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 and Reflect on your work in a written report

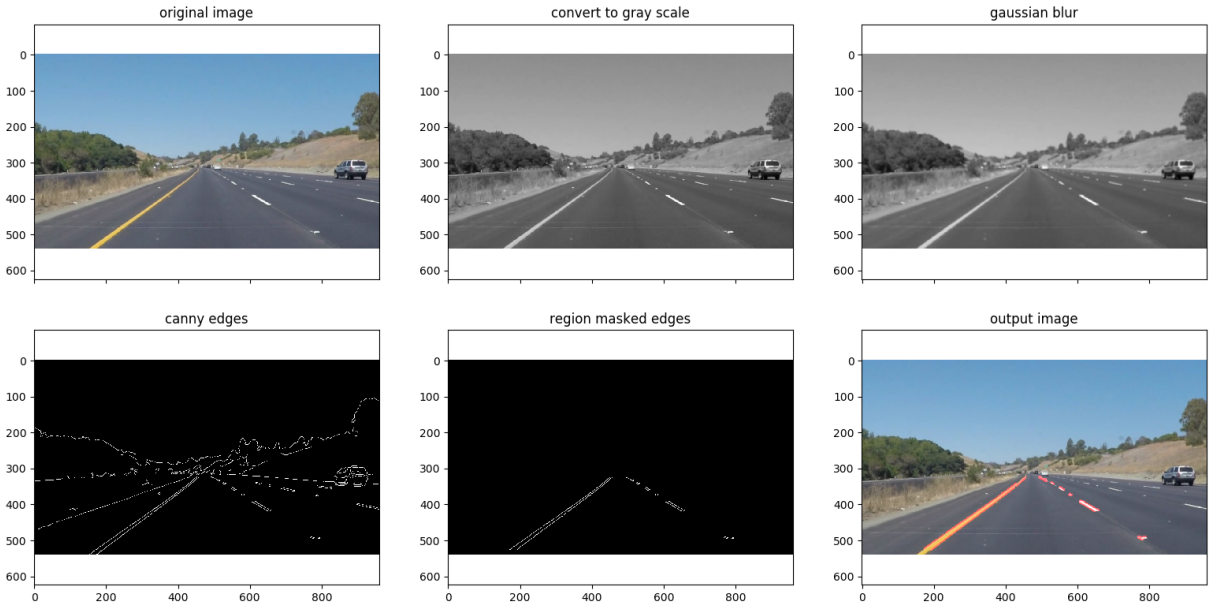
Reflection

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

1.1 Pipeline: As learned in this course, I create my pipeline with the following 6 steps:

1) Convert the original color image into a gray scale image.
2) Apply a Gaussian blur to the gray scale image and get a smoothed gray scale image.
3) Apply Canny edge detection to the smoothed gray scale image and get edges.
4) Define a region of interest around the area of the lane in front of the car, and apply this region mask to those edges detected by the Canny edge detection to get edges of interest.
5) Apply Hough Transformation to those edges got in step 4, and get all line segments of interest.
6) Blend all these segments of interest to the original image and get the final output color image.

Take an input image as an example, the result achieved by each step can be shown as following:



1.2 Modifications to the draw_lines() function

To extrapolate all detected line segments into the left and right lanes, I tried the following steps:

1) Classify all detected line segments into two parts, one for the left lane and one for the right lane.

   Since in image space, (0,0) in at the top left corner, majority of the line segments belonging to the left lane should belong to the left part of the image and have a negative slope, similarly, majority

of the line segments belonging to the right lane should belong to the right part of the image and have a positive slope.

In my implementation, I check through all the line segments, if a line segment has slope < -0.5 and its two end points' x coordinate are both in the left half of the image, I classify it to be part of the left lane. Similarly, if a line segment's slop >= 0.5 and its two end points' x coordinate are both in the right half of the image, I classify it to be part of the right lane.

2) Extrapolate all line segments in the left and right lane, and get the final left and right lane line.

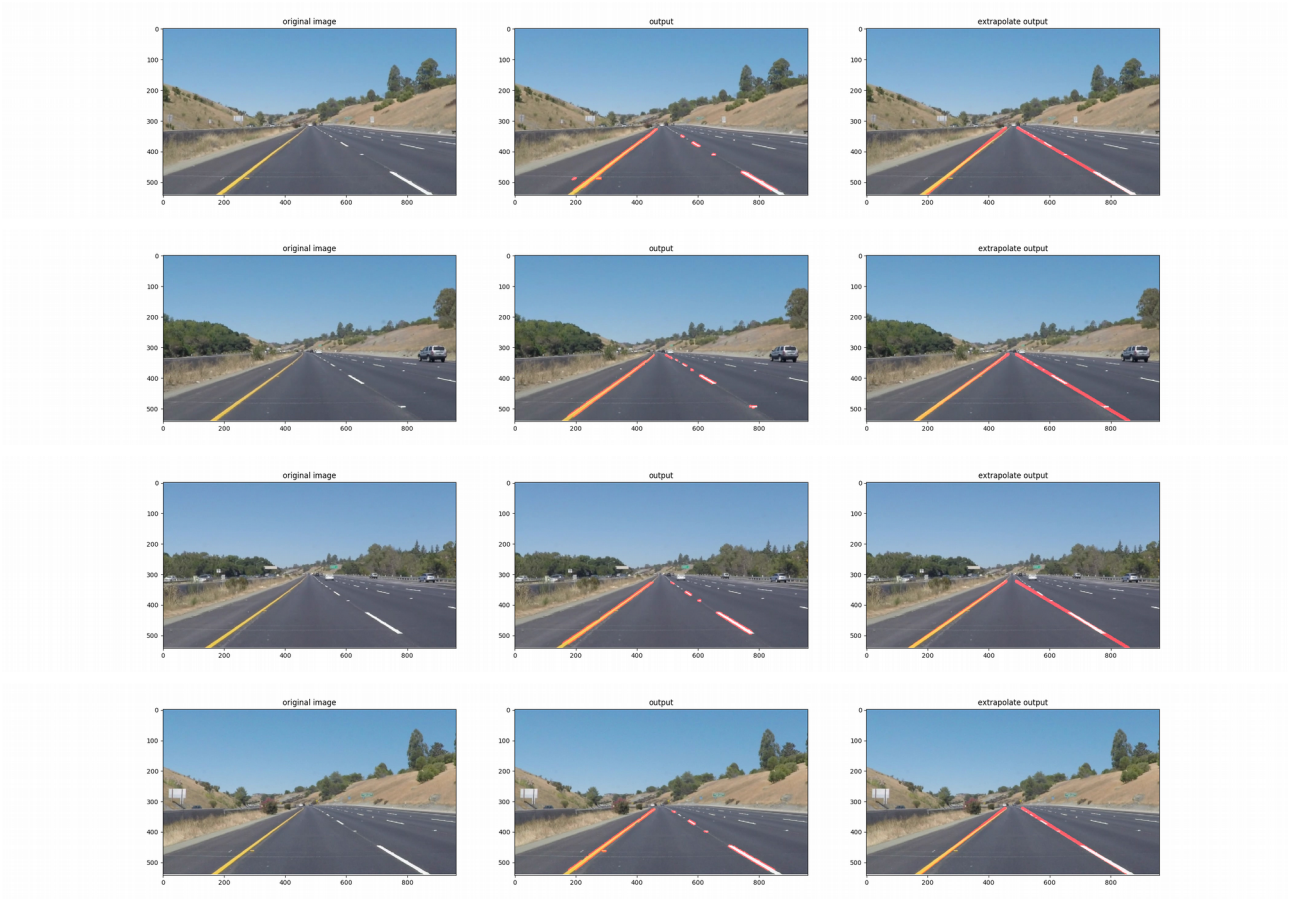
   Take the left line for example, after getting all line segments for the left lane, we need to extrapolate to get a line which can best fit all line segments. To get this line, we just need to get its slope and some point on it.

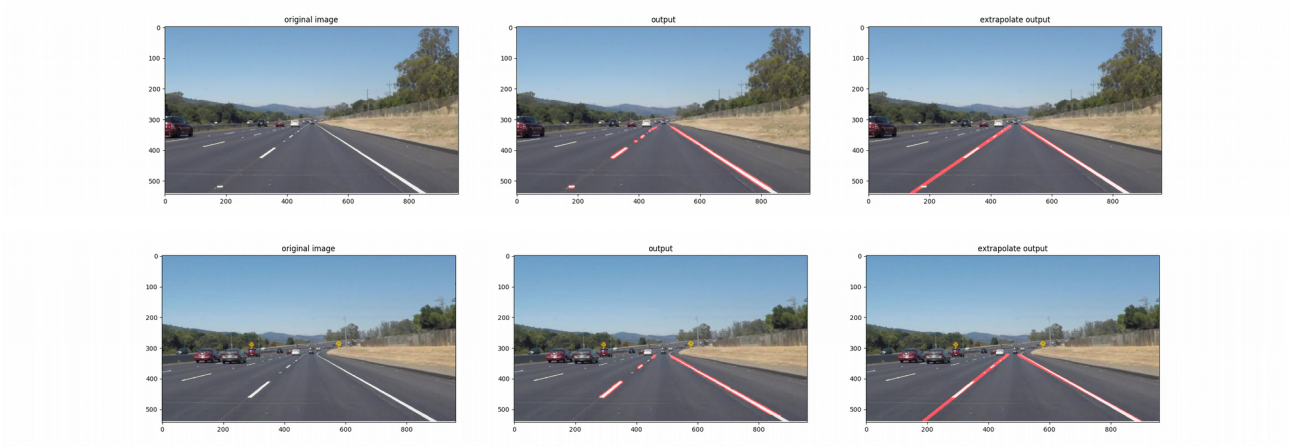
   In my implementation, I use a weighted average over all line segments' slope to get the final slope. The weight for each line segment is the length of the line segment. Similarly, to get a point on the line, I choose to use the middle point of all line segments, and do a weighted average over all middle points. The weight is also the length of the line segment.

3) Stabilize the result in video output

   Due to the implementation, the final line got at each frame sometimes varies a bit, and this causes the video output flickering a lot. To stabilize the output, I used a buffer to store the latest 10 frames' lane result, the detected left and right lane slope and middle point. And the result of current frame is an average over the latest 20 results, including the current one. This makes the final video output very stable.

1.3 Results





By applying the pipeline and extrapolate the line segments, I have got the results for the test images as shown above. The left image is the original image, the middle image is the one without extrapolating and the right image is the one after applying extrapolation.

2. Identify potential shortcomings with your current pipeline

My current pipeline has the following shortcomings:

- Parameters chosen for canny edge detection and hough transform are hard coded and tuned based on testing image. If we change the input images, we probably have to tune that again.
- Interested area is fixed by a polygon, which requires adjustment when the camera is changed or when car is driving not in the middle of the lane.
- Extrapolating of line segments has the assumption that the detected lanes are straight line. In case the lanes are not straight lines, like a curve for turning, it does not work.
- Tuning of the parameters. Due to a limited time, I tuned the parameter close to optimal result, but still a bit distance from the optimal result.
- Line segment outliers are not filtered out when extrapolating line segments to final lane lines.
- Speed and memory wise, the code is for better readability and in Python, it is not that fast and some memory cost can be reduced.

3. Suggest possible improvements to your pipeline

Possible improvements to the pipeline are as following:

- Research and try adaptive methods to chose parameters for canny edge detection and hough transform. So the pipeline can adjust itself when feed with different testing cases.
- Adapt the interested area based on input. Preprocess the image and find the major lane in the image first, and then try to build a region based on that result.
- Try different methods to extrapolating of line segments when the lane is not a straight line.
- Spend more time tuning the parameters to get better result.
- Use statistic method to exclude line segment outliers when extrapolating line segments to final lane.
- Optimize the code for memory and also increase the speed. Maybe change to implement in C++.