ENPM 673, Robotics Perception
Project 1: Lane Detection.
Due on: Wednesday – Feb. 28, 2018

1 Lane Detection - 100Pts

In this project we aim to do simple Lane Detection to mimic Lane Departure Warning systems used in Self Driving Cars.

2 Details

In this section we discuss a few pipelines (with subtle differences) for lane detection.

Note: You are given videos from a driving car and you have to extract frames from the videos and the output should be a video submission - DATASET.

2.1 Brief Pipeline - Lane Detection - 80 Pts

Below are few possible pipelines, YOU CAN USE ANY OTHER APPROACH as you feel comfortable. In any case, please submit a report explaining your approach with relevant outputs after each step.

2.1.1 Simple Hough Lines

In this pipeline the brief idea is to detect vertical edges in the image and find *most consistent* lines which we presume to be lanes. .

1. Denoise the image - so that the noise doesn't aggravate any of the further image processing tasks (Noise Removal).

Figure 1: Input image

2. Apply edge detection to extract vertical edges. You can evaluate performance by either using all edges (Canny) or just vertical edges (Sobel - vertical operator) (Edge Detection).

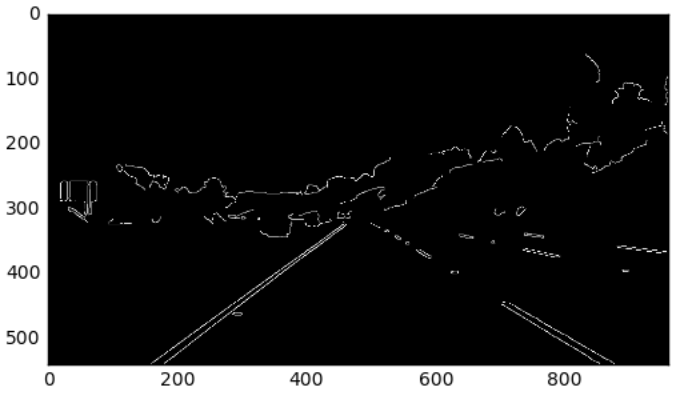


Figure 2: Canny Edges

3. Binarize the edge detection output - using appropriate threshold (if you are using `imfilter` with some vertical edge filters).

4. Extract the *Region Of Interest* i.e., you may mask out the top half of the image as we can assume the upper half of the image doesn't have lanes.

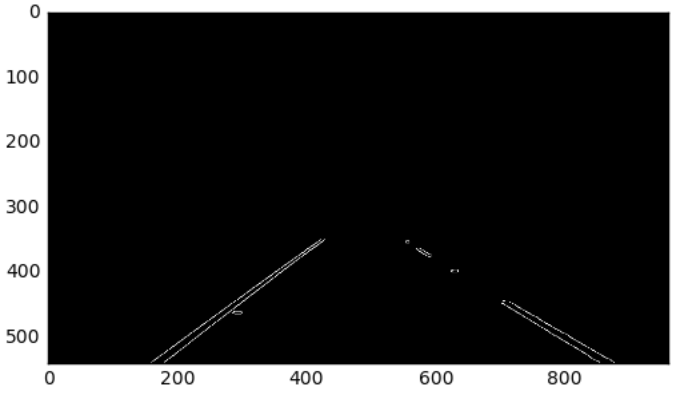


Figure 3: Extracted Region of Interest

5. Get Hough Lines from the previous output (Object Analysis).

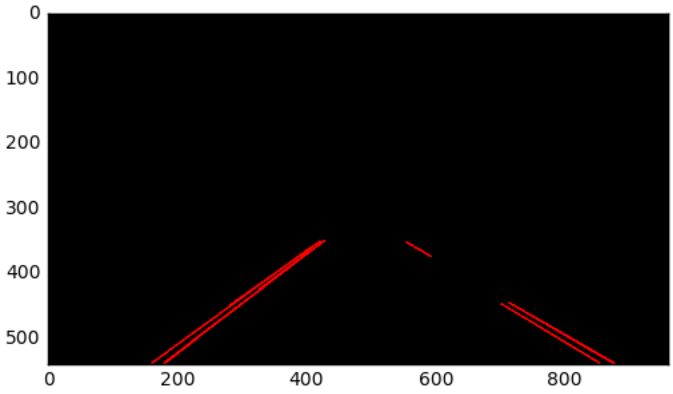


Figure 4: Hough Lines

6. Find out the peak hough lines, you can further group them into two groups (positive, negative gradients) and extrapolate lines in each group (houghpeaks - Object Analysis).

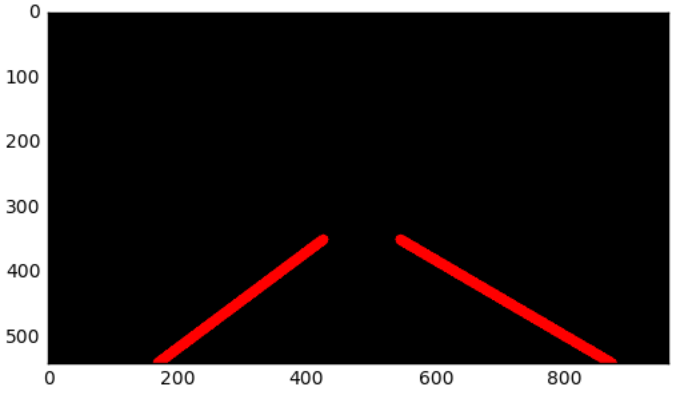
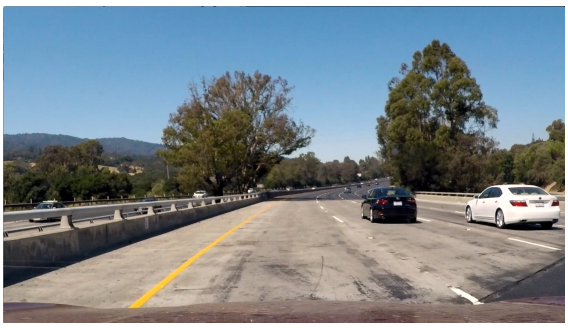


Figure 5: Extrapolated Lines

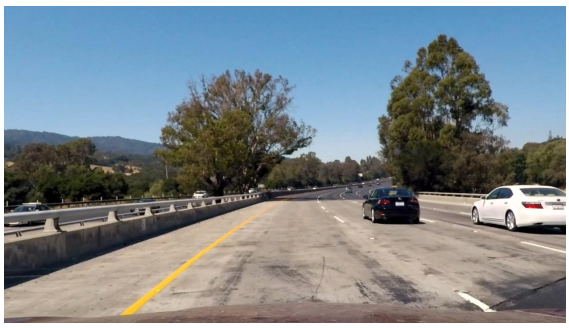
2.1.2 Histogram of Lane Pixels

Here we map the detected lane to a "top view" (as seen from the top). The idea is that in the top view, a lane will be bounded by near vertical lines. Thereafter, you can sweep across the width of the lane (x-axis) to generate a histogram of the number of pixels along the y-axis of the image i.e., say for every 10 pixels along the width of the image get the count of lane pixels in y direction and then identify the *top two* regions to get the *left & right lanes*. Finally you project back these pixel locations to the original camera view (the input image), to obtain the lanes.

1. Undistort the input image (unDistortImage).



(a) Original image



(b) Un-Distorted image

2. Denoise the image and extract *edge candidates* for lanes i.e., using Canny/Sobel edges or color thresholding (Object Analysis).

Figure 7: Combining thresholded & Edge output

3. Identify the regions of the lanes. That is pick the corners of the *trapezoid* (corresponding to a lane) in the image and transform them to a rectangle (as seen from an imaginary top view)

using perspective transform (fitgeotrans).

Figure 8: Top View of Lane

4. Generate a histogram of *pixel count along the y-axis* while you sweep across the width of the image.

5. Extract the top two regions with highest pixel count, which correspond to the left & right lanes i.e., now you will ONLY have two regions corresponding to the vertical lines of white pixels (lane pixels).

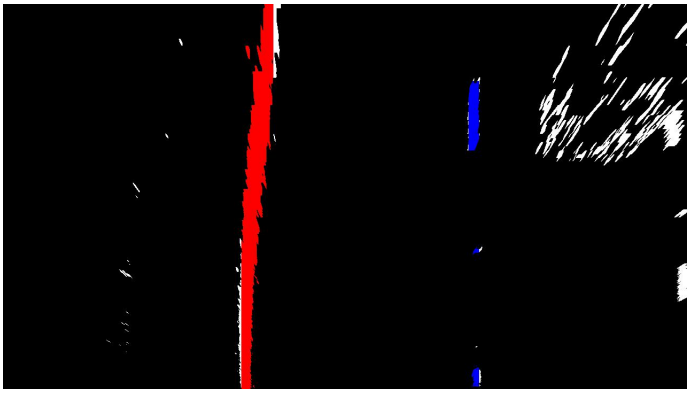


Figure 9: Vertical line candidates

6. Fit a polynomial each to both of the regions (polyfit).

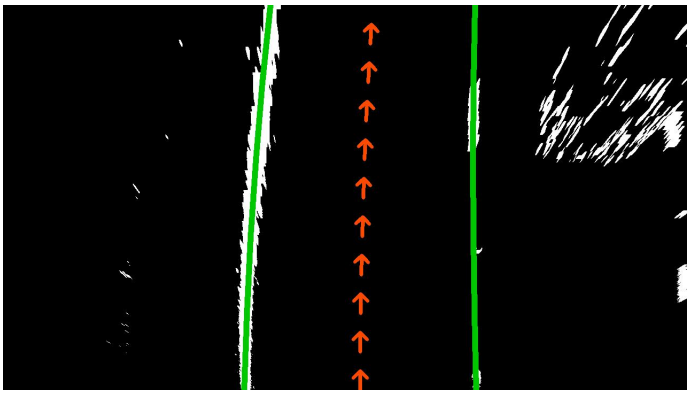


Figure 10: Fitting Polynomial to the extracted lanes

7. Transform back the lane pixels (using the previously calculated perspective transform) to the *normal view* of camera as seen in input image (imtransform).

Figure 11: Projecting lanes back to input image

2.2 Predict Turns - 20 Pts

One idea to predict turns is to extract the *central line* (averaged over left & right lanes) and check if the *central line* has positive gradient (curving towards right) or otherwise. Another way is to use the curvature of the polynomial (fit to right or left lane) to predict the turn. Another approach is to find the vanishing point i.e., intersection point of hough lines corresponding to left & right lanes. If the vanishing point is *fairly* close to the image center (in x-direction)
i.e., ($x_{vanish} \overset{\sim}{=} \frac{width}{2}$) then the road is straight, if ($x_{vanish} \geq \frac{width}{2}$) then the curve could be right and if ($x_{vanish} \leq \frac{width}{2}$) then the curve could be left. One thing to note here is even though the road is straight, the vanishing point might stray left/right if the car is not exactly at the center of the lane. So you might want to keep a threshold i.e., $\frac{width}{2} - thresh \leq x_{vanish} \leq \frac{width}{2} + thresh$ to determine if the road is straight.

2.3 Few Other Tricks

In Section 2.1.1, when you extract peak hough lines, instead of just choosing the top two candidates (for left & right lanes) you can choose the top few hough lines, group them into *left lane candidates* & *right lane candidates*. Now for each of these gropus, get the corresponding pixels and fit a polynomial to the pixel coordinates.

In Section 2.1.2, once you have the top view of the lane, you can extract hough lines (the advantage here is the lanes are near vertical lines, so hough lines might identify the top candidates relatively well). Once you get the top two candidates, get the pixels corresponding to these lines and fit a polynomial to each of the lane.

3 Extra Credit

- Detection of curved lanes - *challenge video in Dataset.* – **50Pts**

4 Submission Guidelines

Your submission **SHOULD be a ZIP folder** (no other file extensions) with the naming convention `YourDirectoryID_proj1.zip` on to ELMS/Canvas. Additionally follow the guidelines given below:

1. You will have a parent directory `P1_Submission`.
2. Under `P1_Submission/LaneDetection` you will have three sub-folders `code`, `input` and `output`.
3. You should also submit a report (`Report.pdf`) under `P1_Submission/LaneDetection` folder.

Your output video should be *at least* `15 seconds` of duration.

5 Useful Resources

Lane Detection has been done using different methods in MATLAB and `OpenCV` and the pipelines given here are taken from multiple such resources.

- For Section 2.1.1, example from MATLAB's Computer Vision Toolbox - Lane Departure Warning.
- For Section 2.1.2, python implementation - Advanced Lane Detection.
- Constrained Hough Transforms for Curve Detection.

6 Acknowledgement

We are using the Advanced Lane Detection dataset from Udacity - Self Driving Nanodegree program.

7 Collaboration Policy

You are allowed to discuss the ideas with fellow students, but you need give credits in the report. But the code you turn-in should be your own and if you **DO USE** (try not to and it is not permitted) other external codes - do cite them and you might get partial credits. For other honor code refer to the University of Maryland Honor Pledge.

You should take the effort to search online for any help regarding function usage, however, any concept related queries can be discussed in TA Office Hours.