SDC Term 1: Lane finding
REFLECTION
Spencer Kelly

1. The Pipeline

My pipeline consisted of seven main steps. First, I added gaussian blur to the image to smooth it, then used canny edge detection to find the strongest gradients in the image. I chose to use 150 as my high threshold and 50 as low after experimenting with many different values and seeing that those also work in the image in the lesson. I then used the image shape to determine the vertices for the mask. I had originally used pixel values, but that failed when used on the challenge video because of its different shape. I then applied the region of interest function to narrow the picture down to the specified segment. Hough lines was applied after that to determine and draw the lane lines. And finally weighted image overlayed those lines onto the original image.

The draw function was a real SOB though. First I had to separate the lines into three categories. Left lane, right lane, or neither. I did this by first discarding all lines with small gradients. This risks discarding lane lines on sharp curves but it's the best way I knew how at this point. Then I separated the lines into left or right lane based on gradient and added them up but keeping track of how many were in either pile. I then took the average of each of those lines.

After I had it down to two (relatively) accurate lane lines I had to extend them up and down to keep all the lines the same length. So I took the slope of both and used this to calculate new end points for the lines. This presented a problem in the challenge video though because sometimes one or both of the lines wouldn't exist and so would have a slope of 0 (the default value). So for images that only had one line I simply made the other line it's inverse, assuming that the car was in the relative center of the lane. And I skipped drawing lines on images that had neither lane lines.

2. Shortcomings of my pipeline

There are many. A static mask makes it very hard to exclude all confusing lines all the time. It stops recognizing lanes on the bridge. It can only draw straight lines. And trying to compensate for not having one lane line by using the inverse of the other is plain stupid. It was fun to do and made the video look better but it wouldn't work on the road. It gets messed up if the car isn't in the center of the lane, or is on a curve. And the fact that there are frames where neither lines is seen by the program is concerning as well.

3. Possible improvements

Dynamically shifting the mask based on a trend in the change of the top of the lane could help with curves especially. You could also use some sort of quickly degrading running average of where the lane was and rely on it if you don't have any lines recognized in the photo. This would also help you determine if the line being drawn was wrong as any dramatic change is probably the fault of computer error since lanes just don't do that between two frames. The parameters I picked are definitely not the best. I am not sure what some of them really mean and don't know the optimal values for the ones I do know. And finally, my draw function just looks messy. I could definitely use a few more helper functions instead of writing everything within the draw_lines function.