Finding Lane Lines on the Road

Reflection

The goal of this project is to provide the solution that will be able to detect the lane lines both in the image file as well as video file. In both cases the pipeline for detecting the lane lines will be the same since video stream can be represented as a sequence of images

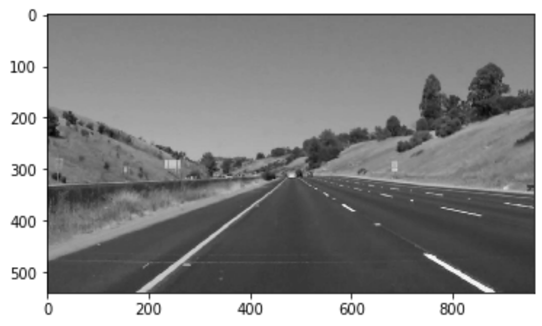
1. Pipeline description.

My pipeline for detecting lines of displayed road lane consist of following steps:

1. Convert the original image to grayscale:

   Function: grayscale(img))

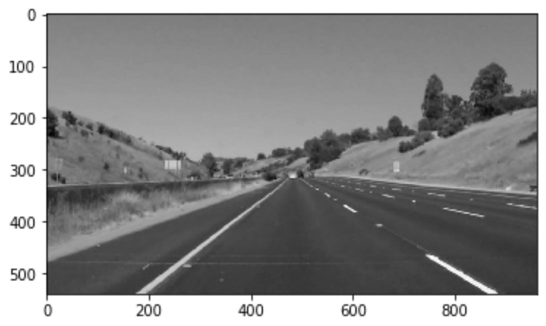
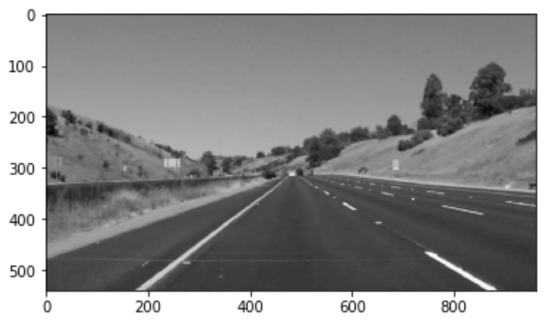
   Input / Output



2. Apply Gaussian smoothing to grayscaled image (bluring):

   Function: gaussian_blur(grayscale_image, kernel_size)

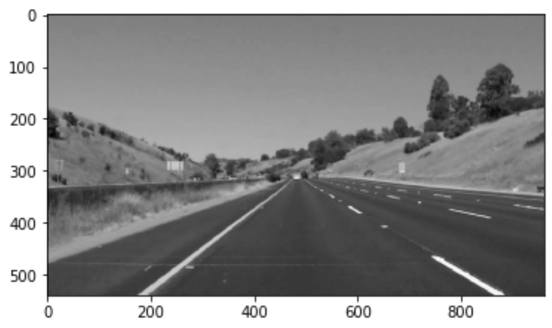
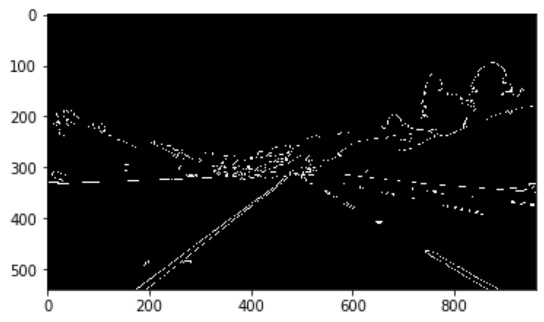
   Input / Output

3. Apply Canny edge detection to blured grayscale image:

   Function: canny(blured_grayscale_image, low_threshold, high_threshold)

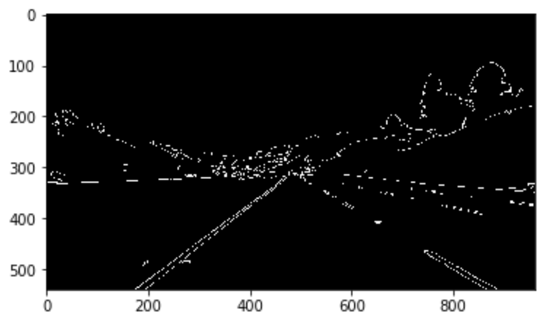
   Input / Output

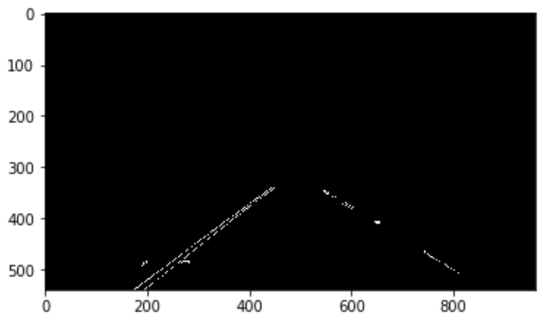
4. Define vertices of the tetragon that represents the 'region of interest' in the image (area where we are expecting the lane lines to be displayed + small margin)
5. Mask (blackout the pixels) all irrelevant area on ‘Canny edged’ image except defined ‘region of interest’:

   Function: region_of_interest(canny_edged_image, tetragon_vertices)

Input

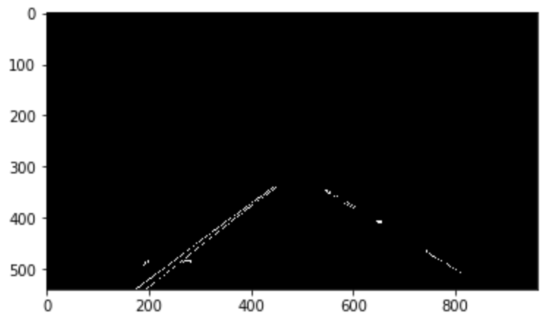


Output

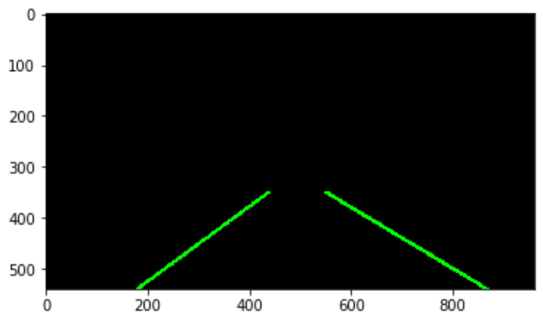


6. Apply Hough transformation to masked Canny edged image in order to get the black image with drawn detected lane lines

Input



Output



In order to draw a single line on the left and right lanes, I introduced alternative function to draw_lines() function named draw_solid_lines(). This function takes multiple detected (or partially detected) line segments detected for left and right line and combines them into single left/right line.

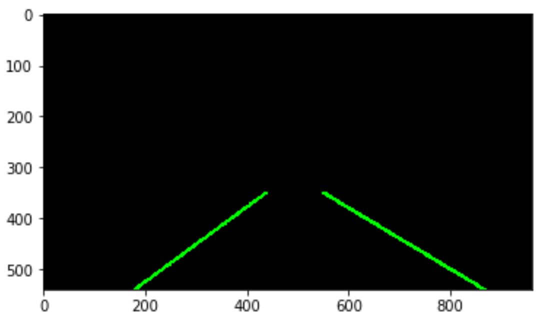
Combination of line segments into solid line is implemented in following way: The goal is to get two lines: one for the left and the other for the right side of the ‘ego’ road lane. The left line has a negative slope, and the right has a positive slope (in the image, y coordinate is reversed. The higher y value is actually lower in the image. Thus sign of slopes is reversed as well). This observation allows me to distinguish whether the each processed line segment belongs to the right tor left target line correspondingly. Having that in mind I group start and finish points of all left and separately right line segments and call my another helping function *build_line_from_segments(segment_start_points, segment_finish_points, y_min, y_max)* for grouped segments points (first for left line group and after that for right line group). This function calculates the averages for X and Y coordinates of provided start and finish points correspondingly that allows to calculate the slope and intercept of ‘average’ line segment which is the base for resulting solid line. By knowing the max and min values for Y coordinate (provided as function parameters) of resulting line I can easily calculate corresponding X coordinates for start and finish points of resulting line.

Note! Trying to slightly filter out ‘phantom‘ detected line segments (caused by changes in light, shadows or color change of lane surface in video **‘/test_videos/challenge.mp4‘**) I introduced the empirically selected min and max values for the slope of left and right resulting lines correspondingly. This filter slightly improved the line detection result for mentioned video file, but has not fixed all the observed issues.

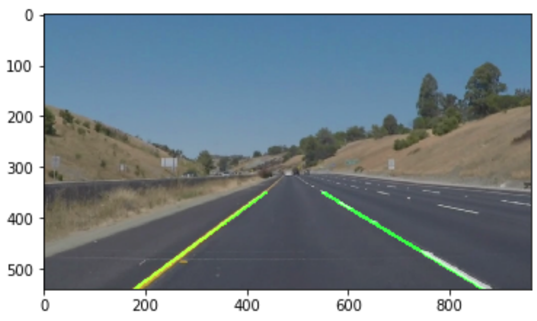
7. Draw detected lines on top of original image (by overlaying lines containing image on top of original image images)

   Function: *weighted_img(lines, img)*

Input



Output



2. Identified shortcomings

One potential shortcoming would be what would happen when the road lane changes it's curvature. In this case calculated slope value is valid only for short part of detected line. This can be observed in **/test_videos_output/cahllenge.mp4'**

Another shortcoming appears when the light conditions are not constant (clear parts of the road are mixed with shadows) or road surface changes it's color suddenly. In this case many 'phantom' line segments are detected which introduce big noise into the line extrapolation' logic and cause unwanted issues in resulting image

3. Possible improvements

To overcome first issue another mathematical approach is required for calculating the shape of desired curve to be detected by Hough transformation.

Second observed issue might be resolved by masking the original image to filter out everything except only the pixels of expected colors of the lane lines (e.g. yellow and white) or converting the original image into another color space, e.g. HSL. and using it as input for pipeline