Reflection – Brent Wylie

Pipeline description

My pipeline had the following stages, in consecutive order:

1. Image was converted to grayscale
2. Gaussian blur applied to grayscale image
3. Thresholds set and Canny Edge detection performed on grayscale image
4. Defined region of interest on Canny edge result
5. Fed masked region of interest into hough transform block which detected lines and drew them on the final image

I used the provided helper functions to create each stage, which was sufficient to draw lines on every detected edge. Some quick additional tuning was required at first for canny edge thresholds and hough transform parameters.

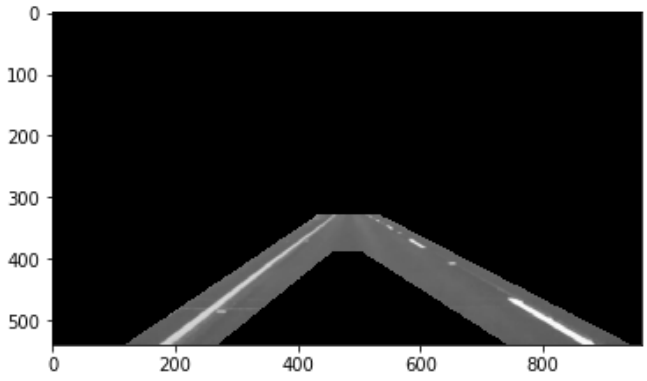
Later, after moving to video testing and moving to improve the draw a single line on the left and right lanes, I found that I had to modify draw_lines() and the region of interest bounding.

Specifically, I modified draw lines by binning lane candidate lines depending if they fell left or right of the center of the image. The region of interest masking prevented any significant collisions between the two. After binning each lane's candidate lines, I used a polynomial fit with degree 1, so a least-squares regression to find a best fit line that approximated the lane's position.

This was the bulk of the work, as managing the numpy data types and the inverted x and y axis took some time. Afterwards, I noticed there were periods of high disturbances to the lines and decided to filter lane candidates by approximate slope. I rejected all lane candidates that appeared to be mostly horizontal. This vastly improved stability and improved overall line accuracy.

Lastly, in the video testing there was one or two portions where erroneous paint was present in the middle of the lane. I modified the region of interest accordingly to have somewhat of an A shape.

Here is an example image of the improved region of interest:



Pipeline shortcomings

My pipeline performs relatively well on the two videos provided. It accurately tracks the lanes (better than some humans at least), with only a small amount of jitter between frames.

The jitter is one small shortcoming of the image pipeline which arises from the pipeline's lack of state. Each frame is analyzed from a clean slate with no influence from prior frames.

Another shortcoming is that since I used a linear regression, at best I will always draw straight lane lines. This will not work very well for sharper curves, and on intense changes it will not work.

Another shortcoming is that the pipeline does not have a dynamic region of interest, so, changes in the video angle (such as through a long turn) may negatively affect the lane prediction. This is not a problem on the test videos but would be present in sharper curves, which already fail due to the linear regression.

Lastly, the final shortcoming is that it is almost certainly unable to deal with varying conditions of pavement. The test videos were very uniform, but a change from asphalt to cement or old lane markings or tire streaks on bright cement would likely cause a loss of lane prediction.

Pipeline improvements

My pipeline works pretty well, but has a few areas that could be improved with enough engineering time.

Specifically, the small jitter problem could be improved by averaging the lane lines from frame to frame. This would greatly reduce any erratic and sharp changes overall. A private global variable, or static private member variable could be used to record state within each call to the process_image() function for this.

The issue with curves could be improved by modifying draw_lines() again to use a quadratic approximation of the lane line instead of a linear approximation. The calculation is not too difficult, though drawing the lines on the image later may be. The slope filtering mechanisms would have to be modified to account for this too.

The dynamic region of interest would be somewhat difficult to program, but it could function similarly to the lane line averaging from frame to frame. Some frame to frame history could be used to allow the region of interest to grow or shrink a small amount each frame as the car goes through the curve. This is probably the hardest improvement to do manually, and there are almost certainly better ways to fix performance on curves than to do this.

Lastly, more extensive color mapping could be used to improve lane detection on varying pavement. Specifically, the use of HSV color filtering could improve detection of yellow lines versus grayscale, as grayscale represents yellow lines as being dimmer than white lines. HSV shows the two colors closer together in terms of brightness. Additionally, selecting color components line Blue channel could be used to distinguish yellow lines on bright pavement.