Advanced Lane Finding

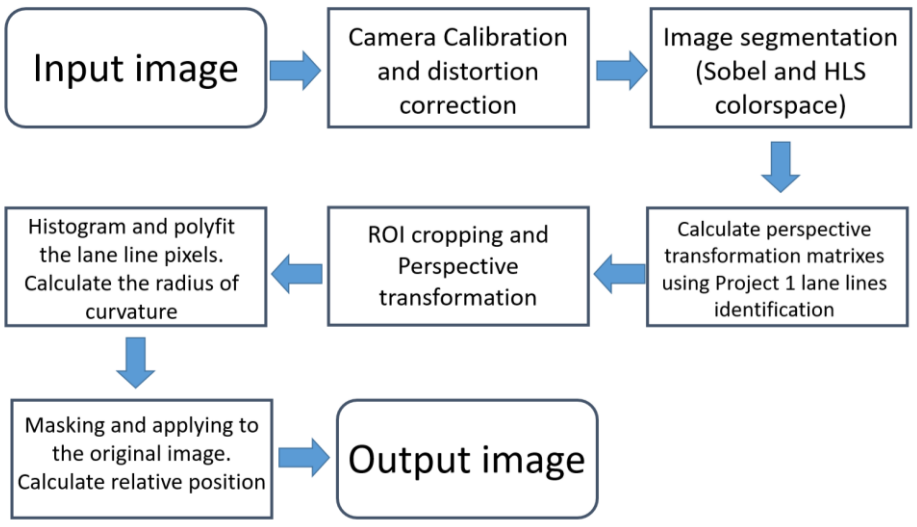
Student: Guido Rezende de Alencastro Graça

Project Goals and Code Pipeline:

This is the report of the second project of the Self Driving Vehicles course on Udacity. This project goal is to build a code that is able to:

1. Calibrate a camera based on a series of chessboard pictures.
2. Find lane lines not only using color threshold on an RGB color space, but using also HLS color space and different techniques of color gradient (Sobel's gradient).
3. Apply an up view perspective transformation on the image.
4. Interpolate the right and the left lane lines pixels with a second degree polynomial fit each.
5. Calculate the mean radius of curvature and the relative position of the car with respect to the lane.
6. Display results on the original image.
7. The code must be able to read images or videos as input.

The code pipeline is shown on flowchart 1. The two main functions used in the pipeline are **main_function_image** and **lines_on_video**. The first one applies the algorithm to a single RGB image whereas the second one applies the algorithm to a video. The code can be found at the Project 2.ipynb file. Each step on the main_function_image is described below:



Flowchart 1: Code pipeline

a) Camera Calibration

The camera calibration is done by the **calibrateCamera** function. Using the glob library, the function adds all the camera calibration images paths to a list (variable **images**). Then, it finds all the Chessboard corners of each image and adds them to another list (variable **imgpoints**). Using the corner list, the program uses the cv2.findChessboardCorners to find the undistortion parameters (variables **ret,mtx,dist,rvecs,tvecs**). Using the undistortion parameters

found using calibrateCamera(), the function **undistort** undistorts an image. Figure 1a shows a chessboard image used for the calibration process and Figure 1b shows the undistorted image.

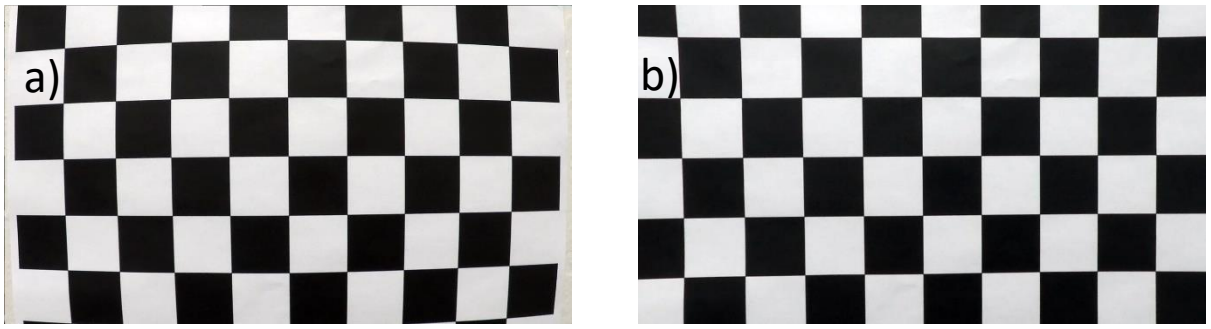


Figure 1: Camera Calibration and image undistortion algorithms. a) Original Image; b) Undistorted image.

b) Image segmentation

The image is segmented using two approaches: The HLS color space approach and the Sobel in the x direction approach. The first is calculated using the **hls_thresh** function. It receives an image and creates two images, one with the HUE channel and other with the Saturation channel. After adjusting the images values to the ranges 0-255, the function applies the desired threshold interval for these two channels. It was used (20,100) on the HUE channel and (170,255) on the Saturation channel. Then, these binary images are combined using Boolean interception operation.

The second thresholding function is the **abs_sobel_thresh**. It calculates the Sobel gradient in either x or y direction. The thresholding used Sobel X image and used the (60,100) interval. At last, these two binary images are combined using the **combine_binary** function, which simply applies a join Boolean operation on the input images. The thresholding process is detailed in Figure 2.

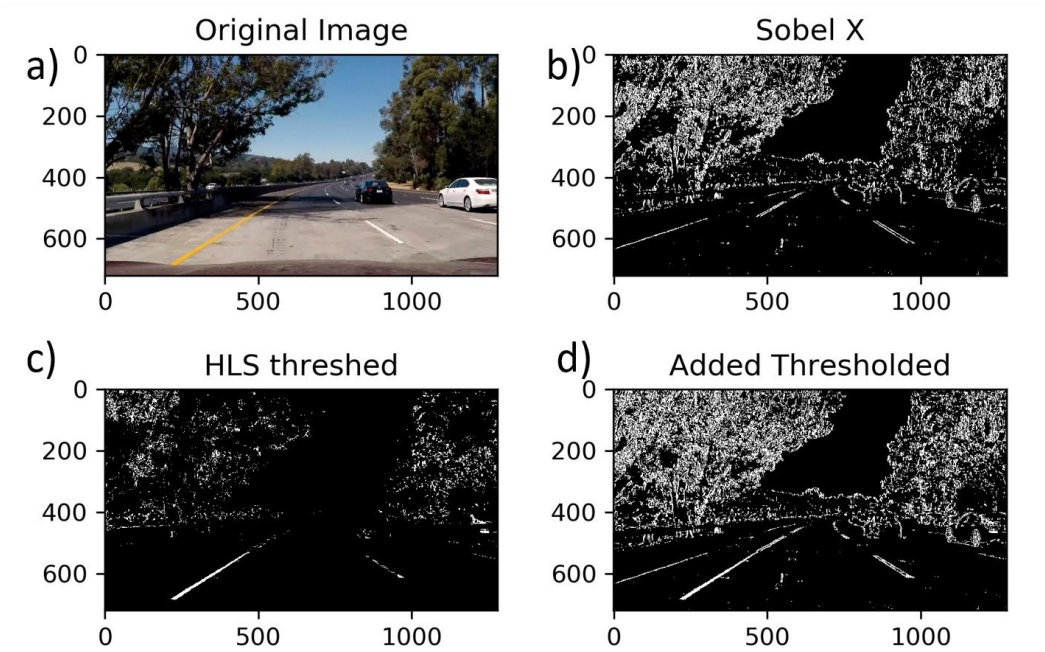


Figure 2: Thresholding process. a) Original image; b) Sobel X thresholding; c) HLS thresholding; d) Combined thresholded image

c) Perspective transformation

The key step to do the perspective transformation is to find the right perspective transformation matrix and its inverse. However, in order to find these matrixes, it is necessary to define the polygon vertices before and after the Perspective transformation. The original vertices positions must be well defined, otherwise the resulting perspective transformation is not parallel to the ground and the resulting transformed lane lines will not be parallel. With this in mind, the idea of the **get_Matrixes** function is to apply the lane lines detection algorithm used on Project 1 to find the two lines that best adjust to the lane lines. Using these two lines, the **perspective_Matrixes** function calculates the perspective transformation matrixes using the concept that these two lines must be parallel on the resulted transformed image. Before applying the perspective transform on the image, the **polygon_masking** function is applied to the thresholded image. It crops the image to focus only on the center, where the lane lines usually are visible. The perspective transformation is detailed in Figure 3.

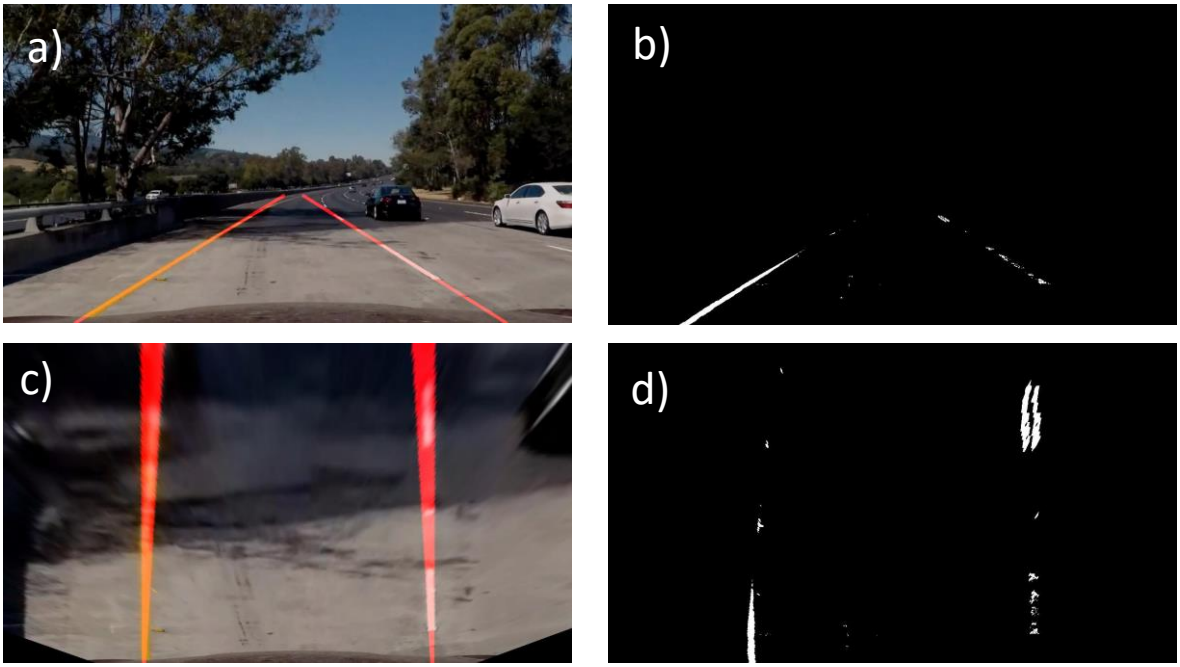


Figure 3: Polygon cropping and perspective transformation. a) Lane lines interpolated by a line (Project 1); b) Polygon cropping applied to the thresholded image; c) Perspective transformation on the original image (Note that the lane lines are upward and parallel); d) Perspective transformation on the polygon cropped image.

d) Interpolation and measuring the radius of curvature

Now, with the up-view of the binary image, the **histogram** function divides vertically the image in two and calculates two initial second degree polynomial fit: one for the left and other for the right lane line. Using the sliding windows technique, the histogram function divides the image into nine parts along the y direction. It then calculate the histogram of nonzero pixels along the x direction for the first two parts and finds two peaks: one at the left and other at the right. Using a margin hyperparameter, it defines two windows at the bottom. Then, it slides into the next part and reevaluate the center of the windows using again the histogram technique. This process is repeated for all the nine parts. Using the identified lane line pixels, it colors the pixels using red for the left lane line and blue for the right, then interpolates and find an initial polynomial fit. Using the polynomial fit found at the histogram function, the main function applies the **Interpol** function. Using the found interpolation parameters, it reevaluate the lane line pixels within a margin and recalculate the polyfit coefficients.

To measure the radius of curvature, the polyfit coefficients dimensions were adjusted for meters instead of pixels. For that, the x direction converting parameter was calculated using the hypothesis that the lane width is 3.7m and in the y direction it was used the hypothesis that the lane length caught by the camera was 30m. It must be said that while the x conversion was a good estimate, the y estimate is rough, so it incurs on errors on the radius of curvature measuring. It was used the following equation for measuring the radius

$$R_{curvature} = \frac{\left(1 + \left(2\,\frac{m_x}{m_y^2}\,A\,y_{eval} + \frac{m_x}{m_y}\,B\right)^2\right)^{\frac{3}{2}}}{2\left|\frac{m_x}{m_y^2}\,A\right|}$$

Where *A* and *B* are the polyfit coefficients measured in pixel space. The m_x and m_y are the dimension adjusting coefficients in the x and y direction respectively. The radius of curvature is calculated for both lane lines and then the function calculates its mean. Depending if the radius is positive or negative, the algorithm classifies it into right curve or left curve.

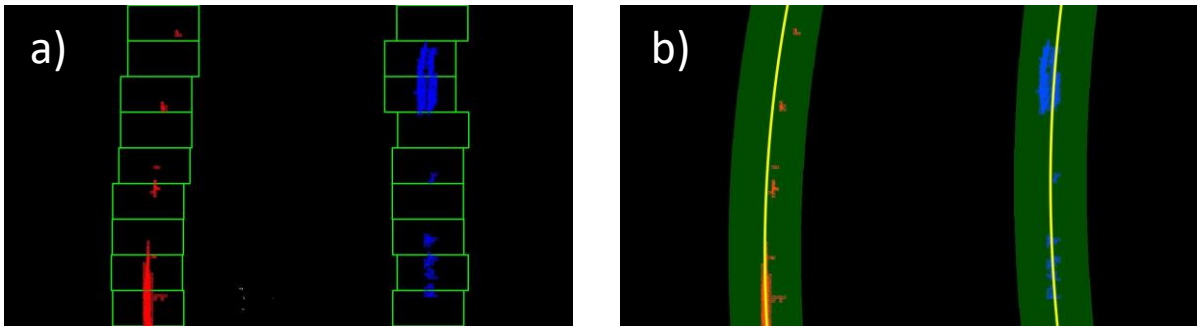


Figure 4: Histogram (a) and interpolation (b) segmentation of the image.

e) Displaying the final results

Using the figure 4b as a mask, the main function applies the inverse perspective transformation on the mask and combines the original image and the newly transformed mask. In addition, it displays the radius of curvature on the image. The position of the center of the vehicle is also calculated. It is measured using the distance between the center of the image on the x direction with respect the center of the lane. The center of the car is considered to be exactly on the middle of the camera, whereas the center of the lane is considered to be on the

midpoint between the lane lines interpolation. On the images, the center of the car is marked as a black dot, while the center of the lane is shown as a white dot (Figure 5).

Figure 5: Lane marked on a test image

f) Applying to a video

Similar to the main_function_image function, the **lines_on_video** algorithm applies the camera calibration and calculates the perspective transformation matrix for the first frame, and then apply each of these steps in each frame. However, as not all the frames are well behaved, there is some evaluating on the calculated polyfit parameters and the radius of curvature. The main function keeps record of the last found parameter and calculates the mean relative error. If the difference is significant, the algorithm uses the last frame parameters and increases the error count by one (variables **left_error**, **right_error** and **radius_error**). If the error count in a streak reaches a certain value, then the new calculated value is accepted and the error counting returns to zero. If an acceptable result is found after an error, the error counting also returns to zero. This is used so that minor bad frames do not compromise the sequence of frames. In addition, if the streak of errors reaches six for the polyline *A* coefficient or three for the radius, the code will accept the new value, in order to readjust the reference value. Otherwise, the parameters would diverge from the original reference value as the line lines naturally changes from frame to frame. Also, the radius of curvature is displayed only each two frames, so that it is more soothing for the viewer.

Overall rating of the code and points of improvement

In general, this algorithm works nice in the test samples images and during the majority of the video's frames. However, as suggested on the course, a more robust and light algorithm would use the anterior frame polyfit region for looking for lane line pixels, instead of calculate the histogram frame by frame.

Two common types of errors in the image processing were found: **shadows and dark stains on bright road**. Areas with shadow, the HLS thresholding alone is not enough for detecting lane lines and it is needed to use the Sobel X thresholding technique as shown in figures 2b and 2c. However, as the Sobel X technique detects any changes in intensity regardless of the original

colors, it detects much noise of tire stains on a concrete road, as seen in figures 6a and 6b. It was tested noise reduction techniques such as the binary morphological operation erosion and dilation (and its combination, opening and closing). However, not only it affects the noise but also the lane line pixels, mainly the one farthest of the vehicle, as they are smaller and have few pixels. The solution used was to use a higher low threshold value (60 instead of 20), although the binary image has now less information on farther lane line pixels.

Even though the radius of curvature is calculated in the algorithm, its value looks off from the expected, seeming smaller as expected. As said before, the pixels to meters conversion value in the y direction is a rough estimative, as the real distance may vary. Not only but also this value depends on which road the vehicle is. In addition, it must be emphasized that the radius of curvature calculation is sensitive on the interpolation coefficients, and these coefficients are sensitive to the pixels used on the polyfit calculation, so small changes in the used pixels imply on significant changes. Similar effect occurs to the relative position of the vehicle, which is very sensitive to the interpolation fit.

Figure 6: Sobel X and its noise detection. a) Original image; b) Sobel X thresholding with (20,100) thresholds

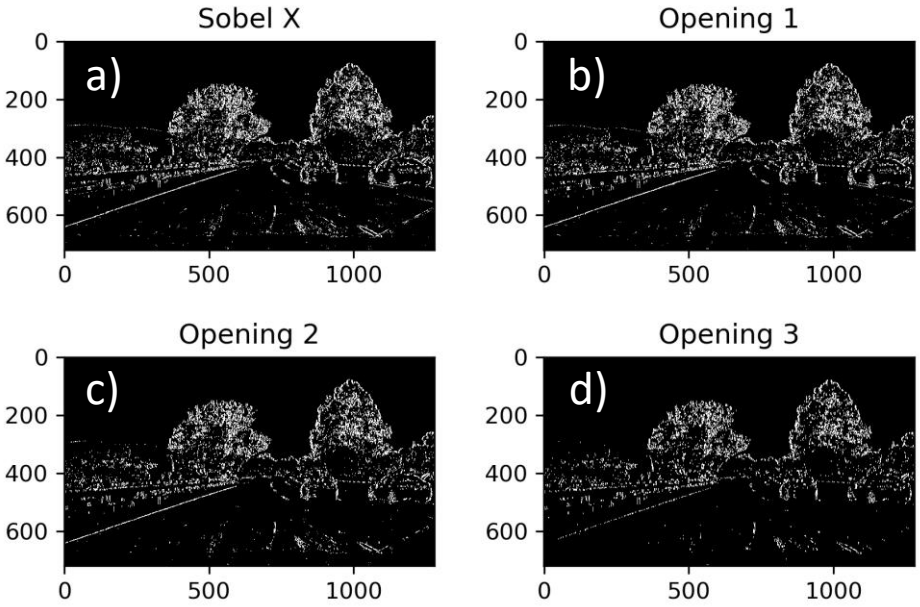


Figure 7: Noise removal algorithm: Opening. a) Original binary image; b, c and d) Opening with one, two and three iterations respectively.

It must be said that the HLS works well on yellow lane lines because yellow is well defined on the HUE channel. However, as the pure white color is not on the HUE color channel, it is needed other channel, like the saturation. Yet, the saturation channel is prone to detect shadows, like the tree shadow on figure 8d, so relying only on saturation channel will induce in

noise detection. The solution was to use both H and S channel thresholding and use the Boolean interception operation for not including the shadow noise, even if it leads to not detecting so well the white lanes. It is likely that a thresholding also on the lightness color channel may help solving this issue.

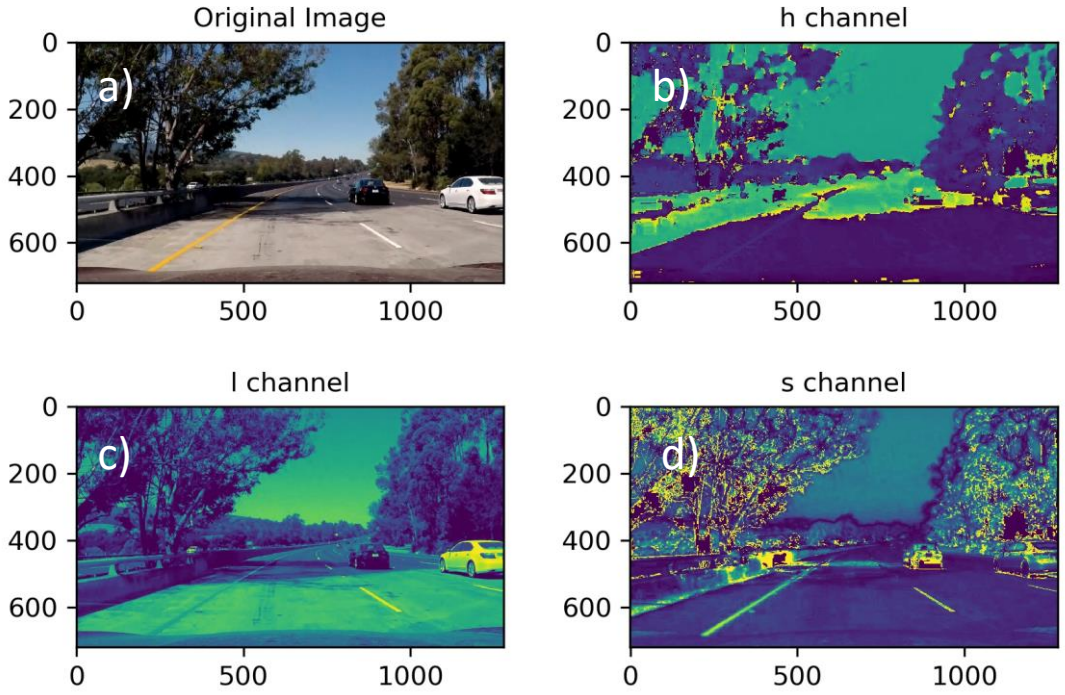


Figure 8: Original image and HLS space channels

Upon testing the code on the challenge_video.mp4 file, it was seen that the code inaccurately detects vertical changes of road pavement as lane lines (Figures 9a and 9b). This issue may be solved using more robust color thresholding, for search only pixels either white or yellow in RGB or HLS color spaces. This erroneously lane detection is due to the intensity's gradient algorithm, which sees the changes in intensity independent of the original colors. Combining or even applying Sobel thresholding to a color channel as HUE or saturation may solve this error. In addition, when testing the algorithm on the harder_challenge_video.mp4 file, it was noticed that the algorithm most likely would fail when a motorcycle is between the lanes, as shown in Figure 10a, because the code will detect the motorcycle pixels and include them on both histogram and interpolation functions. Furthermore, there are times where the lane line is failed to be seen, like under harsh sunlight, leaves covering the lane line or even on sharp turns where one of the lane lines is hidden from view (Figure 10b, 10c and 10d). In these cases, the algorithm is assured to fail. Solutions for these cases are advanced camera detection techniques, which are able to recognize the motorcycle as another vehicle and not include it on the analysis. About the vanishing lane line, it is necessary a robust lane line algorithm detection that keep records of the lane position and even if it fails to detect a lane line it does not raise an error, but expects the lane to reappear soon after.

Figure 9: Attempt on the challenge_video. a) Original Frame; b) Erroneous lane detection.

Figure 10: harder_challenge_video troublesome frames. a) Motorcycle in the middle of the lane; b) leaves covering the lane line; c) Intense sunlight on white lane; d) Right lane line disappears from the video