P1 – Finding Lane Lines on the Road

Udacity CarND
Author: Mas Chano
Date: 02/20/2017

Reflection

A simple lane finding program was written in python. The following are reflections on my experience working on this project.

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of 7 steps. First, the image was converted to grayscale in preparation for feature extraction. Below is the original and grayscale image.

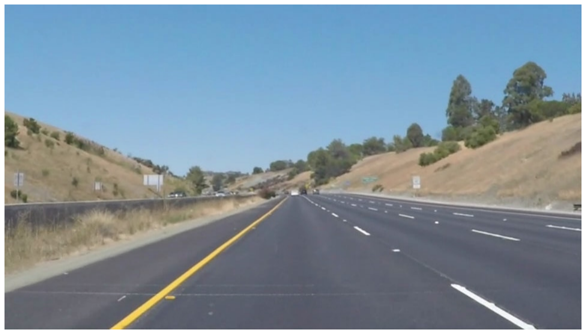	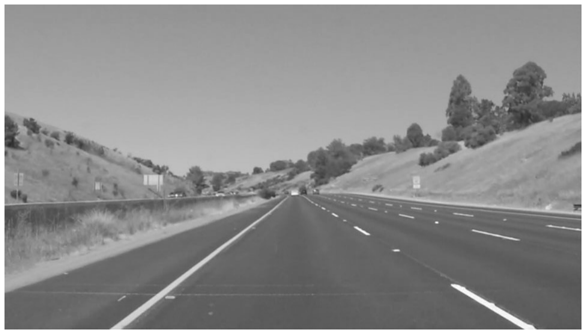
Fig. 1: Original (whiteCarLaneSwitch)	Fig. 2: Grayscale

For the second step, a Gaussian Blur with kernel size 5 was applied to the grayscale image. For cases where there may be more noise (such as the challenge scenario), an increase in kernel size may be justified. The Gaussian Blur is applied to smooth out contrasts in the image (Fig. 3). Small contrasts not related to lane markings will be smoothed out while more prominent edges, which are more likely to be lane markings, will still be detectable.

Third, the Canny algorithm is applied to the blurred grayscale to detect edges in the image (Fig. 4). The values for the low and high threshold were set at 50 and 150 respectively to thin out the detected edges that have a weaker gradient. As seen in the resulting image, only prominent edges remain, where detected edges have a value of 255 and the black space has a value of 0.

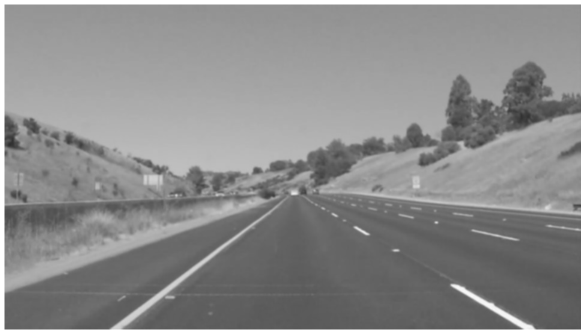	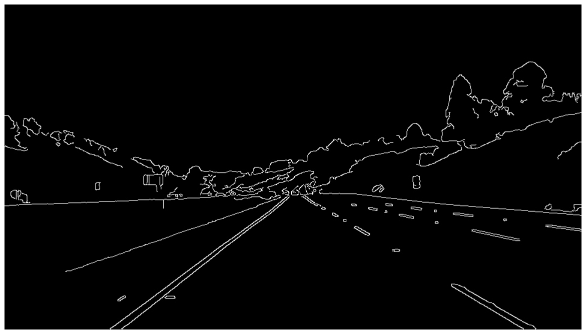
Fig. 3: Grayscale with Gaussian Blur, kernel_size = 5	Fig. 4: Canny edge detection

Next, we define a region of interest (ROI) mask for step 4 and use it to mask-out edges that are not of interest to us. When using the Canny algorithm, edges of objects not related to lanes were also detected (such as trees and hills). Since we are interested only in edges related to lanes, we can define a trapezoidal region (Fig. 5) in front of the car as our ROI because lane lines will typically stay within this region.

We perform a bit-wise AND operation of the ROI and the Canny Edge detected image. The resulting image can be seen in Fig. 6. All pixels outside the ROI now have value of 0 (black), while edges within the mask retain a value of 255 (white).

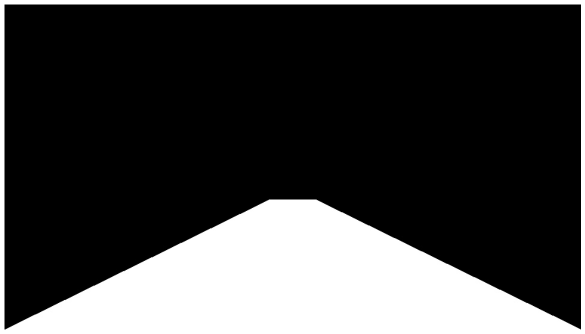	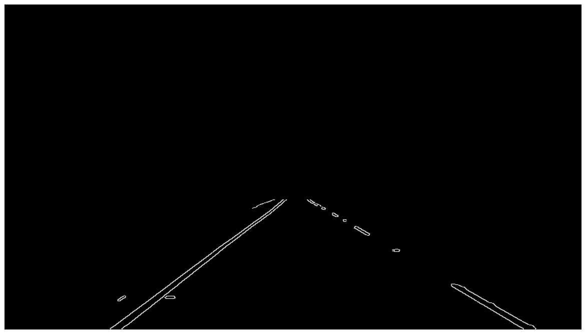
Fig. 5: Region of Interest	Fig. 6: Region of interest applied to edge detected image.

For step 5, we perform the Hough transform on the edge pixels in the image and analyze their representation in Hough space to determine if a group of edge pixels belong to the same line in image space. The following parameters were used in the transform to determine whether edge pixels belong to the same line in image space.

- rho = 1 pixel
- theta = pi/360 radians
- threshold = 9 intersections
- min_line_length = 40 pixels
- max_line_gap = 15 pixels

The output of the Hough transform function in CV2 is an array of edge lines in image space. For step 6 (my equivalent of changes to the draw_lines() function), we use the edge lines obtained to identify a single left lane line and a single right lane line. To identify and draw these lane lines, all edge lines found from the Hough Transform were iterated through and the slope of each edge line was calculated.

We can see that lane lines are angled inwards from the perspective of the car looking towards the horizon. Therefore, the slope of the left line is negative and the slope of the right line is positive. (Please note the top-left corner is has coordinates x=0, y=0).

Considering the left lane line for a moment, we can further assume that the slope of the left lane line in the image will not be close to vertical (-90 degrees) or close to horizontal (-180 degrees), if we consider x=0, y=0 as the origin. We can separate out edge lines corresponding to the left lane line to those that are angled between -15 degrees and -70 degrees. This corresponds to a slope of roughly -0.27 and -3 (obtained by taking tangent of the angle). A similar analysis was done for the right lane where we only consider angles between 15 and 70 degrees to form the right lane line.

After filtering out which edge line belongs to the left and right lane line, a polynomial fit is applied to the edge lines coordinates (start and end points) to find a 1st degree polynomial equation for the left and right lane line. With the equation, we can draw the line on the image as can be seen below.

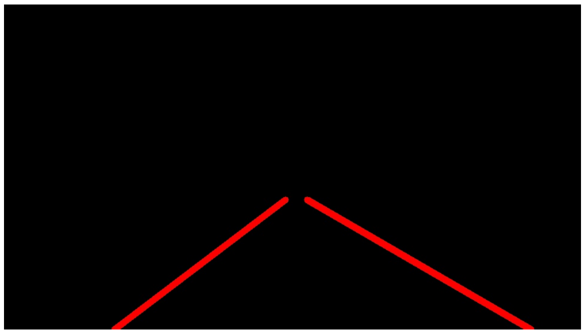
Fig. 7: Lane lines (Calculated and extrapolated from 1st degree polynomial equation found by analyzing Hough Transform edge lines.

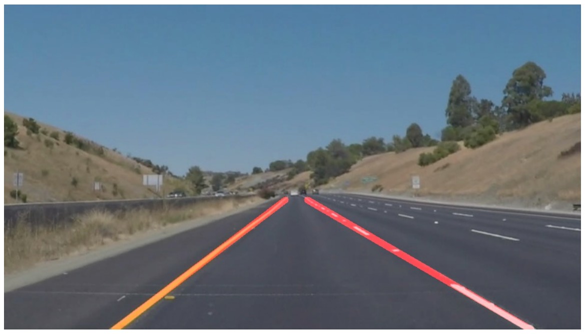
Fig. 8: Lane lines applied to original image.

2. Identify potential short comings with your pipeline

The pipeline is still susceptible to noise and will not perform well outside sunny, shadow-free, straight road, well-marked, highway conditions. When applied to the challenge problem, it could not properly find lanes because the pipeline also detected edges from irregular lighting, shadows, car body, and road color (Fig. 10).

As a result, the output of the Hough transform contained lots of unrelated edges lines causing the resulting slope calculation for the lane lines to be swayed as shown in Fig. 12.

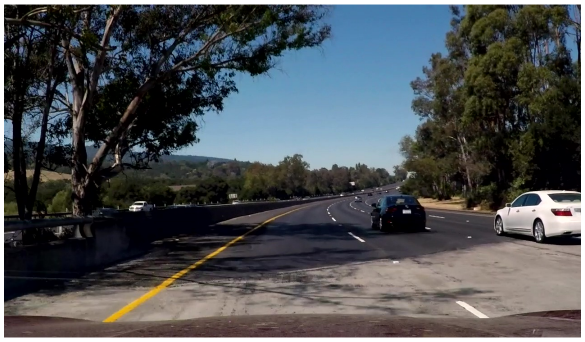
Fig. 9: Unfavorable condition found in challenge video frame

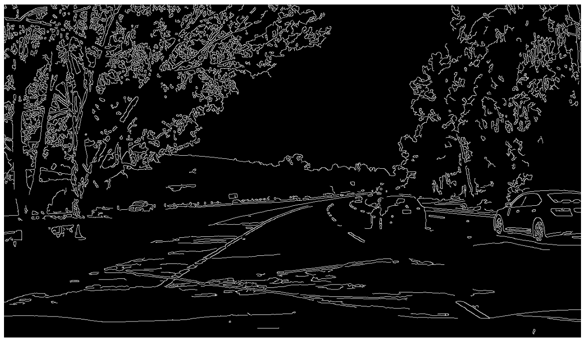
Fig. 10: Canny edge detection applied to unfavorable condition, prior to masking

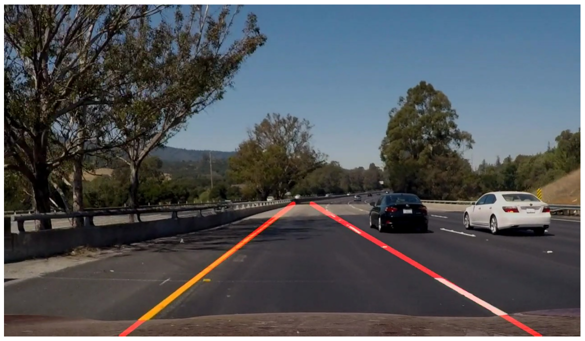	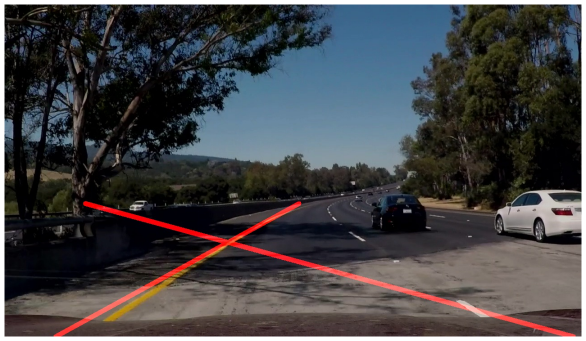
Fig. 11: Pipeline applied to challenge movie where conditions are relatively favorable.	Fig. 12: Pipeline under unfavorable conditions with shadows, road texture change, irregular lighting.

Furthermore, because I try to use the slope of edge lines output from the Hough transform to filter out whether they belong to the left or right lane, there are cases where I will get no edge lines to match the criteria for the left or right lane. Currently, this results in a skipped line calculation where there are no projections for lane lines (this does not occur frequently, even in the challenge video but does occur).

The pipeline will most probably not work if there are sharp turns, intersections, dark streets, or no lane markings.

3. Possible improvements to the pipeline

The pipeline could benefit from better noise rejection prior to performing the Hough transform. Currently, I convert to grayscale and blur but it would be worth experimenting with different color spaces or performing some sort of normalization to reduce the effect of shadows and make the lanes stand out.

Another area of improvement would be during the processing of the lines output from the Hough transform. The current method is rather brute force (and possibly inefficient) as described in Step 6 and only evaluates the current frame. One way to smooth out jitters when calculating lane lines would be to use some sort of moving average where results of past frames would affect the lane line projections of the current frame. This will help with occasional shadows and road irregularities but will not be as effective if road and lighting conditions are constantly changing.

A third area to consider would be better parameterization of the Hough transform and investigate whether there are more things we can do in the Hough space. However, in general, I believe it would be more beneficial to pass a better image (where noise is rejected upfront) to the edge detection algorithm and the Hough transform than in tweaking the Hough transform parameters.