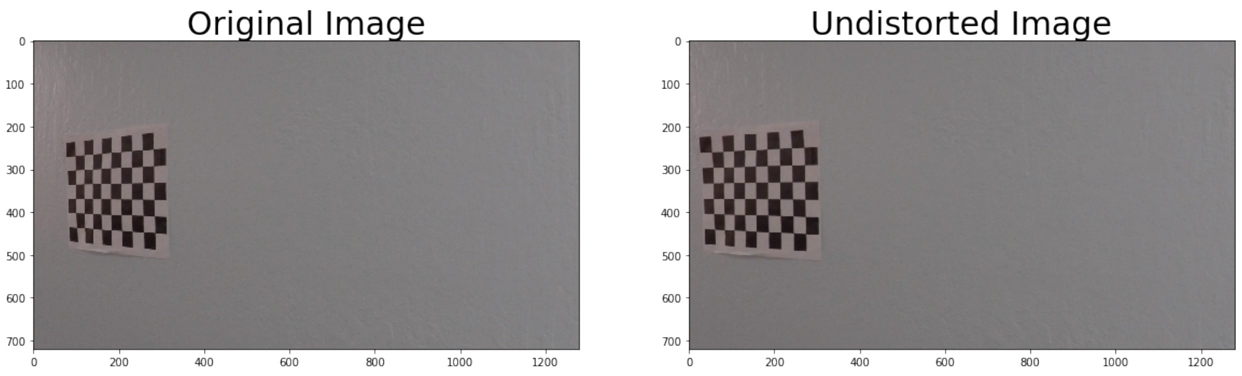
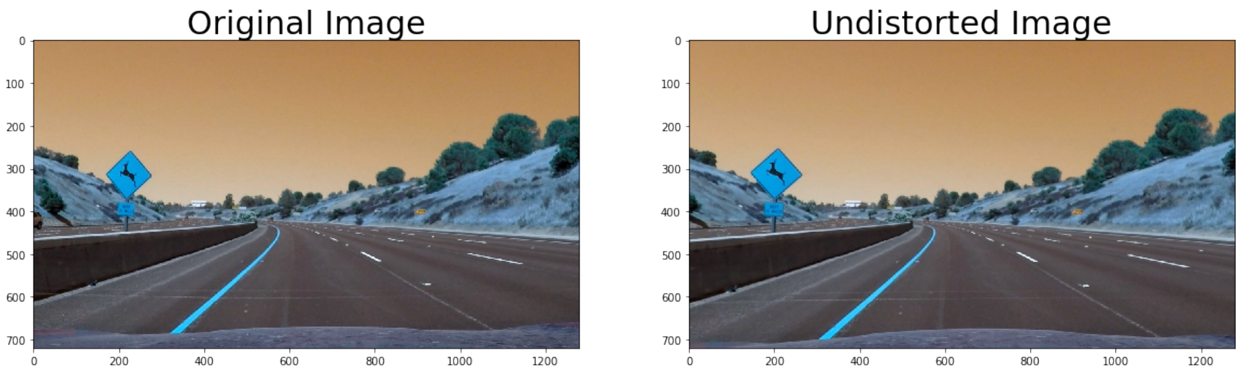
Advanced Lane Finding Hardik Patel
Udacity – Self Driving Car – Term 1

Cameral Calibration

The code of Camera Calibration in in code cells 2, 3 and 4. The provided Camera Calibration images were used to find corners in each image and append to an array, using the cv2.findchessboardcorners function. The arrays (objpoints and imgpoints) were used to find the correct camera matrix and the distortion coefficients, using the cv2.calibrateCamera function. These values are then used to undistort the images, using cv2.undistort function. Here is an example of the matrix used on a test image –

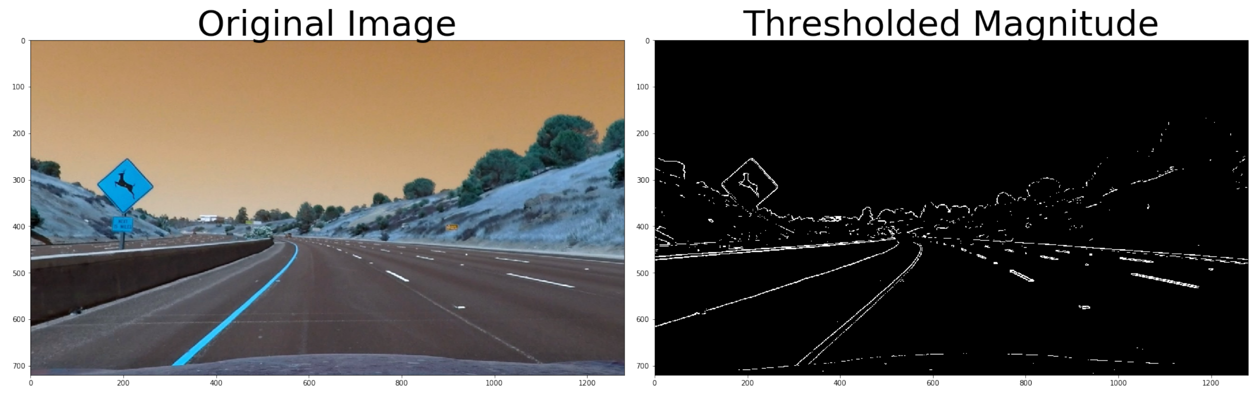




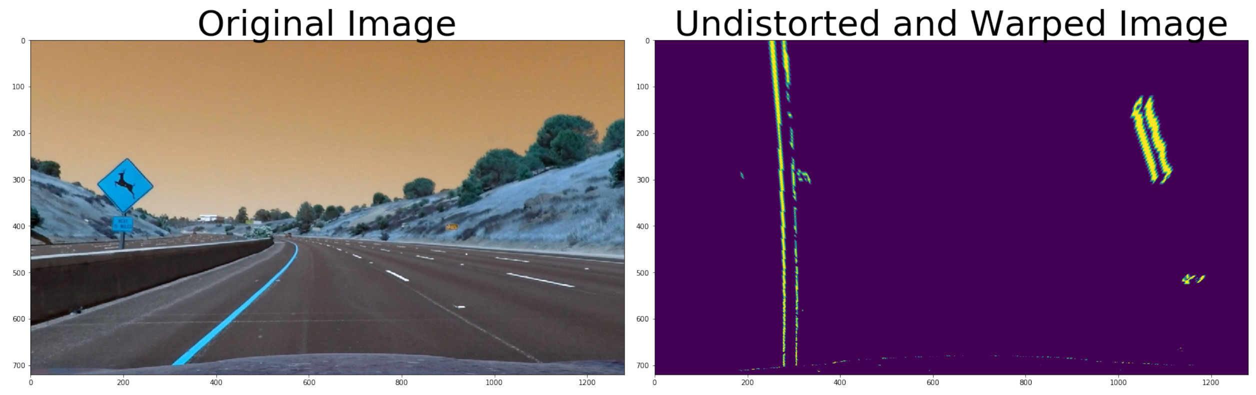
Pipeline (single images)

Through code cells 7 to 16, various different methods of calculating the thresholded binary image, were tested. Absolute, magnitude, direction methods were tested by themselves, the combination of them all and different pairs were also tested. The combination gave good performance on the video pipeline; however, it missed its mark at the parts of the video that had brighter roads. Hence, the LUV and Lab combinations were tried and, in the end, combined image of B and L channels were found to perform the best on the brighter sections of the road.

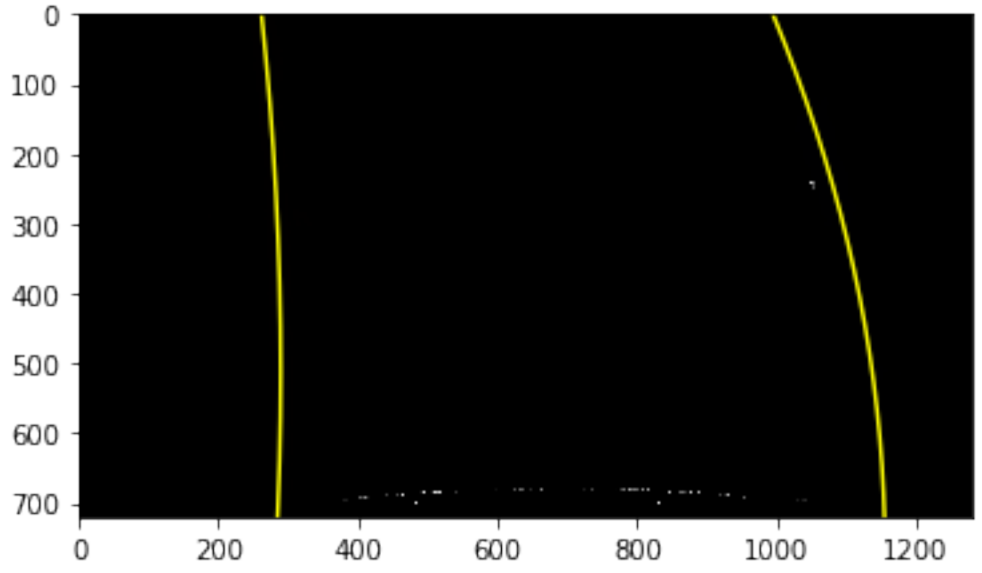
The values for minimum and maximum thresholds were found after trying and testing with different values.
Here is an example image of the thresholded binary image (Other pictures are in the “output_images” folder) –



The next step was to perform perspective transform (cell 17) on the threshold binary image, using, cv2.getPerspectiveTransform and cv2.warpPerspective function. The main bottle here was to find the correct values for src and dst. After testing various values, a set of values were found, that worked properly on curved roads as well. Example image of a warped image –



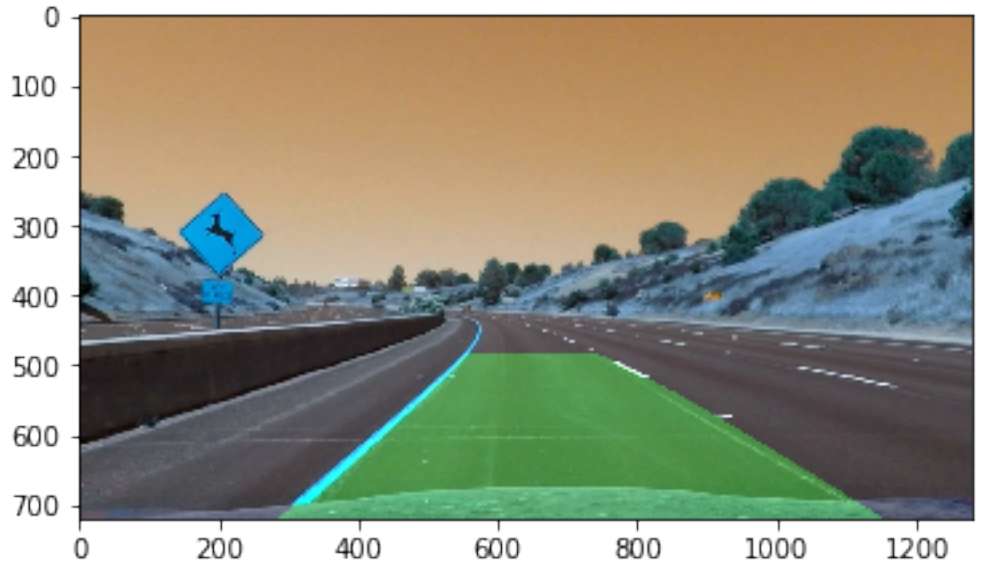
To find the lane-line pixels, the provided code from Udacity was used. In cell block 20, the variables leftx, lefty and rightx, righty hold the left and right line pixels. And a second order polynomial is fit to both, left and right lines. Example image of the lines –



The radius and curvature are calculated in the video pipeline, towards the end. For the radius, the average of the left and right radii is used, and for the position, the center of the car is the average of the two lane lines, and distance is calculated relative to the center.

The values of the radius and position are displayed on the video.

Finally, the lanes identified and plotted back on the original picture -



Pipeline (video)

The same pipeline was used for the video, applying the process on each frame, using moviepy.editor. The file is provided in the zip.

Problems

The biggest problem that I faced was finding the correct method of thresholding to correctly identify lane lines. As mentioned, I tried all the methods with different threshold values. Most methods worked fine on the video, however, the brighter areas of the road gave problems. I think there still is room to improve the method.

Another problem that may occur in real life, is that the pipeline is not optimized. I did not take advantage of the class Line() and it takes 2 – 3 minutes to process the video. Hence, in real life on a highway if a car is moving at 75 mph, will the algorithm be able to analyze the lines before the car has already driven through? That is something that will be interesting to see.