\V
March 30, 2017

Self-Driving Car Nanodegree
Writeup for Project 1: Finding Lane Lines on the Road

1. Describe your Pipeline

The pipeline `process_image` goes through several steps:

First, the image is translated into grayscale and modified using a Gaussian Blur in order to remove noise and smooth out pixel values.

Second, the image is processed using Canny Edge Detection which identifies gradients in neighboring pixel values in order to delineate where edges in an image may exist.

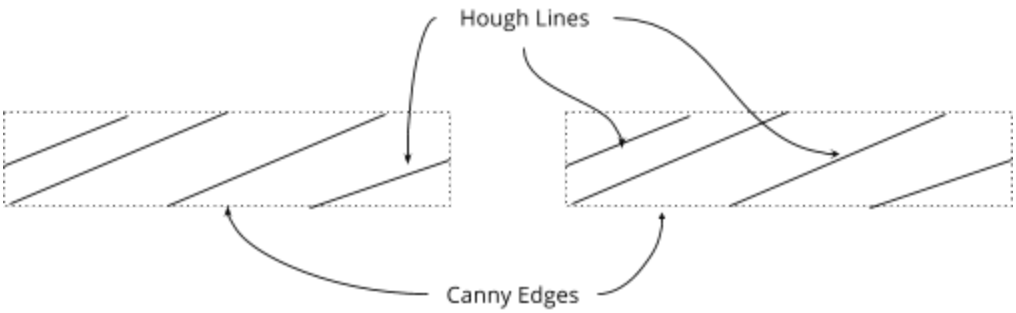
After Canny Edge Detection, the pipeline masks the image given a region of interest - the area within the image that the lanes are likely to exist. The points delineating the region of interest were identified manually. Fortunately, the camera placement, horizon and curvature of the road were rather consistent for all the test images and videos. The resultant image would retain pixel values output by Canny Edge Detection within the region of interest, and blank out the pixel values outside of it.

Canny Edge Detection gives positive values to individual pixels which seem to lie upon an edge in the image. Using the Hough Line Transform, pixels which seem to lie upon the same line are strung together to create a series of lines which represent edges in the region of interest. The idea was to approximate the lane markings visible in that area.

The Hough Transform returns a series of lines in the form `[[X11, Y11, X12, Y12],[X21, Y21, X22, Y22]...[XN1, YN1, XN2, YN2]]`. Instead of processing these lines within the function `draw_lines`, I created a new function called `process_lines` for this purpose. I left the `draw_lines` function for doing only and exactly that: drawing a given array of lines upon a black image.

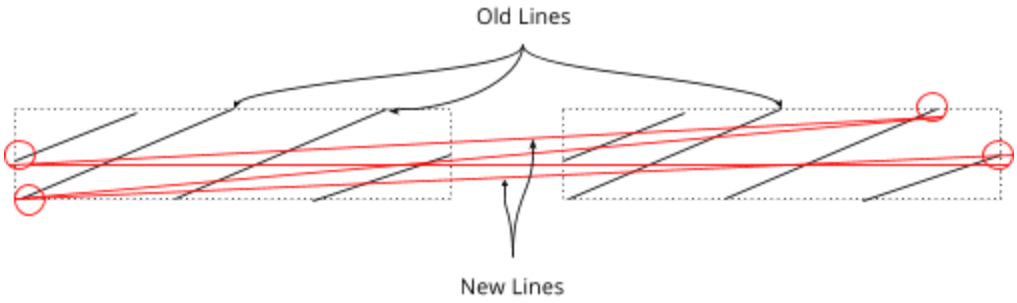
The first task performed within the `process_lines` function was to sort lines into those which demarcated the LHS (left-hand side) and RHS (right-hand side) lane markers.

The second task was to try and aggregate the lines on either side to yield one line which represented that sides' lane boundary. One of the main problems I encountered when trying to accurately identify lane boundaries was that lane markers are not lines, per se, but rather rectangles of paint. As a result, I would often get skewed lines identifying lane markers because the Hough Line Transform would delineate these rectangles of paint as such:



So while the actual lane boundary should run parallel to one of the lane markings' horizontal edges, averaging the lines returned by the Hough transform would yield a line slightly rotated relative to the lane's true direction.

To work around this difficulty, I took the extreme endpoints from the lines output by the Hough transform and connected *those* to create new lines which were more indicative of the lane's true direction, as such:

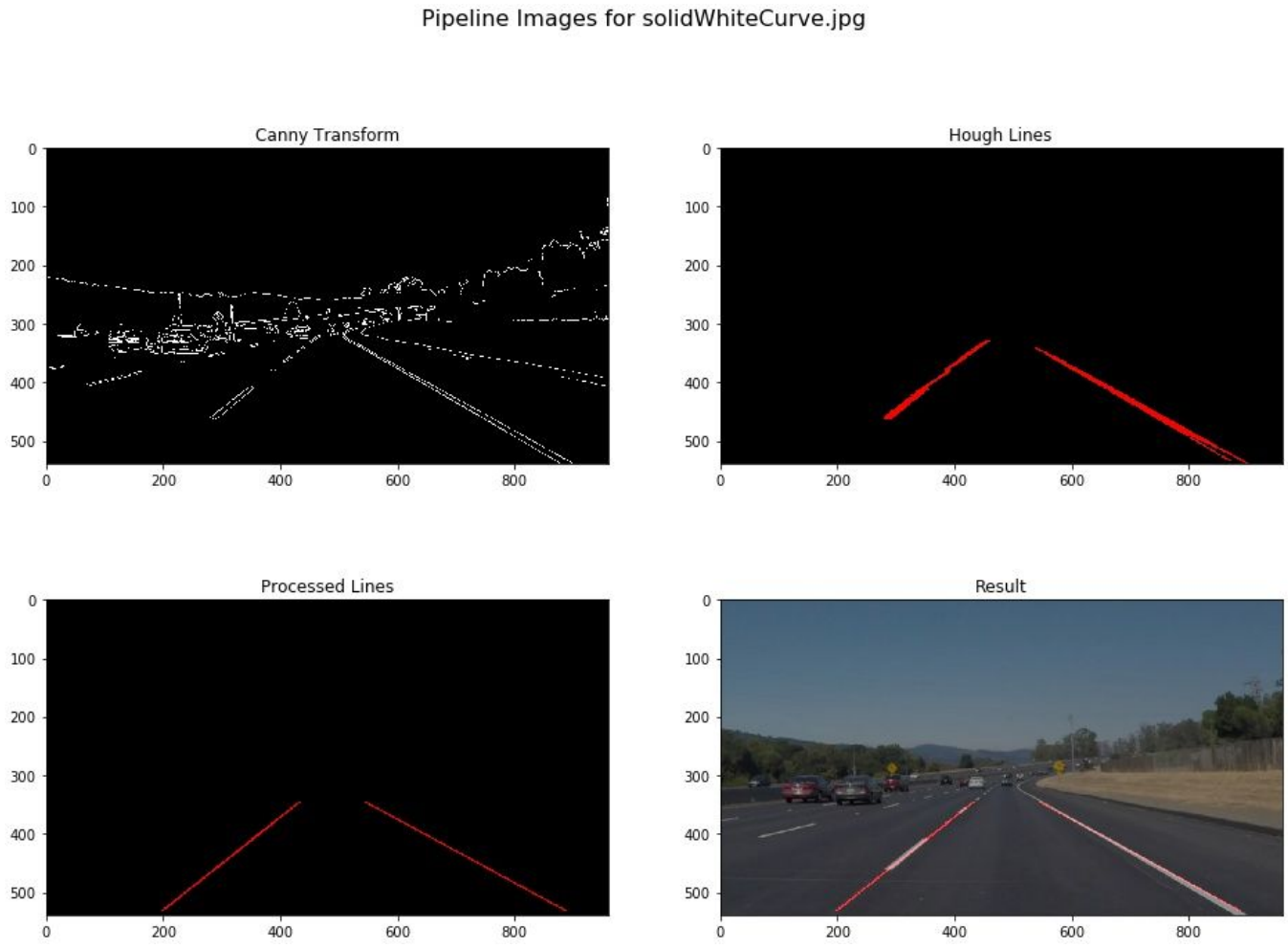


These new lines more accurately estimate the lanes' boundary. These new lines' slopes and intercepts were averaged and then averaged again using the lane boundaries calculated in previous frames. The result of the `process_lines` function consists of two lines, one for demarking the LHS lane boundary, the other demarking the RHS lane boundary. Both lines are extrapolated to the maximum and minimum values of y yielded from the Hough Line Transform from all previously processed frames of the current video.

The slopes and intercepts of those lines are then stored in global arrays via a method called `line_locker`. This was done such that previous lines calculated by the pipeline were accessible when processing the current frame.

Those lines output by the `process_lines` method were then drawn onto a black background and then that image was blended with the original color image to create a picture of the road with the approximated lane boundaries overlain.

The figure below depicts how the different steps in the pipeline modify and process the image from the camera feed:



2. Identify Potential Shortcomings

The approach taken in the pipeline is rather simplistic and likely has several shortcomings that would present themselves in real-world usage.

One of the foremost generalizations this pipeline makes is the assumption that lane markers are *straight* lines. Lanes, indeed, are often curved. The current version of this project would not be able to handle those curvatures.

It's easy to imagine real-world situations that would flummox this pipeline as written, as well. For example, lane markings are often not just single, straight lines. Double white or yellow lines can be solid or dotted - and I am uncertain whether the current manner of processing input images would be able to handle all those variations. Another example would be zig-zag lane markings leading up to pedestrian crossings in the UK - it's not clear whether the current method work work in that instance, either.

Construction areas also pose another challenge. Sometimes lane markings are replaced with concrete barriers or lanes markings are shifted so there appear multiple lane markings of different intensity or color. A Tesla on autopilot recently demonstrated this technology's inconsistent ability to adapt to these more challenging situations when, in a construction area, the car crashed into a concrete barrier when the lanes shifted[1].

I was also disappointed I could not figure out an easy way to actively and automatically calculate the region of interest. Using cameras with different mounting points, angles, etc would yield different points which define the area in which lane markings would be present.

Another shortcoming I noted was when attempting to apply this pipeline to the optional challenge video. The mix of shadows and concrete often made the lane markings blend in more seamlessly with the road and thus tuning the parameters to consistently detect their edges across the various road surfaces and lighting conditions posed a real challenge.

[1] Links to the Reddit post by the driver as well as dashcam footage from a following car.

3. Suggest Possible Improvements

There are many ways to improve the current pipeline to make it more robust, consistent and accurate.

Firstly, I would want to adjust it so it could work seamlessly with the Optional Challenge video. As written, my pipeline was having difficulties dealing with the frames containing shadows or concrete because the contrast was not sufficient to find lines in many of those instances. I would like to build in a way for the pipeline to manage instances where the Hough Line transform does not yield any lines or yields lines with a high variance in slope (indicating the transform was picking up more noise than desired).

There are also a myriad of different road, weather and lighting conditions a car must traverse on a regular basis. The parameters tuned for the various image processing algorithms seemed to work in a very specific set of instances. Therefore, it would be preferable to incorporate a means of tuning those parameters actively in order for the resultant processed image to yield the highest possible amount of actionable information.

As mentioned in part (2), I was also disappointed that I could not easily incorporate a means for actively identifying the horizon or region of interest in which the lane markings would reside. Being able to do this automatically, to adjust for various road gradients and sensor placements, would be ideal.

Lane boundary approximations should also be able to handle instances where lanes are not straight or near-straight. A lane-detection system for an autonomous vehicle must be able to deal with curved roads and lanes.

Another shortcoming mentioned in part (2) was the pipelines potential inability to deal with different types or colors of lane markings. A more robust system would be able to actively identify the color, type and significance of different lane markings and be able to respond to those factors (i.e. changing lanes permissible across dotted, but not solid, lines).

I imagine some of these possible improvements may be addressed in the Advanced Lane Finding project we'll be undertaking later this term.