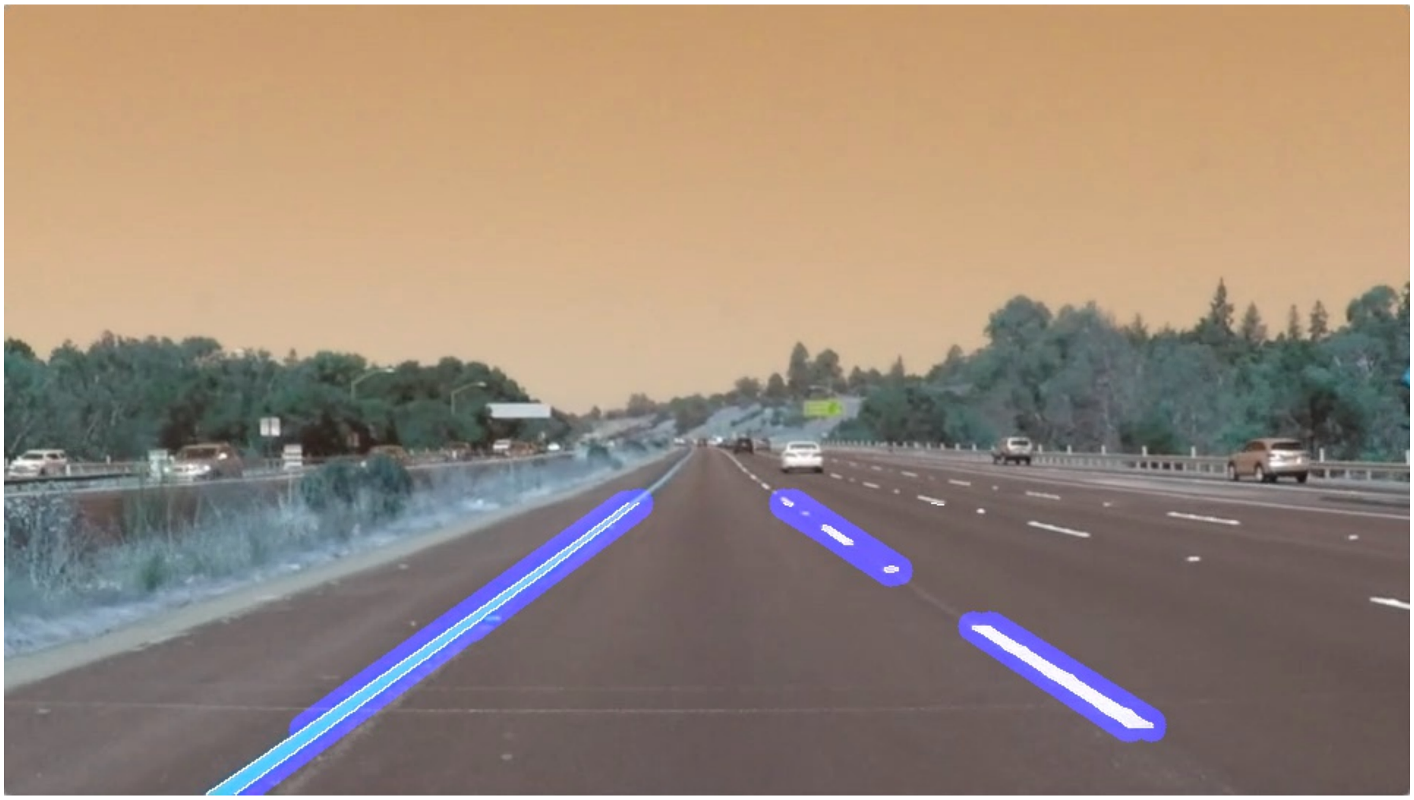
Summary

My pipeline extends the basic pipeline (that finds line segments in a selected region), and then modifies the `draw_lines` function to extrapolate these line segments into two line segments that have slopes in the range of slopes we might expect from lane lines.

Pipeline Description

1. Gaussian Blur with kernel_size=5 to blur the images. I do not convert the images to grayscale because of the shadow in the challenge output.
2. use `Canny(70, 210)` to identify edges
3. Filter to an isosceles triangle in the bottom 37% of the image, starting 33% in from the left, and ending 33% in from the right.
4. use `HoughLinesP( rho=2., theta=np.pi/360., min_line_length=60, max_line_gap=40,)` to find line segments that can be made from those edges. At this point, images look like:

5. Go from many segments to two lane line segments with modifications to `infer_and_draw_lines` a. Split into two groups of points (negative and positive slope)

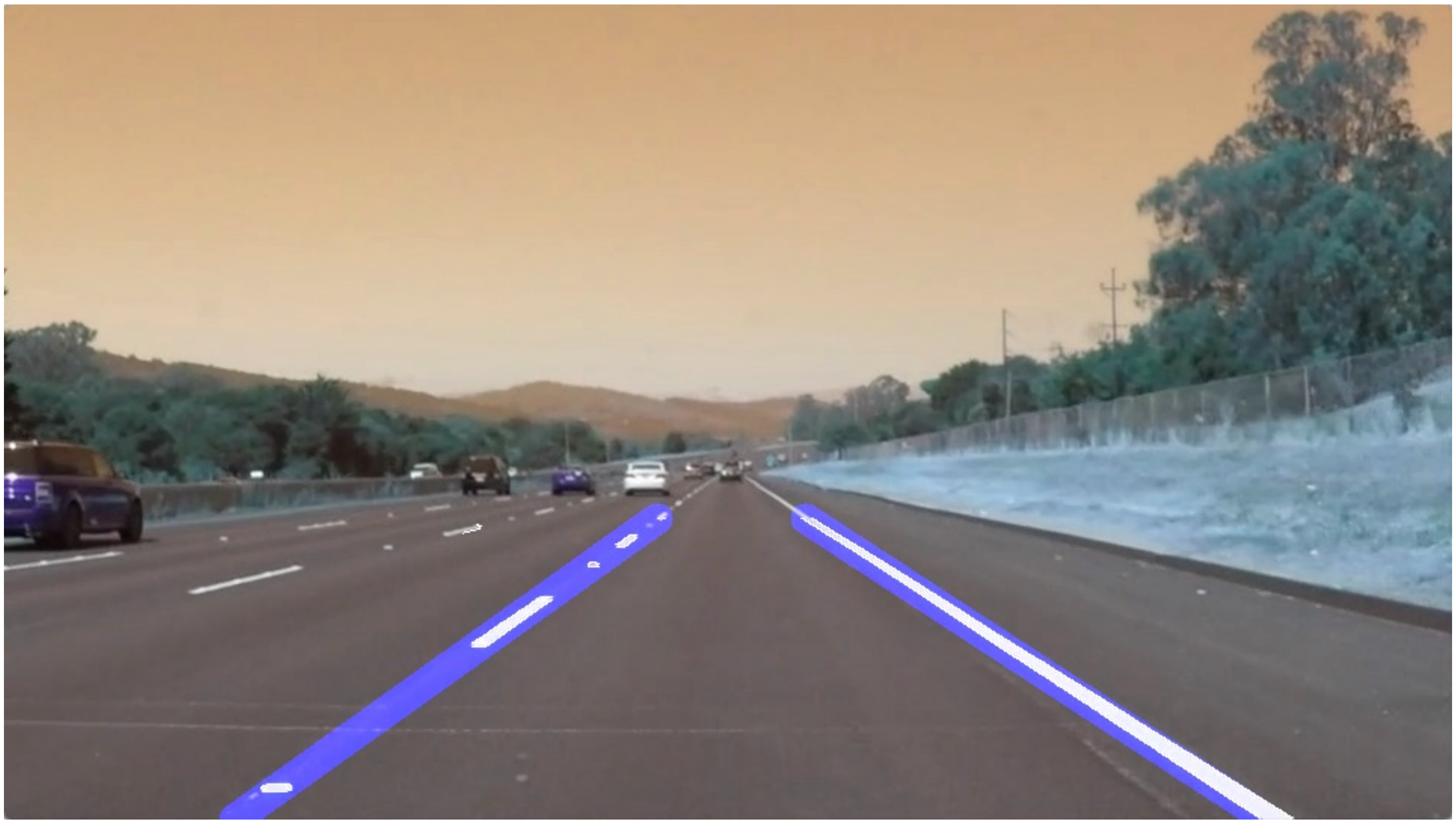
   b. for each group of points in [neg*group, pos*group]:

i. run a `HuberRegressor` on the (y, x) pairs in the group

ii. start a line at the the bottom of the image and use the fitted model to extrapolate what x would be

iii. end the line at the fitted prediction for the minimum y.

We end up with:



and the videos in the notebook.

`draw_lines` modifications with Huber Regression

Step 6 was the modification to `draw_lines` {Note that I chose to do the consolidation in the function in `infer_and_draw_lines`, since I didn't want to corrupt the function that was doing the annotation with other logic.}

In order to draw a single line on the left and right lanes, I modified the hough lines function to split the segments it sees into three groups, those with positive slope between .5 and 1 (right lane) and negative slope between(-.5 and -1) (left lane), and those with slope outside those regions (ignored). This allows us to filter the large number of line segments identified by the hough_lines procedure to those that are likely to be part of the lane line. (eg not the front of our car, not another car) Then we find the best line to fit the left lane segments and right lane segments, and then we connect that line from the bottom of the image to the top of our triangle

37% of the way up the image.

There were a few difficult parts in this part of the exercise. First reshaping the data from (x1,y1,x2, y2) format to (x,y) and then reshaping it back was annoying (especially when combined with reasoning about (0,0) being top left!).

The larger challenge, and the reason I couldn't use `LinearRegression` in my final solution, was the median/ boundary with the fields on the right. These are lines and edges with similar slopes to the lane lines, parts of which are contained in the top of our region of interest. None of my attempts at parameter tuning could prevent the inclusion of the left median in the annotation of the challenge video, so my only option was to make the consolidation procedure more resistant to outliers by optimizing for the median absolute error for some samples (Huber) rather than the mean squared error (OLS). According to the sklearn documentation,

> The Huber Regressor optimizes the squared loss for the samples where |(y - X'w) / sigmal < epsilon and the absolute loss for the samples (outliers) where |(y - X'w) / sigmal > epsilon

This way, even though my algorithm is incorrectly considering the line segments of the median as the same as the segments in the lane line, they do not have an outsized impact on the result.

Challenge

The challenge took a fair amount of time, but almost effort was wasted. All I had to do was to stop gray scaling. It is literally impossible to for a human to see the lane line when the road turns gray in grayscale, so it probably isn't that easy for `Canny` to find an edge.

Shortcomings

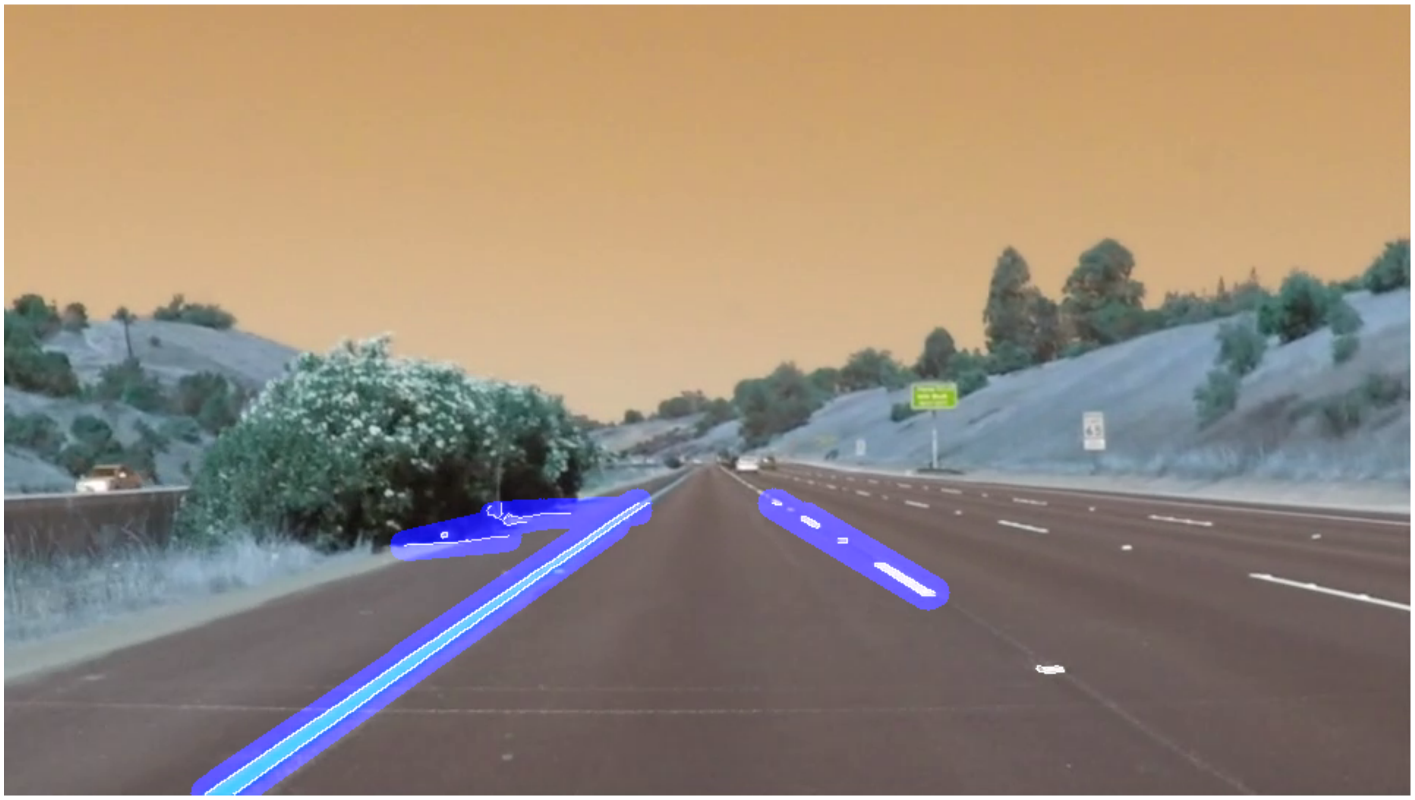
There are many shortcomings with the current pipeline:

1. It has only been tested on open high way with no cars in front of the dashcam, if the side of a car and a lane line are nearly parralel, my slope finder will get confused.

2. The hough_lines procedure often identifies the median as a useful line segment, which I combat by using Huber Regression the median. Still however, a better procedure would identify that the median is not the lane line.

3. The lanes flutter a bit, when there are mispaintings or changes in lighting.

4. in 'solidYellowLeft.mp4' the bottom of a bush is identified as a line



before consolidation. Luckily these points are underweighted by Huber and the extrapolated version looks fine.

Possible Improvements

1. Using information from where the lane line was at t-1 could help the algorithm anchor its prediction to the last prediction (with the understanding that the lane line probably hasn't moved very much in one frame), and help solve the flutter problem.

2. The biggest step needed to improve the pipeline is a labeled dataset of annotated images, so that if one were to change a parameter they could see whether they improved their solution without manually inspecting so much video. I could have used deviation from provided examples to score my submissions, for starters, to get closer to this. This would also allow a more automated search of the `Canny` and `HoughLinesP` parameters.