Udacity Self-Driving Car Nanodegree

Project 1: Finding Lane Lines on the Road

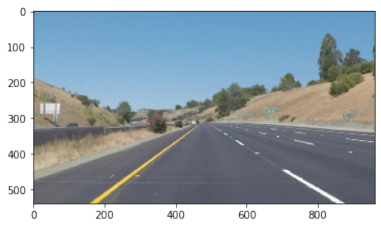
Hana Chew

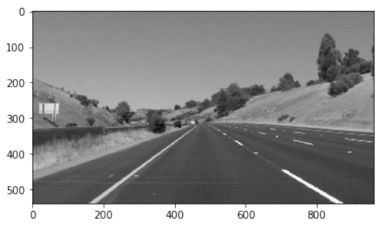
1 Task

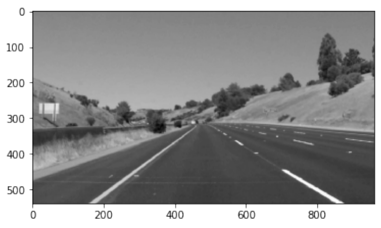
The goals of this project are to make use of the `Jupyter` notebook with starting helper functions in order to develop a pipeline that finds lane lines on a road – first on static images, and later use this as a basis for a function that also works on videos. The second goal is to write a reflection on the work and the steps taken.

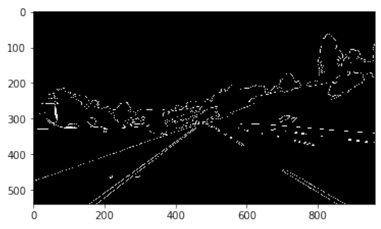
2 Description of Pipeline

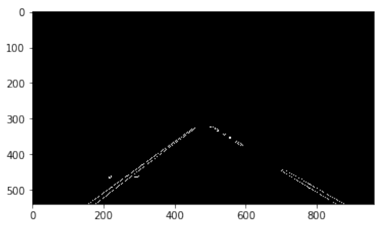
This initial part of this process was covered in the lectures, and consist of the following steps for each image (or later, each frame of a video):

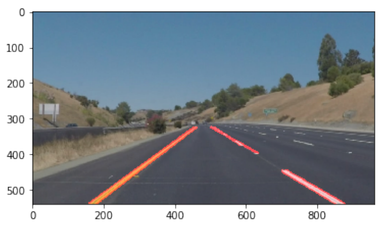

1. Original Image


2. Grayscale


3. Gaussian Blur


4. Canny Edge

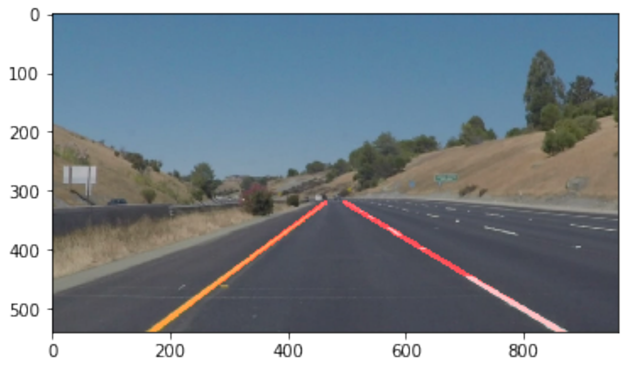

5. Relevant Mask


6. Overlay Hough Lines

I. The images (or frames from each video) are read.
II. The images are converted to grayscale.
III. A Gaussian Blur is applied with a kernel size of 5 to remove noise.
IV. The canny edges of the image are detected.
V. A region of interest is defined, edges outside of defined trapezium is ignored.
VI. A Hough transformation is applied on the extracted edges – the various parameters were tuned in order to obtain a suitable result for the given images and videos. This results in a cluster of lines that already closely resemble the lane markings. These are however, still broken or dashed, and the task was to create two continuous lines.

VII. The `draw_lines` function is invoked and applied to the lines obtained from the Hough transformation in order to get the straight lines required for the lanes (explained in detail in the next section).

VIII. The detected solid lines are then drawn with the original image overlaid to get the desired outcome, given in the figure below.



End Result of Lane Detection Pipeline

3 Averaging and Extrapolating to obtain Full Lane Lines

The next part of the process involved modifying the `draw_lines` function in order to obtain the complete lane lines, rather than a cluster of broken dashed lane markers.

I. Each of the lines that were identified in the Hough transformation are then extracted, and the values analysed. Each line is represented by two sets of coordinates, which are used to obtain the gradient of the line in question.

II. If the gradient is positive, it would be part of the left lane marker, if negative it would be classified as a right lane marker. A threshold of the magnitude of the gradient is introduced (in this case, 0.5) in order to discount any horizontal lines which may be picked up (car bonnet, changes in tar or road, stop lines).

III. Using the Right Lane cluster of points as an example, the function `polyfit` is then used in order to come up with the line of best fit for these points as a way of averaging out the various lines detected. This gives us the value of m, the gradient, and c, the intercept.

$$y = mx + c$$

IV. The y-values for the start and end points of the single desired lane line would be fixed: they would start at the bottom of the image, and continue up towards the just under the halfway mark of the height. This would be used as the 'vanishing point'.

V. Given that we know the y-values and the m, and c, the x-values for the desired lane line, we just need to inverse the straight line equation in order to obtain the relevant X-values that we require for the lines to be drawn.

$$x = \frac{y - c}{m}$$

VI. Using these values, in combination with the threshold of a magnitude of 0.5 again, we then draw the right line.

VII. The process is then repeated with the Left Lane points cluster and the end result is then two straight lines for each lane marker, left and right.

4 Potential Shortcomings of Current Pipeline

I. Within `draw_lines`, if Hough lines are not detected within the first frame, there wouldn't be a lane line drawn.

II. If the cars ahead are not within their lane markers it could affect the lanes drawn.

III. Region of interest currently does not account for cameras mounted on different positions on the car, as demonstrated in the challenge video when the bonnet still shows up and was originally identified until draw lines was modified to ignore it (lane markings still start at the bottom of the frame, rather than the end of the bonnet).

IV. `Challenge.mp4`: The contrast is a little low for the left yellow lane lines in the challenge video. Perhaps the contrast may be increased to improve detection.

5 Possible Improvements to Pipeline

I. Find out the ideal vanishing point for the lines – where would be the ideal point for the lane markers to end.

II. The straight line lanes could be modified to give a (curved) line of best fit - or some other method to handle curves.

III. Optimisation of code so it runs more efficiently - currently `draw_lines` is not completely optimised.

IV. Test with other cases to improve robustness. For example, if the car drifts, needs to change lanes (or if another car changes lanes).

V. Above could be solved by modifying code to include an averaging component or a weighting component with respect to the gradients or lines from previous frames (in a video) – this could for example be flagged by too drastic gradient changes from one frame to the next – and an averaging of the values could perhaps account for lane changing.