Lane Finding

Udacity Self-Driving Car NanoDegree - Project 1
jmoeller@gmail.com

Summary

This project implemented a simple lane-finding pipeline using OpenCV to draw left and right lane markers over images and video from a front-center camera view of a car travelling on US Highway 280 in California, provided by Udacity.

Pipeline

The simple pipeline consists of 6 stages:

1. Grayscale `(cv2.cvtColor)`
2. Gaussian Blur `(cv2.GaussianBlur)`
3. Canny Edge Detection `(cv2.Canny)`
4. ROI (Region of Interest) clipping `(cv2.fillPoly & cv2.bitwise_and)`
5. Hough Transform & Lane-finding logic
    a. Hough Transform `(cv2.HoughLinesP)`
    b. Line / Lane Mapping `(custom code)`
    c. Lane Line Averaging `(custom code)`
    d. Lane Line Extension `(custom code)`
6. Overlay & Alpha Blending `(cv2.addWeighted)`

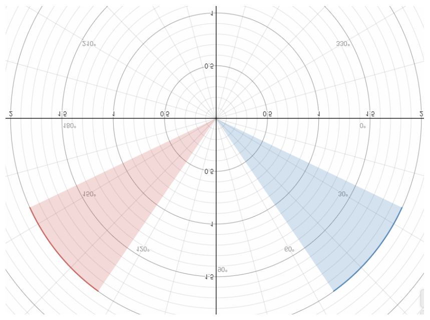
Algorithm Description

The openCV steps are well documented elsewhere, so I will focus on the Lane line mapping, averaging, and extension processes.

After tweaking openCV parameters on steps 1-5a to get an optimal set of lines from the hough transform (maximally identifying the lane markings, while having minimal false positives and using a tight ROI), the next step is to identify a set of lines that could reasonably mark the left and right lanes.

Lane / Line Mapping

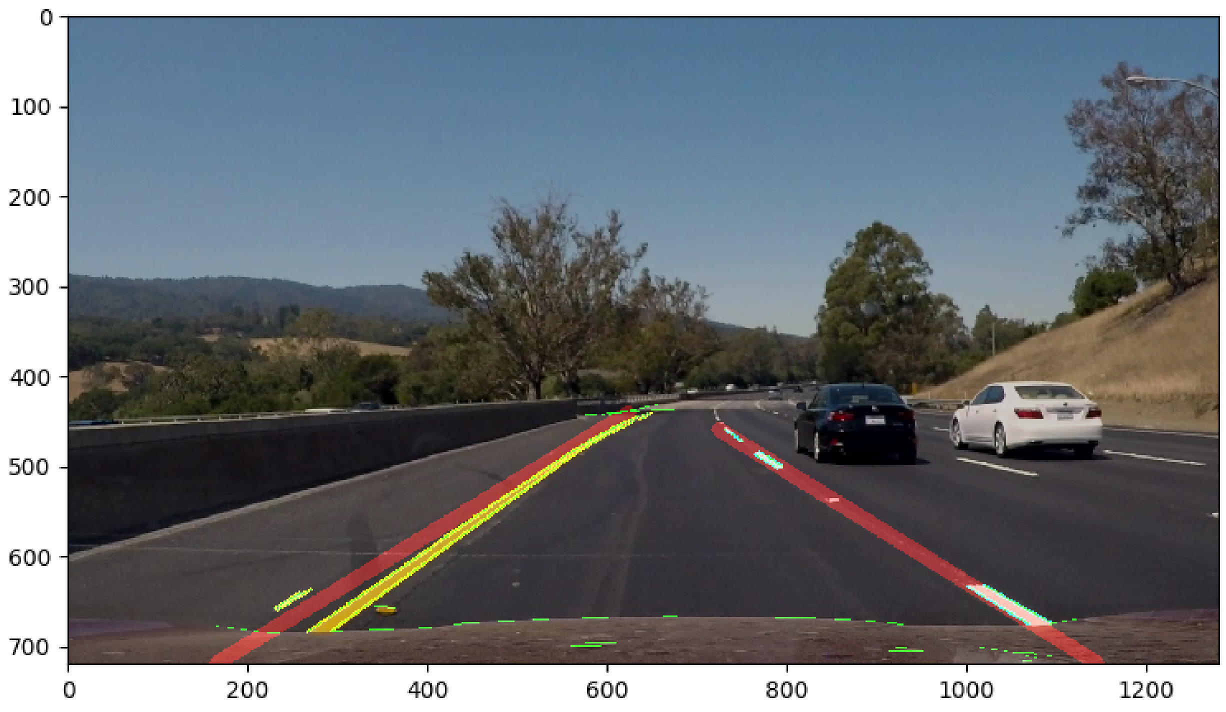
From the example images and videos, it appeared as though an angular metric would be a simple way to discriminate left and right and noise, so I mapped left lane lines to angles between 125 & 155 degrees (shown in red), and right lane lines to those between 25 and 55 degrees (shown in blue). These degree values assume the y-axis is inverted (increasing as you go down), like it is with our image representations in python.



Lane Line Averaging
The next step is to take all the lines from each set (left lane, and right lane), and “average” them. By “average”, what we mean is to average the slope **m** and the offset **b** across all lines in each set. This offers a crude approximation of the “average direction” for each line, as well as the “average intersect” along the bottom of the image frame, which we will use in the next step.

Lane Line Extension
The final step is to extend the lines to their maximal extent as seen in the image, and to the bottom of the image frame, to produce a result similar to the example. The maximal extent of each lane in the image is determined by picking the line endpoint with the lowest y coordinate in each set of lines. This is noisy, but it works well enough. The “average intersect” at the bottom of the image frame is used as the first x-coordinate for the lane line (the y coordinate being the maximum y extent of the image), and we draw a line up to the maximal extent, which we found earlier, with the average slope calculated earlier.

The final output looks like this:



In this image, the thin green lines denote all the lines we found using the hough transform, the yellow lines denote those selected as left lane line candidates, the cyan lines denote those selected as right lane candidates, and the red lines show the final lane lines calculated and drawn by the pipeline.

Drawbacks and Potential Improvements

This algorithm works reasonably well on the source videos provided, but performance will suffer in any number of cases not accounted for in the development of this simple pipeline. I will list and discuss these below.

Nonexistent or Ambiguous Lane Markers

This case is obviously unaccounted for in this pipeline. If there are no lane markers on the road, this pipeline will be unable to identify any lanes. If there are ambiguous lane markings or many candidates for “lane markings”, the pipeline may misidentify the lane, or improperly average non-lane features into the lane marker calculation. This problem is a very real issue in real SDC systems, and resulted in the crash of a Tesla Model X on US HWY 101 in Mountain View, California in 2018. [1]

Solutions to this problem are complex. A precomputed map mixed with GIS data could help solve the problem of nonexistent lane markings, but does not account for things like construction or other instances where the local geometry doesn’t match the model. Expanding the feature set, improving the recognition algorithms, and using these together to build a lane abstraction model is likely the only way to identify a true “right of way” for vehicular autonomy.

For ambiguous or low-contrast lane markings, creation of a hysteretic lane model may help in situations where there is a temporary loss of contrast or marking, so that the lane markings “remember” where they were in the past and use this data to inform where they should be now.

Fixed Image Pipeline Parameters

Because all parameters in the pipeline are fixed, the system is not adaptable to different driving conditions: night, wet, snow, etc. An adaptive image processing pipeline would be preferable, although the design and implementation of such a system is fraught with difficulty. There are many variables at play when it comes to identifying a lane marking, and they can all be defeated by something as simple as an autoexposure error. The only real solution to this problem is to collect an enormous amount of training and validation data to evaluate performance in real-world conditions.

Another simple example of this is the ROI clipping and angular filtering. Yes, it works for highway driving, but does it work when faced with a San Francisco hill? If you’re at the base, the lane lines appear almost vertical. If you’re at the apex, you may not even see them through this camera view. An adaptive solution would take these cases into consideration (and it’s likely that ROI is the wrong approach in general, except for removing obvious non-data like the body of the vehicle).

Summary

I implemented a fixed-function pipeline for lane line recognition and annotation using python and openCV, and described above how lanes were identified and labeled. Drawbacks (and there are many) to the solution were discussed, as well as potential mitigations and modifications to improve recognition.

References

[1] NTSB Preliminary Report HWY18FH011
https://ntsb.gov/investigations/AccidentReports/Reports/HWY18FH011-preliminary.pdf