Written By: Tyler Lewis
Date: 26 November 2017
Project: 1_Finding Lane Lines on the Road

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. Pipeline Description

My pipeline consisted of 9 steps. A description of each step in the process is given below along with example images that show the result of each step:

1.) Image files are loaded from a folder. The pipeline is designed to test batches of images as opposed to one image at a time. Steps 2-9 are applied to each of the 7 images shown below.

2.) The original image is converted to grayscale.

3.) The grayscale image is blurred. (This step helps reduce the amount of noise detected in the next step.)

4.) The Canny Edge Detection algorithm was used to detect edges in the blurred grayscale image.

5.) The Canny Edges are blurred. This helps "smooth out" the edges found in the previous step and bridges small gaps between pixels.

6.) A region mask is created. A trapezoidal region is "cut out" of the blurred Canny Edges image and passed on to the next step. Edges that were detected outside of the region of interest are discarded.

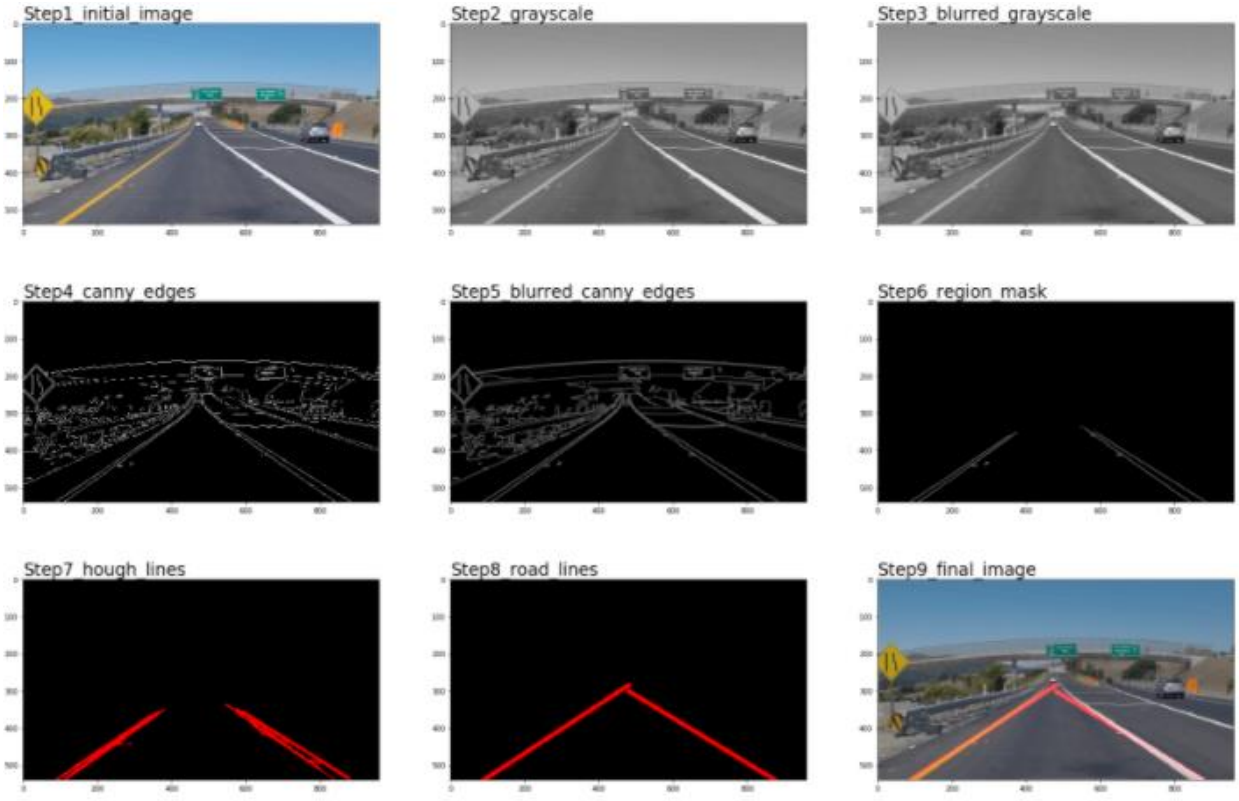
7.) The Hough Lines transformation is applied to all the pixels in the region mask image. The Hough Lines function determines, from a set of points, whether any of the points lie in a straight line. The lines that are detected using the Hough Lines function are shown as several thin red lines.

Written By: Tyler Lewis
Date: 26 November 2017
Project: 1_Finding Lane Lines on the Road

8.) The Hough Lines are grouped into two groups, Left Lines and Right Lines, based on the slope of each line. A "line slope filter" is used to filter out lines whose slopes are either too steep or too shallow. The remaining Left Lines are averaged by slope and y-intercept to form a single idealized "average Left Line". The remaining Right Lines are averaged by slope and y-intercept to form a single idealized "average Right Line". The two lines are combined to form a Road Lines image.

9.) The Road Lines image is combined with the Original Image to form a weighted image.

The images formed in Steps 1-9 are shown below:



2. Potential Shortcomings with my Pipeline

One potential shortcoming with my pipeline would be losing the lane line from the region of interest. As it is now, if the pipeline can't detect either a left or right final lane line, the final result is just the original image. In video testing, it is easy to tell when the pipeline has failed to determine at least one of the lane lines. (When the red lines "flicker", the pipeline has failed for that frame.) Most of the failures occur when the lane line in the video stream drifts outside of the region of interest.

Another potential shortcoming is that the red lines "jump around" too much in the final video and appear shaky.

3. Possible Improvements with my Pipeline

A possible improvement to the pipeline would be to apply some sort of "smoothing" function to the two idealized red lines from frame to frame in the output video. Rather than "jumping around" from frame to frame, the lines should "move slowly" over the course of several frames. Also, to get rid of the "flickering" effect that occurs when the red lines disappear for a frame because no new lines were detected during that frame, the lines from the previous frame could be re-used.

Another possible improvement would be to add "analytics" to the side of the output function. Statistics showing the "strength" of each line could be shown. It would be nice, for testing purposes, to be able to pause the output video and view statistics such as: Number of Hough Lines detected, Minimum and Maximum Hough Lines lengths, number of Left and Right Lines detected, etc.