region to black.

The cropped image was then used as the input into the `cv2.HoughLinesP()` function in order to extract the lines from the identified edges. The parameters were the tricky to determine as I had to go through a lot of visualization to see what I was doing wrong. The ρ and θ parameters were kept as default with $\rho = 1$ and $\theta = \pi/180$. The `threshold` was experimentally determined at set to 25. The `min_line_length` was set in such a way that could detect the closest dashed lane line in a single segment as the image depth made it appear the longest, and this resulted in a value of approximately 40 that produced stable results. `min_line_length` in combination with `max_line_gap` set to 25 allowed me to link the smaller dashed line segments that appeared farther down along the road. My conclusion was that the farther segments should be grouped rather than setting my parameters so small that each dashed line received its own unique line as this produced too many slope and intercept values for further calculation. In addition, setting the parameter of `max_line_gap` too small resulted in a blinking phenomenon which is most likely due to the Hough Transform trying to link these smaller segments. Figure 1 displays the outcome of the described parameters prior to extrapolation of a single line.

1.1 Modifying the `draw_lines()` Function

The `draw_lines` function was modified with a simple approach involving averaging the slope and intercepts that were generated from each line out of the `cv2.HoughLinesP()` function. The left and right lines were generated using these average values and the sign of the slope was used to filter whether a line belonged to the left side or the right side as a positive slope corresponded to the left lane and a negative slope corresponded to the right lane. Figure 2 shows the results of averaging and extrapolation.

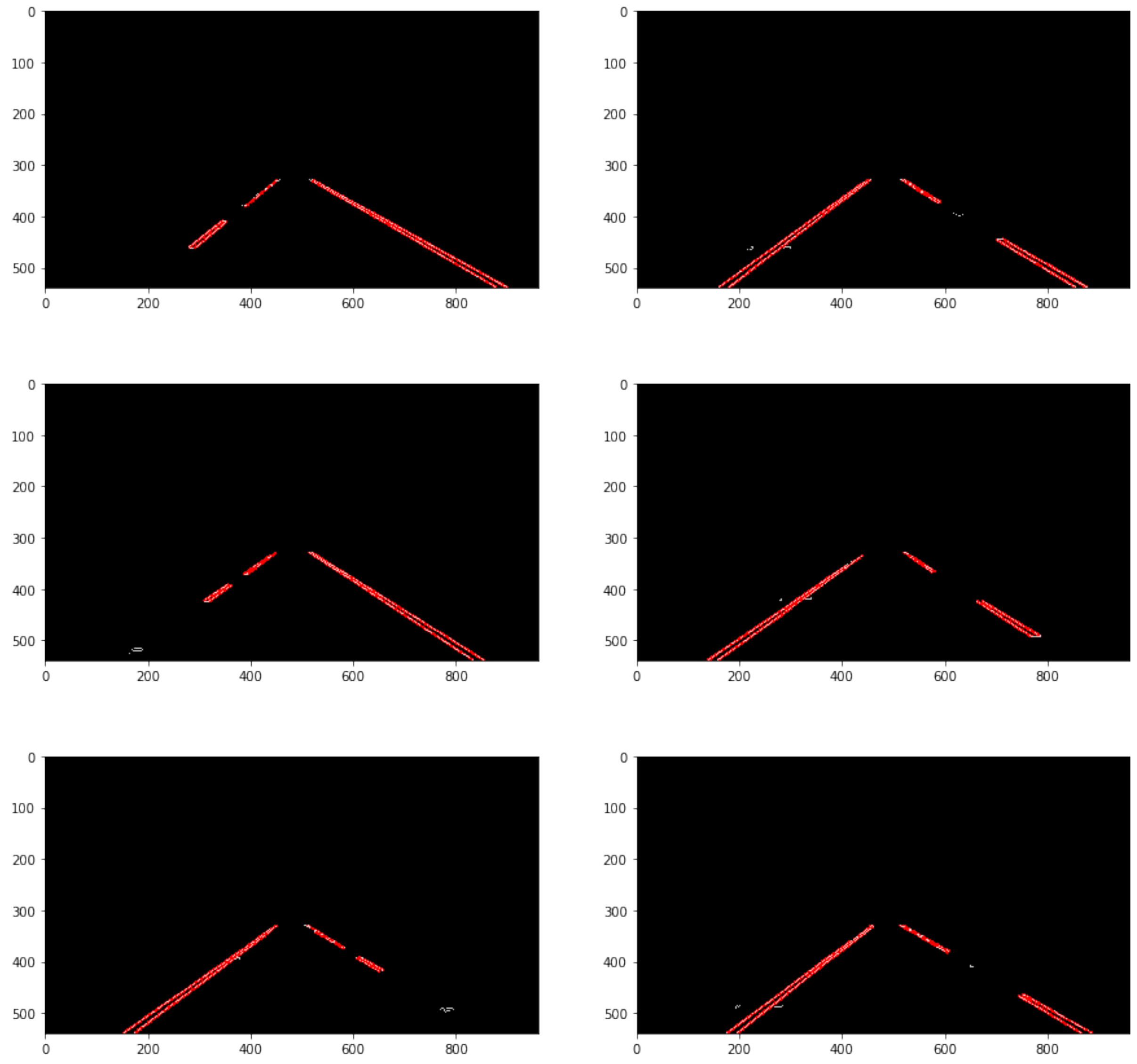
Figure 1: Hough Line detection on Canny Edge filtered and ROI cropped image

2 Task 2 Potential Shortcomings with Current Pipeline

I believe there are a lot of shortcomings in this pipeline and the list could be of exponential size, but for the sake of this particular scenario an obvious one from my implementation is that the lines generated does not appear *confident* in its prediction because of small fluctuations especially along the dashed lanes, not so much the solid lanes. I believe this is due to the parameters of the Hough Lines function not being tuned as much as they can be as well as the approach for extrapolating the line could be further improved such as tuning the outlier filter for the slopes. The approximation itself is line and thus linear, which does not accurately describe the behavior of the road that can bend, compress and appear distorted in image space both due to the depth aspect and lens distortion. Lighting conditions were also not accounted for as I am still not sure about how to approach this issue, but I would try transforming the image into HSV space rather than gray-space and indirectly monitor the pixel color based on the ”value” param of Hue-Saturation-Value. The list can probably go on for a while as these are just some of the obvious issues at hand.

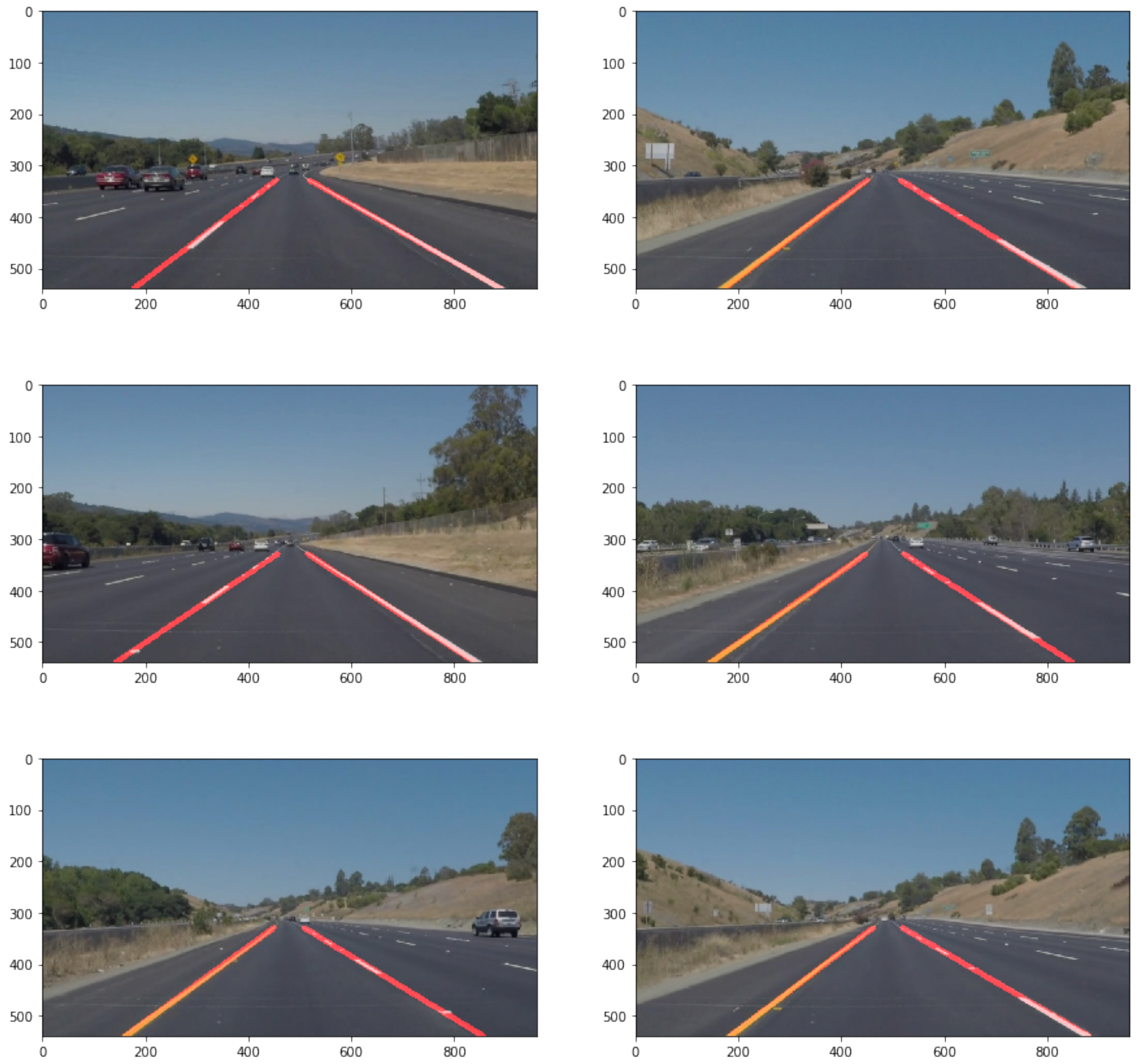

Figure 2: Extrapolated lines after averaging

3 Task 3 Possible Improvements to Pipeline

As mentioned above in Task 2, increasing the complexity of the line that is modeled from a linear one to a higher order polynomial would improve the fitting to different shapes of the road. To improve dealing with different lighting conditions, one can work in the HSV space rather than a single color channel of gray-scale.