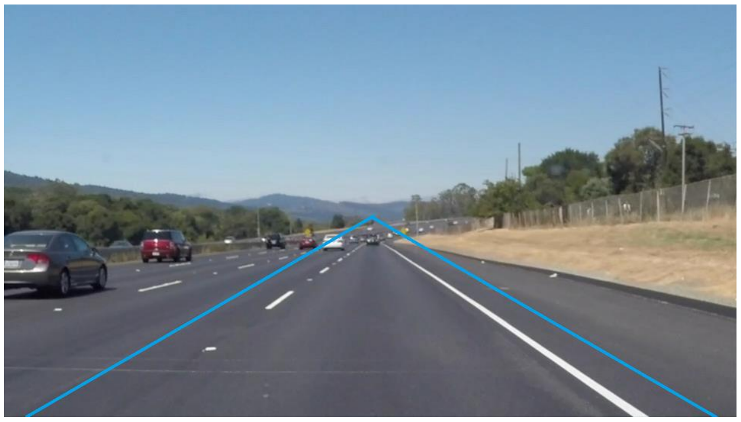
Finding Lane Lines

Pipeline description (Refer code "P1_video1_video2_WORKS.ipynb")

My program consists of the following steps:

1. Fetch color images from a video stream in each time frame sequentially and pass them to the function "find_lanes"
2. Inside the 'find_lanes', the input color image is converted to grayscale
3. Then the grayscale image is smoothed using a Gaussian blur function with kernel size of 5
4. Then the smoothed image is passed to the canny edge detection function to identify the edges in the image. Brighter lane lines (yellow or white) with dark background are most likely to be identified along with other sharp edges in the picture. In order for the pipeline to be able to work with both white and yellow lane lines, the lower and higher threshold values for the canny function are set as 60 and 170 respectively.
5. To filter out the unwanted portion of the image where lane lines are not likely to appear, a polygon mask is created and applied on the canny output. This step assumes that the camera is mounted at a fixed location in car. Shape of the polygon is selected after a view trial-and-error on the sample images. For the videos "solidWhiteRight.mp4" and "solidYellowLeft.mp4", shape of the polygon is as shown below (image size 960x540):
   {(40, 540), (480, 290), (920, 540)}



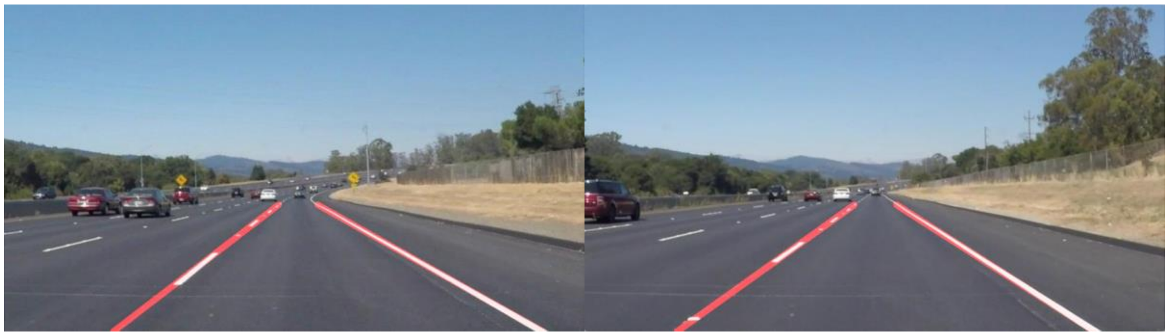
6. Next the canny output within the above mask is passed to the "Hough" transform function to identify the line segments. Choice of Hough transform parameters are as follows: resolution of 'rho' = 2 pixels, resolution of 'theta' = pi/180 radians, minimum number of Hough votes = 40, minimum length of line = 40 pixels, maximum line gap = 25 pixels. These values work fine for both the test videos. The 'hough_lines' function identifies and outputs the (x,y) co-ordinate values of each line segments satisfying the aforementioned parameters.
7. Once the line segments are identified, next step is to use these information to draw two lines, one each for the left and right lanes. This is achieved by the 'draw_lines' function and it involves the following steps:
   a. Fetch each line segment data from the Hough transform output
   b. Slope classification: Calculate the slope of each line segment and make the following decision
      i. If the slope of a line segment is positive, identify it as the segment belonging to the right lane. Copy the x1, x2 values to an array 'x_r' and y1, y2 values to an array 'y_r'

        ii. If the slope of a line segment is negative, identify it as the segment belonging to the left lane. Copy the x1, x2 values to an array ‘x_l’ and y1, y2 values to an array ‘y_l’

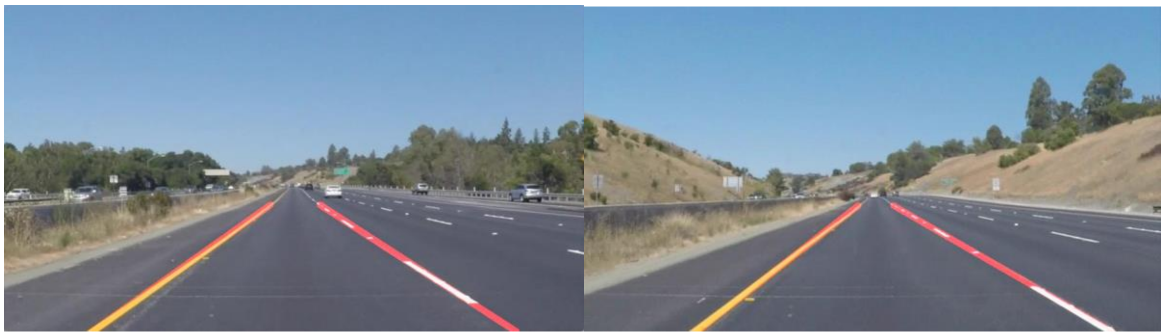
    c. Extrapolation: Take all points classified as right lane points copied in x_r and y_r array. Make first degree polynomial fit using the least squares polynomial fit function ‘numpy.polyfit’. The outputs of this function are ‘m’ and ‘b’ parameters in the line function (y = mx + b), where m is slope and b is bias of the fitted line.
    d. Using ‘m’ and ‘b’ parameters computed in step ‘7.c’, calculate the ‘x’ points for the given starting and ending values of ‘y’, representing the final right lane segment to be drawn on the image
    e. Repeat steps (7.c) and (7.d) for the points classified as left lane points and get the final left lane segment

8. Final step is to draw the left and right lane segments on to the original color image using the function ‘weighted_img’

Test image outputs:



SolidWhiteCurve(left), solidWhiteRight(right)

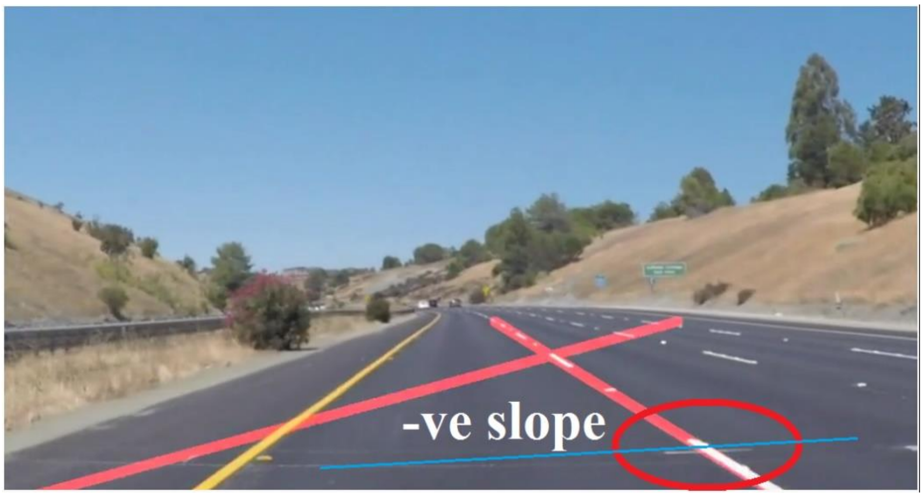


solidYellowLeft (left), whiteCarLaneSwitch (right)

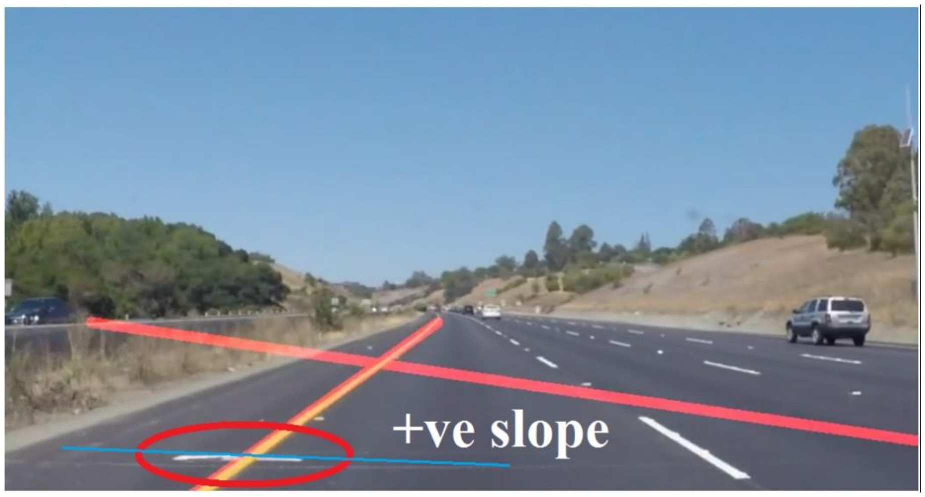
Shortcomings and modifications in software:

The above mentioned pipeline worked fine for the “solidWhiteRight.mp4” video stream. However, for the “solidYellowLeft.mp4” video, there were some shortcomings.

At some epochs of this video, there were certain horizontal markings on either side of the lane lines, that were wrongly identified by the algorithm as line segments. Such line segments were getting classified either as right or as left lane segments by the aforementioned ‘slope classification’ logic (step 7.b above). Example plots are shown below (Example A and B):



Example A - Wrongly identified line segment with negative slope - contributes to left lane points



Example B - Wrongly identified line segment with positive slope - contributes to right lane points

If these line segments are not filtered then they will contribute in extrapolating the line segment points (step 7.c and 7.d). Therefore to avoid 'false detections', such points are filtered based on the location of co-ordinate points. For right lane detection, only those points that are in the right half of the picture are chosen, likewise for the left lane detection.

Refer to the code, in 'draw-lines' function; two 'if' statements:

```
 # x points for right lanes must reside in the right half of the image, as indicated by x > 460
if (x1 > 460) & (x2 > 460):
    #Right lane co-ordinates
    x_r += [x1, x2]
    y_r += [y1, y2]
```

```
# x points for left lanes must reside in the left half of the image, as indicated by x < 500
if (x1 < 500) & (x2 < 500):
    #Left lane co-ordinates
    x_l += [x1, x2]
    y_l += [y1, y2]
```

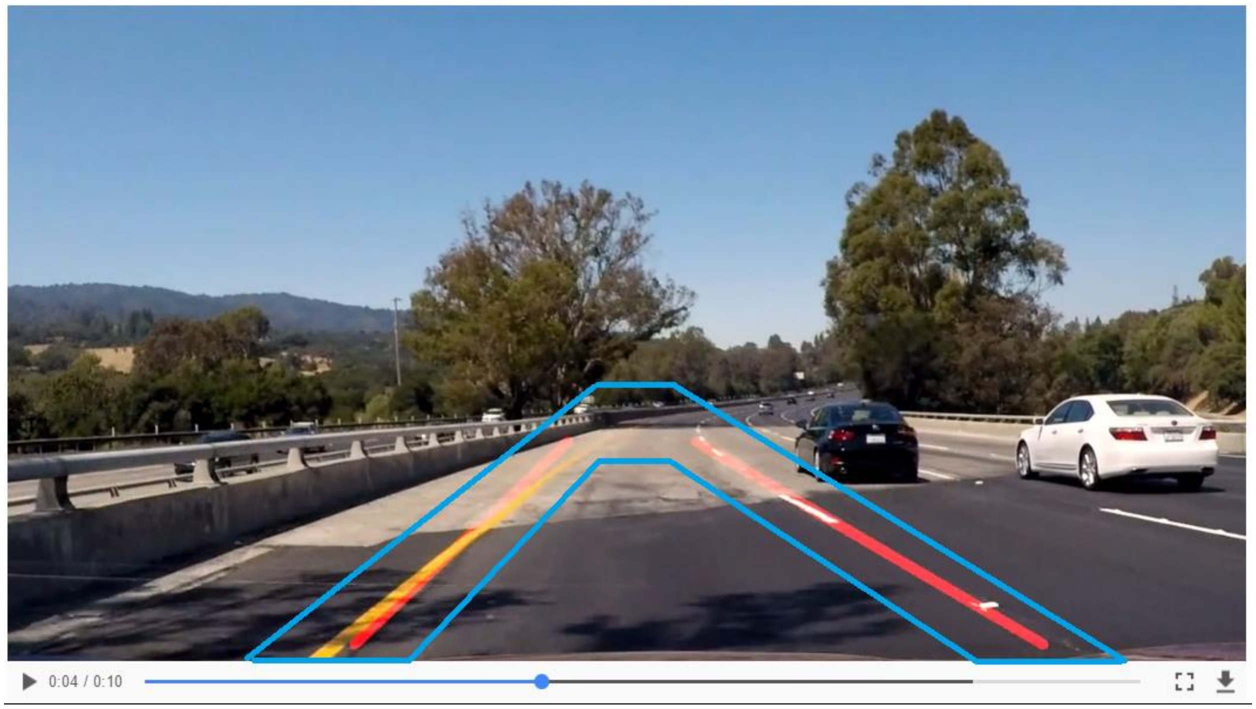
Testing “challenge.mp4” (Refer code “P1_challenge_video3_WORKS.ipynb”)

At first “challenge.mp4” video did not work with the above code. Therefore following changes are made to the software.

Change 1: The masking co-ordinates were not suitable for this video. Therefore with several trial and errors, it was determined that the following mask points were able to provide some level of lane detection functionality:

Mask points: {(200,690), (658, 420), (682, 420), (1180, 690)}.

Change 2: Even after selecting such a mask, the results were futile at those epochs where the car was moving under tree shades. To detect the pale yellow lanes even under the tree shades, the canny edge threshold values were changed to 50 (low threshold) and 130 (high threshold). Nonetheless, still some errors were observed during certain time epochs when the road color changes (often, there seems to be some abrupt change in color of road tar). To reduce false detection under such conditions, the shape of the mask is changed as shown below:

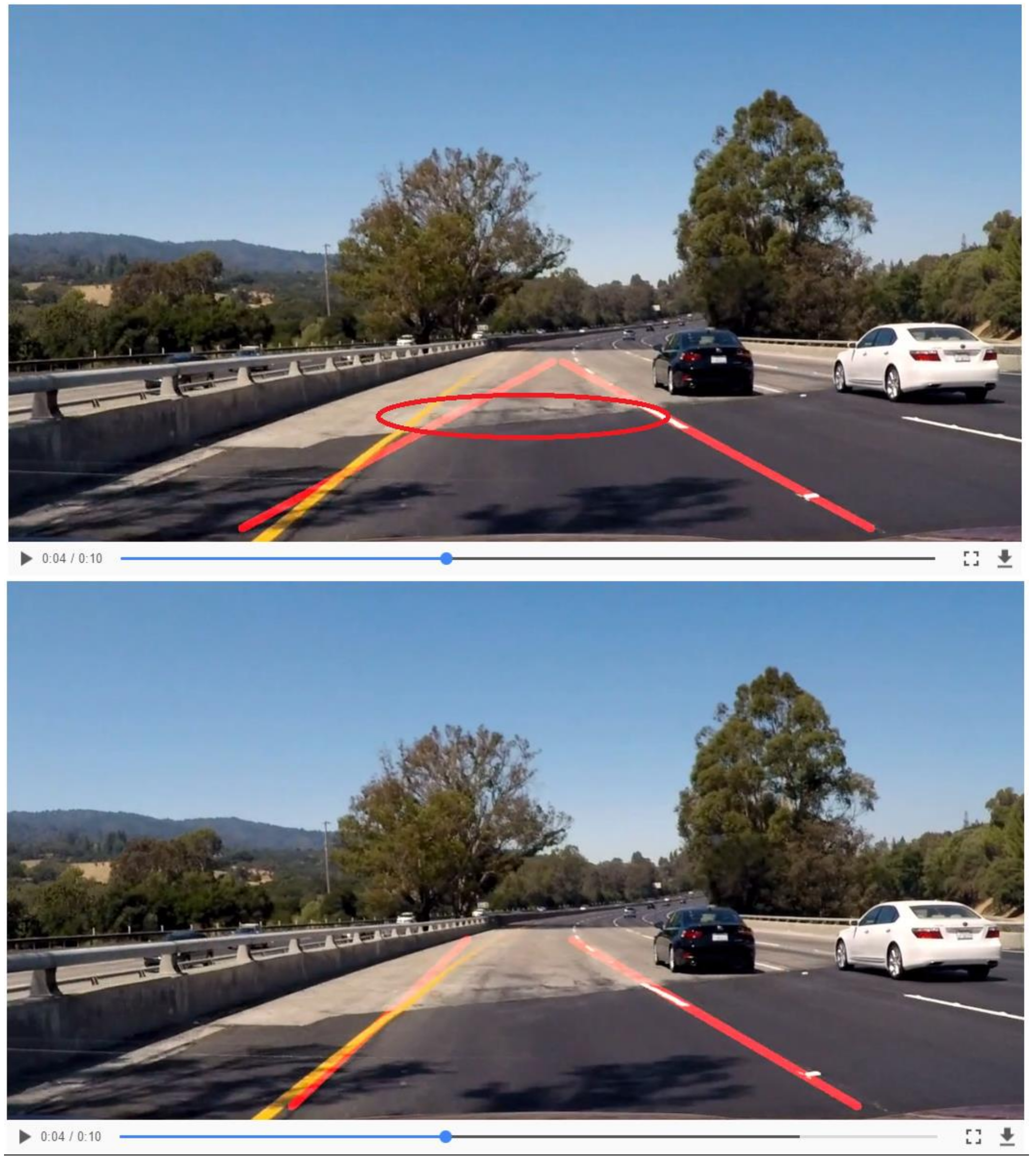


New shape of the mask for "challenge.mp4": helps to avoid picking pixels wrongly identified as edges due to tree shades or change in color of road tar

Change 3: Since most line segments were shorter and often not distinctively identifiable by the canny function, the hough line parameters are changed as follows: threshold = 5, minimum line length = 5, maximum line gap = 10

Change 4: Significant modifications were required for the ‘draw_lines’ function

a. Only those points belonging to the left portion of the image were chosen to polyfit the left lane and likewise for the right lane.
b. Only those points within ‘y lower limit’ 450 and ‘y upper limit’ 680 were chosen; this is where the lanes are most likely to be present (this is sort of second level of masking)
c. To avoid picking the line segments due to change in road color, the line segments were filtered based on the ‘range of slope’. Line segments with too low slope values (absolute slope <0.6) to too high slope values (absolute slope >1.0) were not chosen for polyfit operation.



Before performing slope-range based filtering (top), after slope-range based filtering (bottom) as explained in 'Change 4.c'

Possible improvements

1. Even after making several modifications, adapted specifically for “challenge.mp4” video, there were false-detections at couple of the time frames. This was mainly due to residual edges detected within the mask region during abrupt road color changes. These can possibly be avoided by implementing advanced filtering methods, or by implementing ‘adaptive shape changing masks’.
2. Second possible improvement could be to implement a higher degree polynomial fit to extrapolate the non-linear line segments. This may help to identify lanes during turns in road.