Finding Lane Lines on the Road

June 30, 2020

1 Objective

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on the work in a written report

2 Reflection

2.1 Pipeline Description

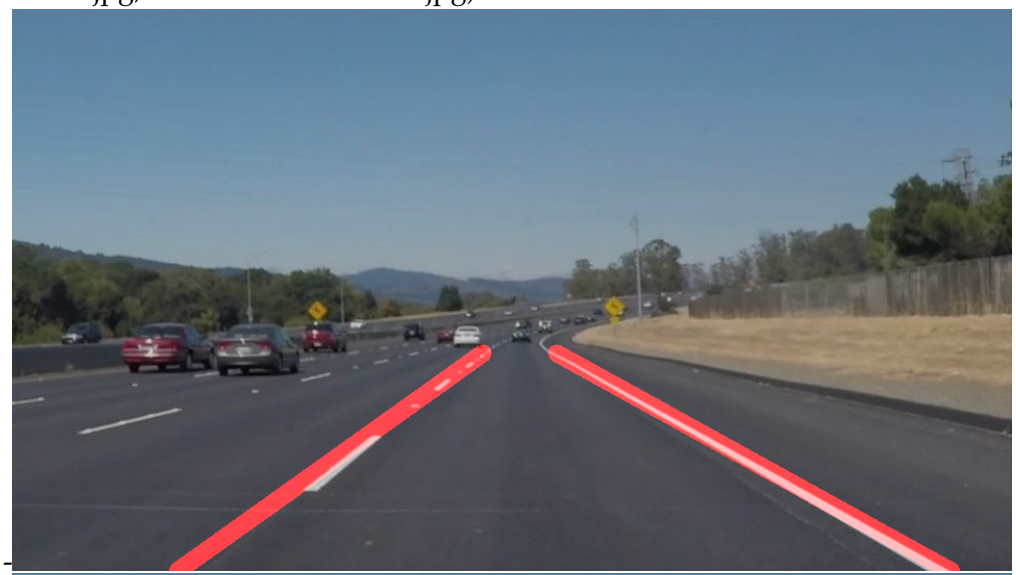
The key objective of this project was to build a pipeline that can identify lane lines on the road given images or video clippings from car cameras. The following were the key steps in the pipeline for images:

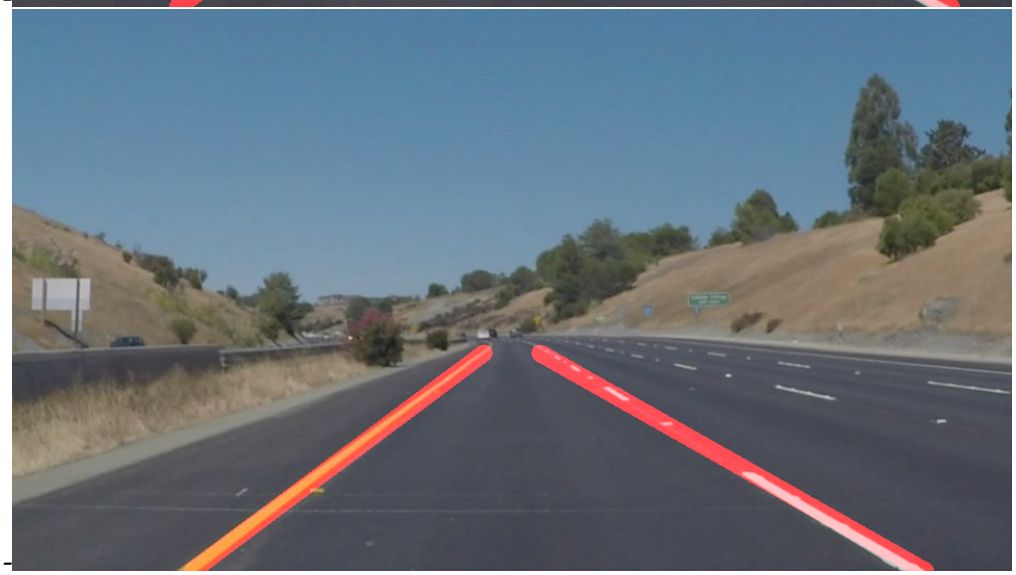
- Step 1 : Read Image
- Step 2 : Convert image to grayscale
- Step 2 : Blur Data to remove noise. A gaussian kernel of *size* 5 and *sigma* 1 was used to blur the grayscale image.
- Step 3 : Canny Edge detection. The *low* and *high thresholds* for the canny edge detector was fixed to 50 and 150 respectively post multiple iterations over different values.
- Step 4 : Region selection was performed to focus the lane detection on a specific region in the image. The region was fixed to have a trapezoidal shape with the base at the lower edge of the image and the top edge below the half way point of the image and narrowed to a length of 40
- Step 5 : Run hough transform to find lines in the image from the detected edges in the selected region. The *rho* and *theta* were fixed to 1 each while *threshold, max_line_len and max_line_gap* were set at 20 after paramter tuning to account for broken lanes that had alternating long lines and smaller lines. The *draw_lines()* function combined the lines into two single straight lines as described in the next section.
- Step 6 : The final step was to use the *weighted_img()* to superimpose the lines onto the original image using the default parameters

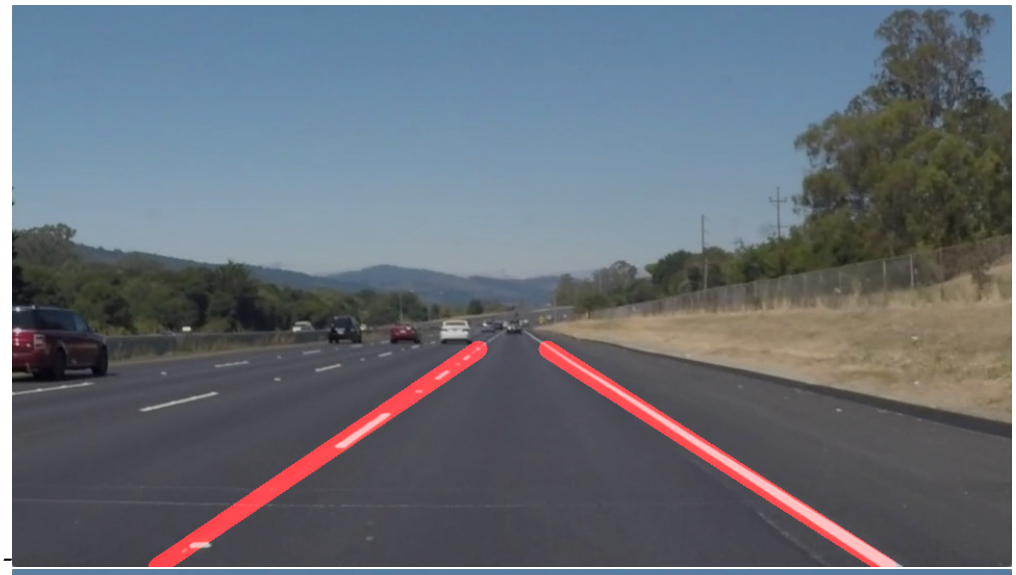
Draw_lines() function To aggregate the lines detected in the left half and right half of the image to single straight lines representing the left lane and the right lane, the *draw_lines()* function logic was modified as follows:

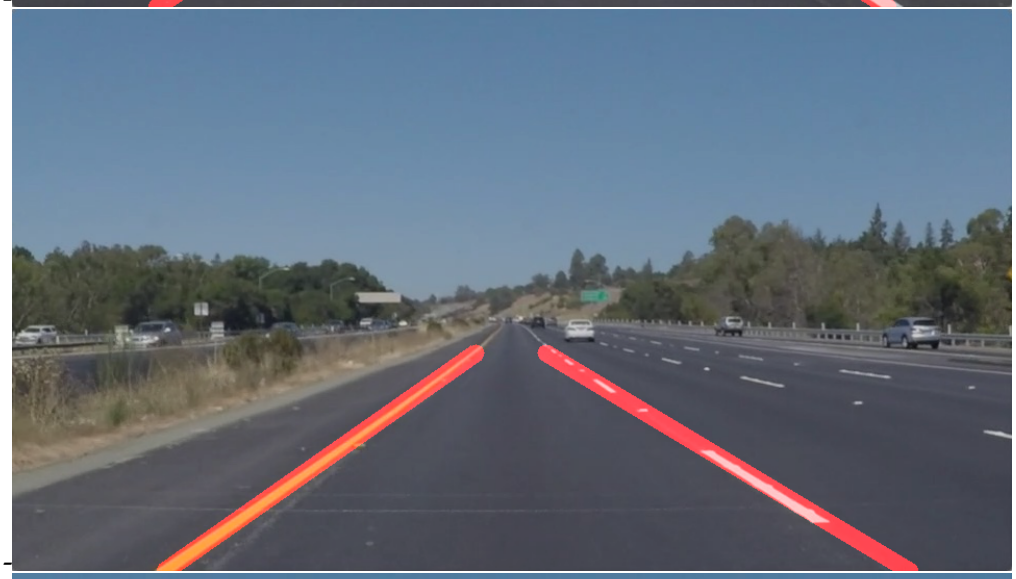
- Calculate the slope and intercept of each line
- Lines with negative slope are collected together as the lines that represent the left lane and lines with positive slopes are collected together as lines that represent the right lane
- In each lane, remove noisy lines by putting thresholds for acceptable slopes (Left lane lines have slopes between -0.4 and -0.8 & right lane lines have slopes between 0.4 and 0.7)
- Calculate the median slope and median intercept for both the lanes to define the single line representing the respective lane.
- Since the Y axis values for the points in each of the lanes are known (the lower end would be equal to the image height and upper end would be equal to the coordinates as defined in the region selection, step 4 in the pipeline), compute the respective X axis values.
- Once all coordinates are computed draw the left lane and right lane with a defined thickness

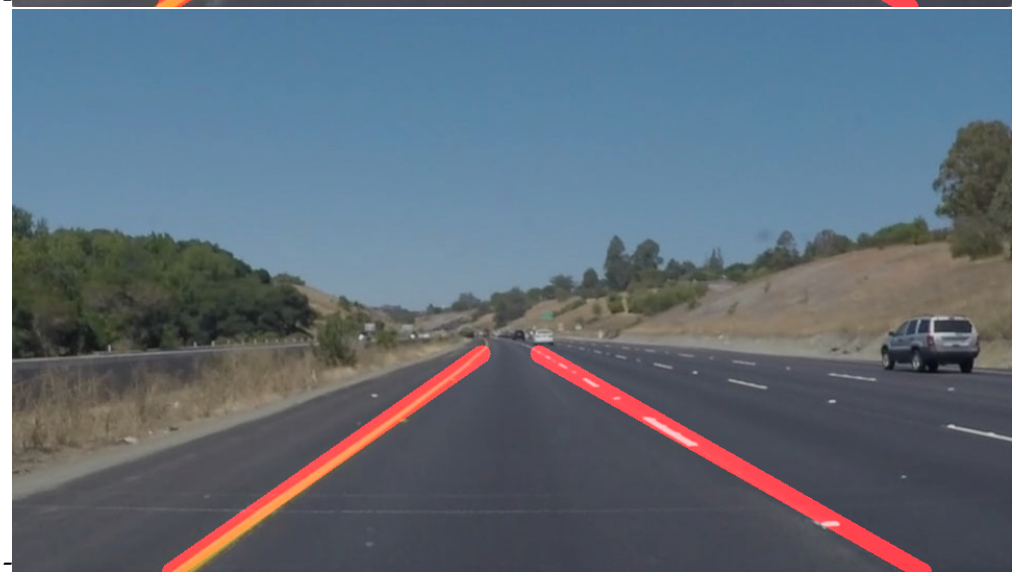
Validation : The outputs of the 6 test images are as below (in the following order solidWhiteCurve.jpg, solidYellowCurve2.jpg, solidWhiteRight.jpg, solidYellowLeft.jpg, solidYellowCurve.jpg, whiteCarLaneSwitch.jpg)

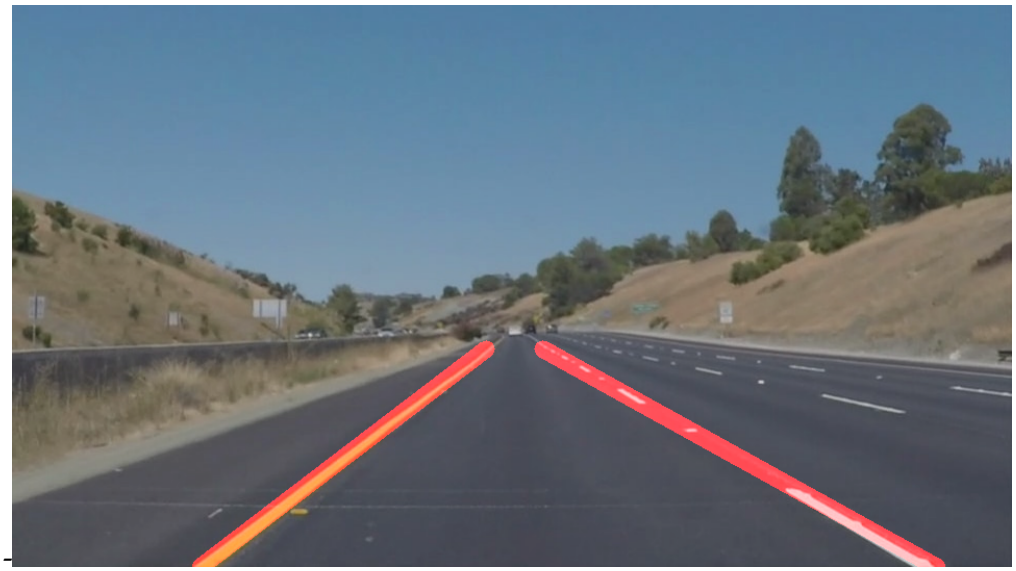












2.2 Detecting lanes in videos

The pipeline to detect lanes from videos had a few additional steps :

- Read Video in required format (using the VideoFileClip & f1_image functionalities)
- Sharpen original image using unsharp masking (https://en.wikipedia.org/wiki/Unsharp_masking) - This was primarily effective for the *challenge.mp4* video under the optional exercise
- Continue steps as in the previously described pipeline for detecting lanes in image

2.3 Identify potential shortcomings with your current pipeline

- **Detecting turns and curves** : the current pipeline can detect straight line lanes and will not give accurate results in the presence of turns and curves
- **Broken lanes** : In case of non edge lanes (lanes with broken lines as pattern), the **draw_line()** function recomputes the lines' characteristics (slope , intercept) on every frame and this can give a jittery feeling to the detected lane line
- **Pedestrian Crossings** : The algorithm might not be able to correctly identify the top end point of the lanes in case of pedestrian crossing which would get detected as valid lines

2.4 Suggest possible improvements to your pipeline

- Improve sharpening of images for better and smoother edge detections
- Handle turns and curves by accounting for multiple lines in the same lane; a solution could be split th region of interest into two halves and define different lines for the same lane in the two different halves.
- Capabilty to leverage information from previous frame to guess lane in the current frame in case of video