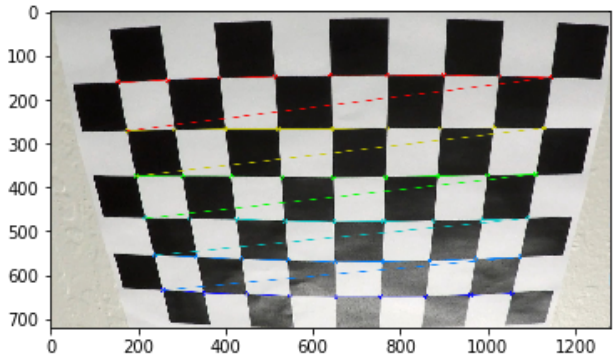
Advanced Lane Finding Project

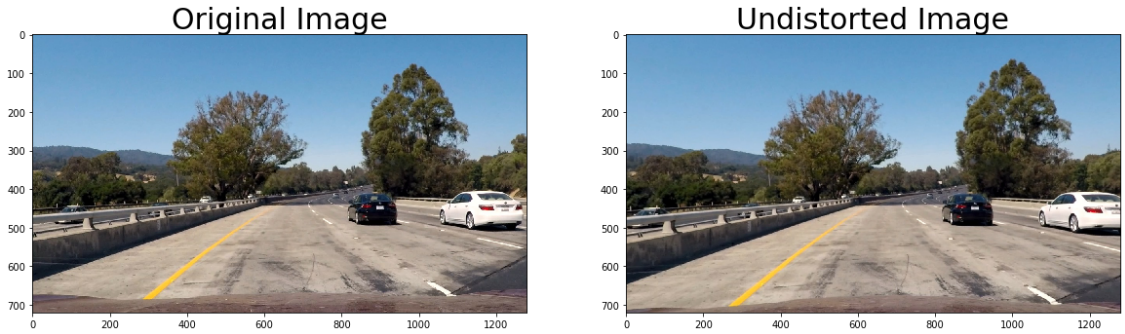
Note: There are two main files for the program the first is named “Advanced Lane Lines Pruebas” and the second is named “Advanced_Lane_LinesG1”. The first contains all the code required to produce an image with the desire characteristics and code to plot images. The second code is equal to the first except that all the print commands and plot commands were eliminated in order to make the code more readable.

Camera Calibration

1. The code for this step is contained in the section Camera Calibration of the “Pruebas” IPython notebook. I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Then the image is converted to gray scale and the corners (image points) are found with the findChessboardCorners function. This result is obtained:



Next, with the image points of every image the matrix of the camera and the vector of distortion is calculated, using the cv2.calibrateCamera() function. I applied this distortion correction to every image using the cv2.undistort() function and obtained this result:

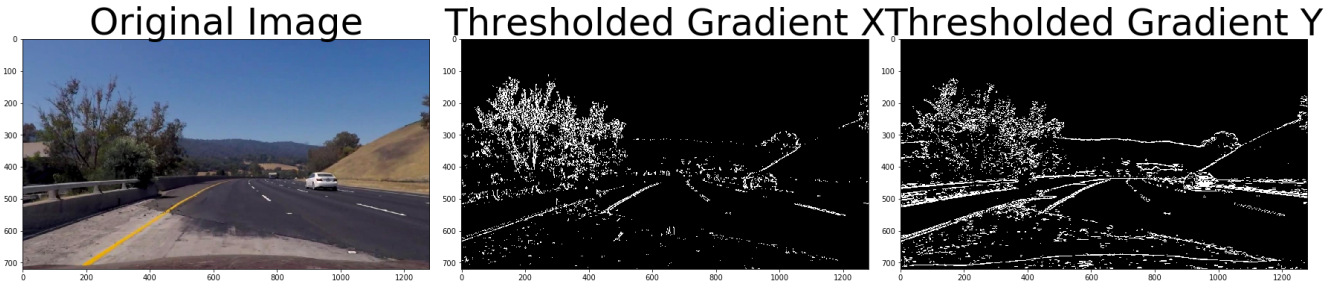


Pipeline (single images)

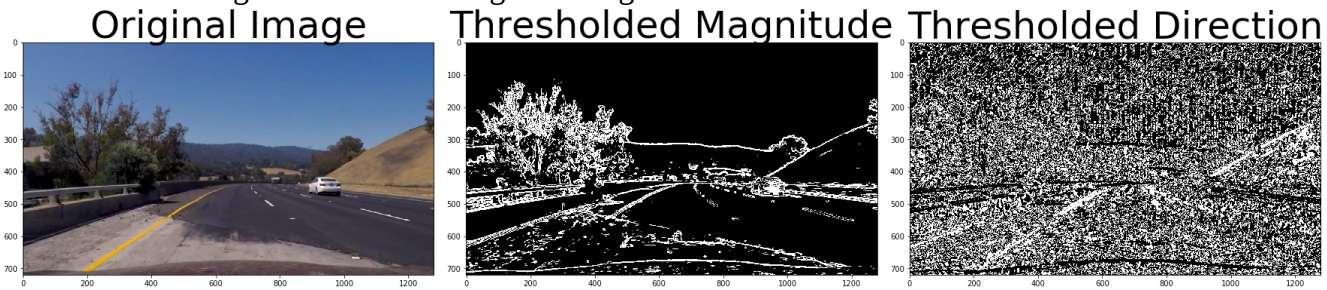
1. Once the undistorted parameters were found, for every image to be tested the he cv2.undistort() function was applied. Here is the example code:

img = mpimg.imread('test_images/test7.jpg')
image = cv2.undistort(img, mtx, dist, None, mtx)

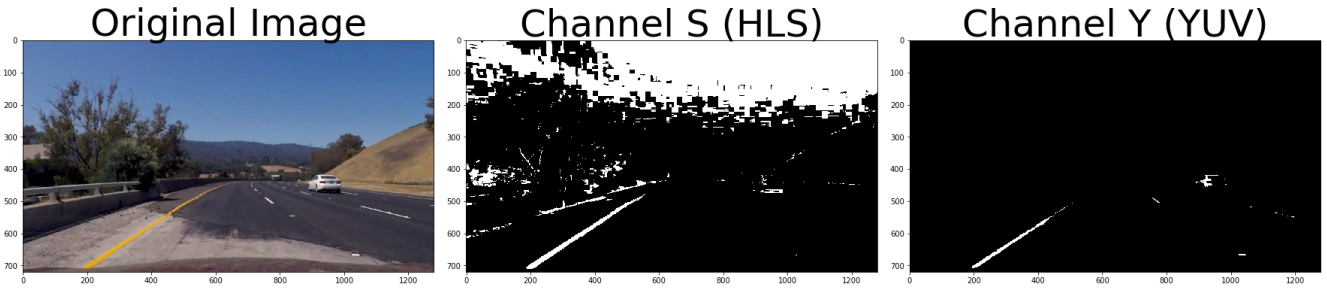
2. In order to create a binary image, many forms of gradients and color space were tested. The gradient in X and Y.



The mean of both gradients and the angle of the gradient.



For color space the HLS and YUV were selected.



3. For the perspective transformation an image with strait lines was used to select the rectangular area. The code uses as inputs an image (img), as well as source (src) and destination (dst) points. I chose the hardcode the source and destination points in the following manner:

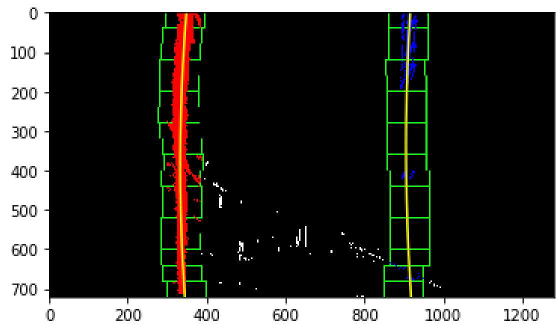
sp1 = [600,450]
sp2 = [235,720]
sp3 = [1075,720]
sp4 = [680,450]

dp1 = [350,0]
dp2 = [350,720]
dp3 = [900,720]
dp4 = [900,0]

p1 = (300,0)
p2 = (300,720)
p3 = (1000,720)
p4 = (1000,0)

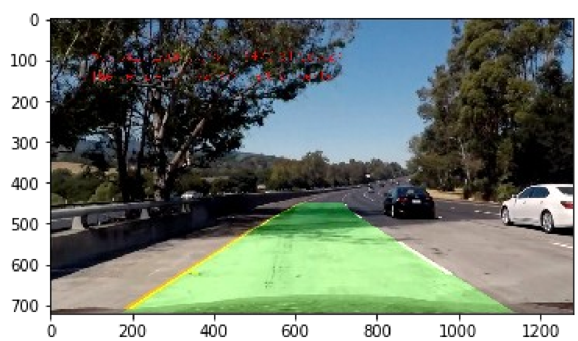
I verified that my perspective transform was working as expected by drawing the src and dst points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

4. In order to find lane pixels the sliding windows method was used. Due to the perspective transformation the approximate position of the lane lines was known so a full histogram of this position was calculated. The maximum values of the histogram for the left and right side were calculated, with this values a window was made to search pixels. If an amount of pixels were found a new mean was calculated and the next upper window was center using this mean value. In every search the nonzero pixels in x and y were identified and saved. Then with an array of all the positions of nonzeros a polynomial fit was made.



5. The curvature of the image was calculated using the equations analyzed in the videos:
left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])2)**1.5) / np.absolute(2*left_fit_cr[0])**
right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])2)**1.5) / np.absolute(2*right_fit_cr[0])**

In order to calculated the center of the image. First the mean X center of each line was calculated. Then the X center of the right and left lines were added and divided by two. Finally the distance from the center was calculated by subtracting the position from 0.5*image width.



Discussion

The problem with all the images was the noise. If we want a better result the noise needs to be eliminated. I tried with erosion, dilatation and another filters but i can conclude that a better option is to select another color space and reduce the noise produced by the gradient. In order to improve the quality of the procedure a better image preprocessing should be made also a new color space could be used like CIELUV and CIELAB.

The pipeline could fail under different circumstances of lighting. For example when the sun is hitting directly the road, when a cloud is passing by or with the shadow of a tree. Also when a frame of the camera has few segment of lane line to identify (here in Mexico the lines are fuzzy in some streets), when the street has different colors or when the sun hits directly the camera. The sliding window method could fail when the image is noisy.

In order to improve the quality of the pipeline is necessary to find a better method to segment the images in Color or Gradient under a great variety of lighting conditions. Also a dynamic segmentation could be implemented when a change in lighting is detected. In the case of perspective transformation it would be useful a programatically method to detect a rectangle and assign automatically the best points to do the transformation. A sliding window method that could identify differences between the two lines and detect when one of the lines is so different from the other and select the best alternative.