Udacity Self-Driving Car Nanodegree

Project 1: Finding Lane Lines on the Road

Mark Helms, November 5, 2017

Goal/Summary

Make a simple lane-finding algorithm for front-facing images from a self-driving car, using primarily Python, OpenCV and NumPy. All three videos were successfully annotated to a high and usable degree of accuracy.

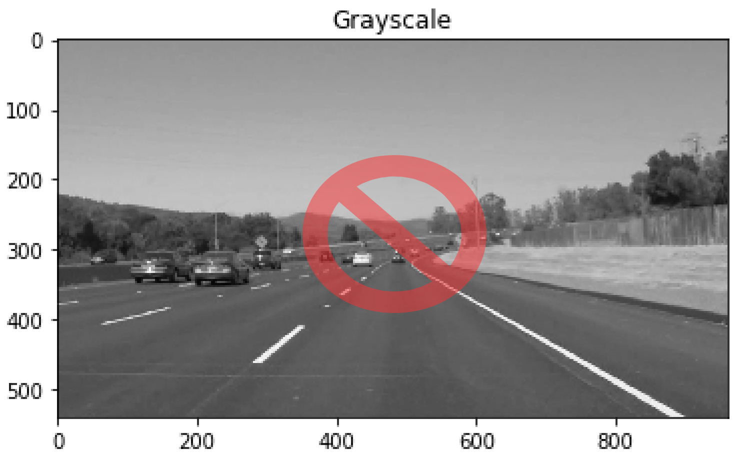
Approach

Step 1: Read File

Two sizes of images were identified. One has a height x width of 540 x 960, whereas the other has 720 x 1280. These are generally compatible, as they each have a height:width ratio of 1:1.78.

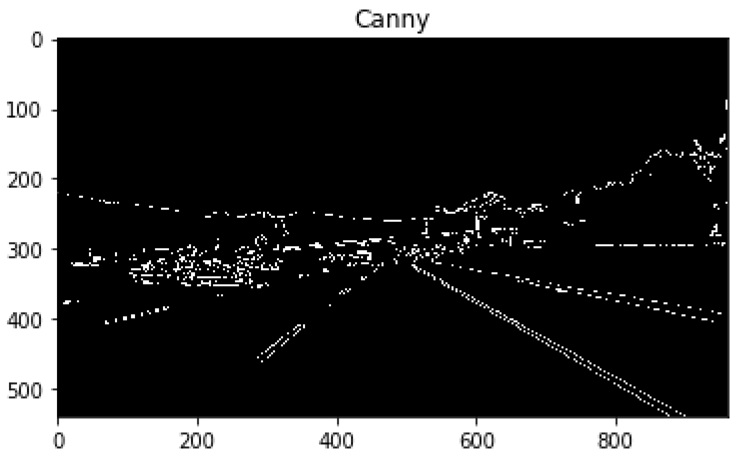
Step 2: ~~Convert to Grayscale~~

While this feature was enabled for most of the project, it was found that *not* converting to grayscale allowed the algorithm to perform better in the challenge video at two times: 1) when the car crosses a concrete bridge and the contrast of yellow on white is reduced and 2) when the road is shaded by the tree. There was no apparent quality loss in the other sections from doing so.



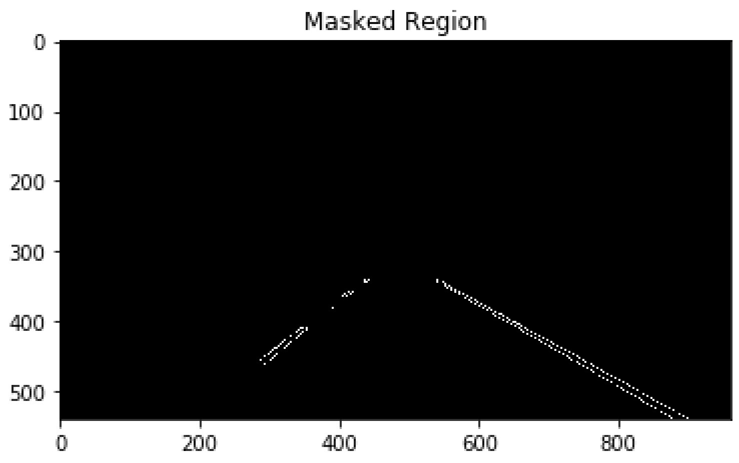
Step 3: Canny Edge Detection

This step performs a slight blur, then detects edges by passing a 2D kernel over the image. Pixels below a low threshold are turned off as edges, pixels above a high threshold are turned on as edges and pixels in between are turned on as edges as long as they are next to an edge.



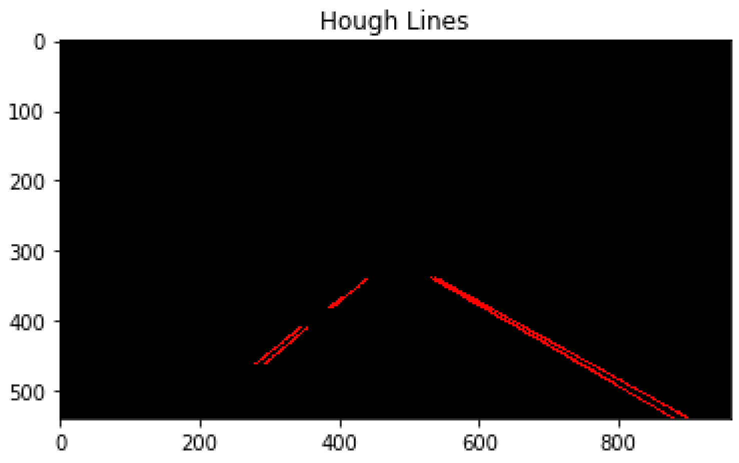
Step 4: Region of Interest

The entire edge detection image is masked by a region of interest that extends from the lower corners towards a vanishing point in the middle of the screen. This is one of the most effective methods of filtering the data down to the most important features.



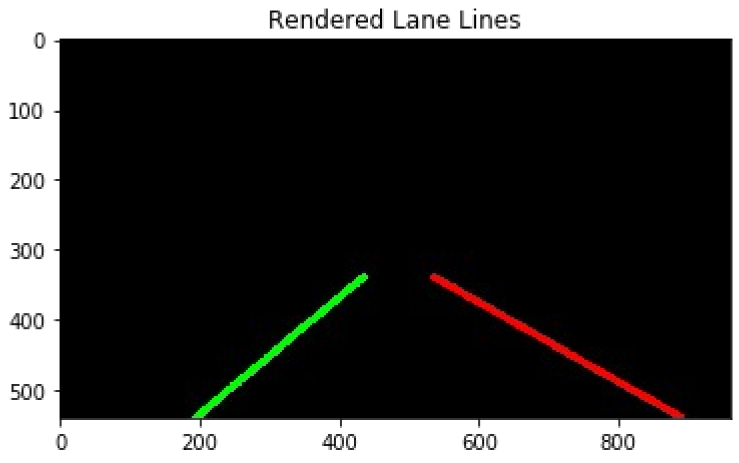
Step 5: Probabilistic Hough Transform

Using the OpenCV function HoughLinesP, lines within the edge detection image are identified and reported. They must meet a certain minimum length, with a maximum allowable gap.



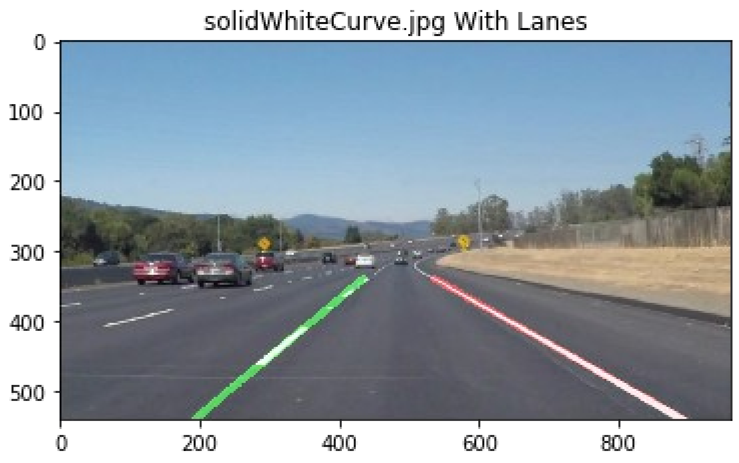
Step 6: Generation and Tracking of Lane Lines

To generate lane lines, a few steps are taken with the Hough lines. First, some lines are ignored, including: those with a slope that is too steep or shallow, those with a bias (y-intercept) value that is out of an expected range, and those which have points that cross over the middle point. The lines are split into left and right, depending on the sign of the slope. They are extrapolated to reach the bottom of the screen and a certain height towards the vanishing point, near the center of the screen. The points are averaged using a weighting average based on the length of the line used to generate them. The maximum length of the lines on either side is tracked and compared to a total distance in order to see if that side of the lane is demarcated with a solid line or a dashed line. The end points are averaged with the previous end points, with less-backward looking average for solid lines than for dashed lines, as they tend to be much more stable. This helps to settle the lines so that their appearance is smooth to the viewer.



If no acceptable lines are detected, the previously used lines are used again. When either line represents a dashed line, it will be shown as green. A right solid line will be shown as red. A left solid line will be shown as blue. There is also an option in the code to draw lines as dashed when the line they represent is dashed, however this is deactivated to more easily comply with a project Rubric that prohibits that feature.

Step 7: Overlaying the Lane Lines on the Image
The lane lines are overlaid on the image using a weighting adding method, whereby the lane lines appear somewhat transparent.



Potential Shortcomings

Some of the potential shortcomings of this algorithm are:
- It is tweaked for the given dataset (sunny, US-style road, long spaces between dashes, certain line width, image sizes)
- It assumes that most of the line will be visible.
- It assumes there won’t be sharp turns in the road.
- It assumes you won’t be changing lanes.

Possible Improvements

Some possible improvements for the project are:
- Allowing for changing of lanes, by allowing left or right line to transition to becoming a right or left line.
- Using the color of the lines to determine a region of interest or filter out Hough Lines.
- Using a curve method to the lines more accurately when drawing further ahead.
- Using polygons instead of lines of identical thickness to create a better looking annotation.