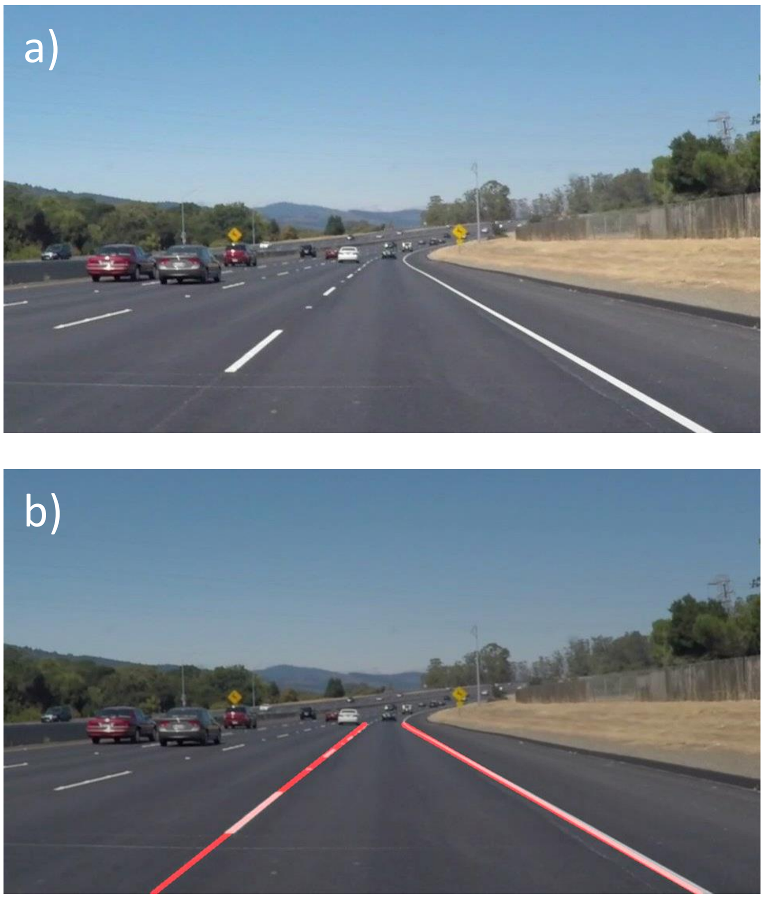
Finding Lane Lines on the Road

Student: Guido Rezende de Alencastro Graça

Goal of the project:

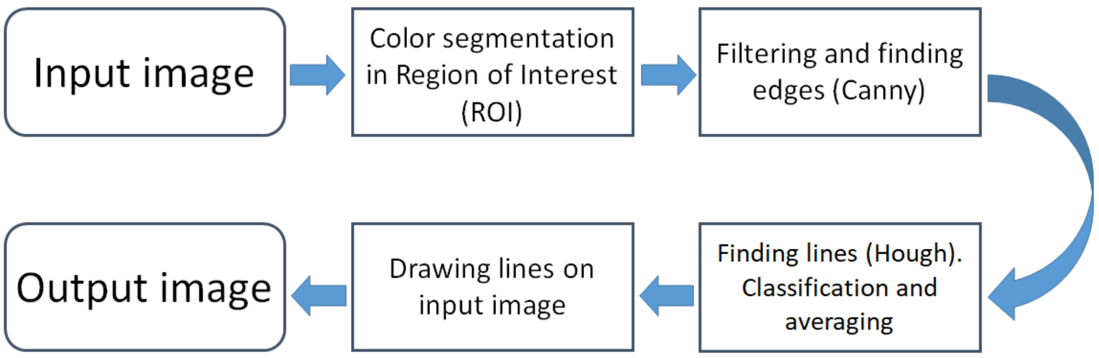
This is report of the first project of the Self Driving Vehicles course on Udacity. This project goal is to build a code for finding lanes on colorful digital images and videos. The code can be found at the Project 1.ipynb file



Figures 1.a and 1.b: Example of input image (a) and desired output image with lane lines highlighed in red (b) (images can be found at test_images/solidWhiteCurve.jpg and test_images_outputs/solidWhiteCurve.jpg)

Code pipeline

The pipeline is shown on the flowchart 1. The code is divided in four main parts, plus opening the input and writing output images files parts. These steps are divided in functions and a main function **lines_identifier()** applies all these functions to an input image.



Input image
Color segmentation in Region of Interest (ROI)
Filtering and finding edges (Canny)
Finding lines (Hough). Classification and averaging
Drawing lines on input image
Output image

Flowchart 1: Code pipeline

The first process is the Color segmentation only in the central area. This central area is a four-sided trapezoid symmetric in the x direction. This polygon is created using the **polygon()** function. The color segmentation is done using the **color_segmentation()** function. This function uses the polygon function to define a region of interest and within this area it applies the color threshold operation. It converts the value of all the non-lines pixels to 1 and the line pixels to 0, and later inverts the selection By default, the threshold values for the background chosen were 10, 10 and 180 in red, green and blue, respectively. This non-symmetry in the blue value was chosen in order to capture the yellow lines. (2.b)

The next step on the code is edges calculation. First, it uses a Gaussian blur using the function **gauss_blur()** for noise removal. Next, it applies the **Canny()** function. As the name implies, it uses the cv2 Canny algorithm. (2.c)

Then, the code uses the function **Hough().** After identifying the lines, the function analyzes if the line is on the left or right of the image and verify its slope. Therefore, the code classifies the lines into the left or right lane line. Then, it averages the two coefficients *m* and *b* of the line. The last function, **draw_lines()** uses the line coefficients of the right and left lane and colors the original image from the bottom up to the middle of the image. (2.d)

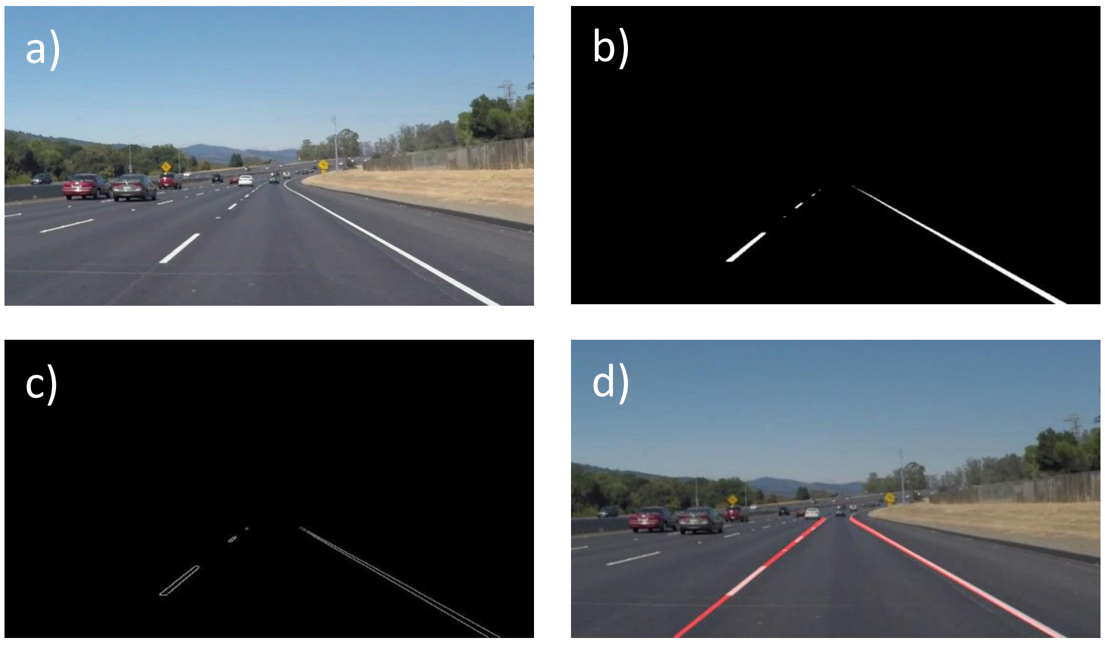


Figure 2: steps of the pipeline. a) Original image b) Color threshold in the cenral area c) Canny edges detection d) Output image

The code also implement the lines_identifier function to a video file. The function **lines_on_video()** reads a video file and applies lines_identifier frame by frame. It saves the resulting video on the folder test_video_outputs with the same file name as the input.

Overall rating of the code and points of improvement

In general, this algorithm works nice in the test samples images. However, it must be emphasized that these test samples are well-behaved images, with the lane lines having a strong contrast against the road pavement and with few cars around the center of the image.

The previous versions of the code had to be improved in order to do a decent result on the challenge.mp4 file. The color threshold is needed for filtering out the car hood in the image. It works fine because the car has a dark color. However, if the car was yellow or white, the color threshold would not be enough to filter out the car hood. Further algorithms shall be added. This could be solved by adjusting the region of interest drawn by the polygon more to the middle of the image. However, as the car hood may vary a lot from image to image (considering different sources of images), it is necessary to adjust the polygon for each camera position and car hood.

Another problem with the challenge.mp4 is that there is a moment in which the road changes from asphalt to concrete. This change in the road pavement changes the color of the background from black to gray. This makes the contrast of the lane lines with the road not so clear and the program fails to detect far lane lines between seconds 0:04 and 0:07. This might be solved with proper image processing before applying the color threshold. Filters may be applied that sharpens edges or enhance contrast.