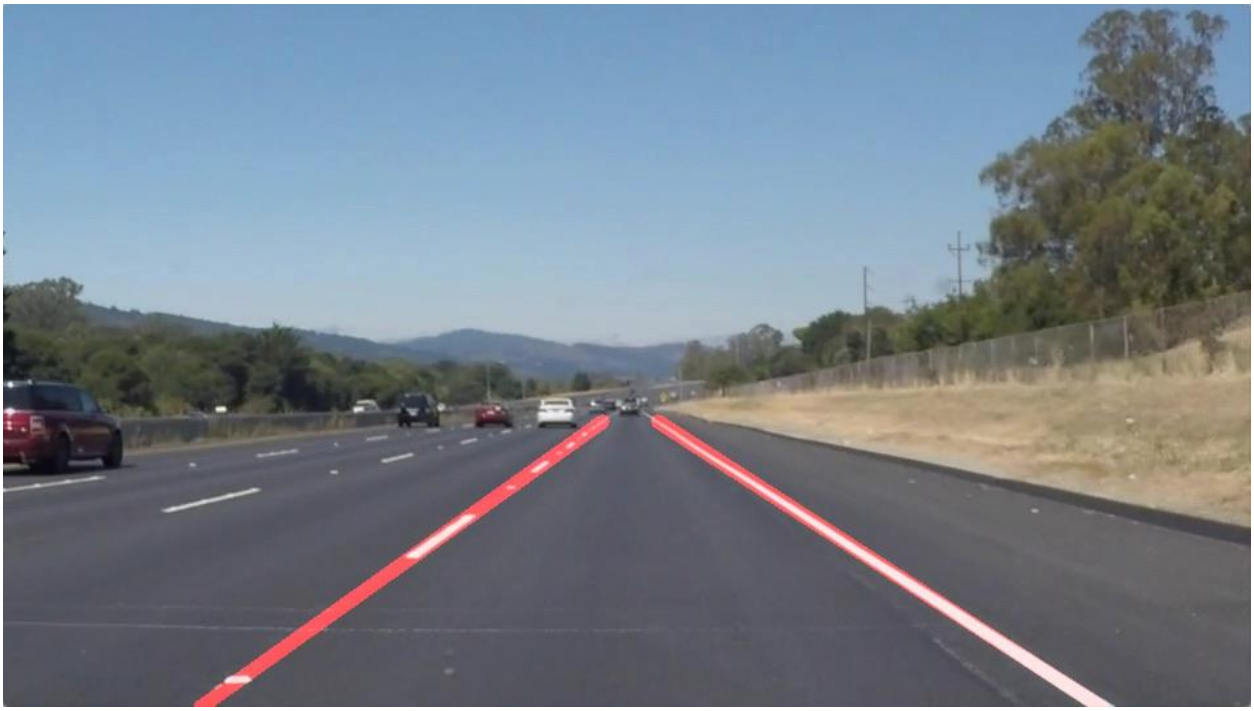
Finding Lane Lines on the Road

The goals / steps of this project are the following:

* Make a pipeline that finds lane lines on the road

* Draw the lane on the image that was captured on a self-driving car camera like the following:



Design:

The overall design is basically what was being described in the Udacity Term 1 lane detection material. And the pipeline uses all the helper functions that was provided with some modifications. We do gray scale on the image first so that instead of 3 channels color (RGB), we only have black and white. After that we apply gaussian filtering with a kernel size of 5. This value is what is recommended in the OpenCV documentation. Need we do canny transformation and apply region of interest helper functions. Finally, we do Hough lines algorithm and combine the lines output with the original image.

I am not going to describe what parameters I used for each functions/algorithm about since they are all defined in my pipeline function (DrawLaneOnImage). In fact, most of the values I used are directly from the Term one lane detection lecture material.

There are two functions that were modified from the original helper functions, the gaussian_blur and hough_line function. The gaussian_blur, I just set a default kernel size to 5 since this is what is recommended in OpenCV decoumentation

The interesting part is the hough_line function. This function calls the OpenCV API, cv2.HoughLineP. This function returns all the straight lines even the one that we do not want. So, I create an inner function (Thanks for the python feature to allow create inner function) called filterAndProcessLines. What I do is remove all the lines that are too flat (lines with slop that are close to zero) or lines that are too deep (line with slop that are too large). Currently, I remove all the line with slop less than 0.4 (absolute value of the slop of the line) or greater than 3.

After removing all the lines that are not quality as lanes, We need to connect all the lines that has same "*m*" and "*b*" since lane may defined as dash line on a road. The formula of a line equation is y=mx+b. If on an image, we detect dash line for the lane, they have to have the same m and b. Therefore, I connect them together.

Next, we know there must be two lines detected for a lane, one on the left and one on the right. And therefore, I categorize lines with either positive slop and negative slop. And for each group of the lines, I find the longest possible line. Lastly, I have two lines left, one is the longest possible line with positive slop and the other one the longest possible negative slop.

Finally, we made both lines the same length by calculate the minimum and maximum "y" value of the endpoint on both line and the extend the shorter line to minimum and maximum "y" values that I just calculated.

Potential shortcomings of the current pipeline

One of the potential weakness I can see in this implementation is the lane that was detected maybe slightly off align from the actual lane. This really depended on how well the input parameters were tunes when calling the cv2.HoughLineP API as well as the selection of the lines that my inner function "filterAndProcessLines" were used to merge lines with close "*m*" and "*b*" values. This generally does not matter in my view because perfectly detecting a lane doesn't mean the self-driving care will drive safety. The self-driving car design need to account for lane detection error. As long as this error is within certain margin, I think we should be safe. However, look like the error margin is too big for Optional Challenge Video in the Project.

Another potential issue is the lane that was detected, the two lines may cross each other.

Suggest possible improvements

One improvement that can be made easily is to remove the potential that both lines cross each other. This can be done by calculate the intersection of two lines and cut both lines to the points where they are intersected. Again, formula of a line is y=mx+b, and we can easily use the formula to figure out the point of intersection.

Second improvement is to improve the alignment of the lane line that was detected. That can be done by either further fine tune the input parameters for all the helper functions or modify the algorithm in the “filterAndProcessLines” function that merge lines with close “*m*” and “*b*” values. One suggestion is group all the lines with close “m” and “b”, find the longest line within the group, and merge other shorter lines into the longest one.