Advanced Lane Finding Project-Avni Sharma

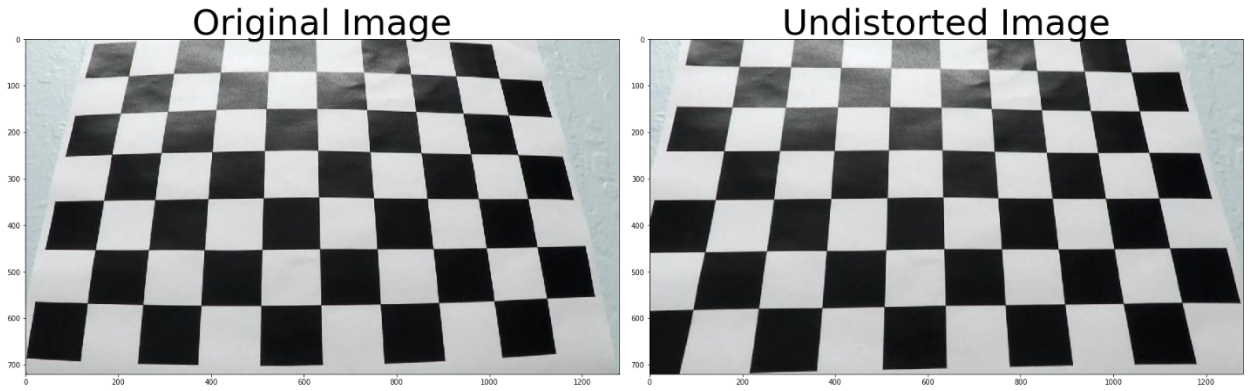
OBJECTIVE:

The goals / steps of this project are the following:
* Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
* Apply a distortion correction to raw images.
* Use color transforms, gradients, etc., to create a thresholded binary image.
* Apply a perspective transform to rectify binary image ("birds-eye view").
* Detect lane pixels and fit to find the lane boundary.
* Determine the curvature of the lane and vehicle position with respect to center.
* Warp the detected lane boundaries back onto the original image.
* Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration

The code for this step is contained in the second code cell of the IPython notebook named Advanced_lane_finding.ipynb. 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. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

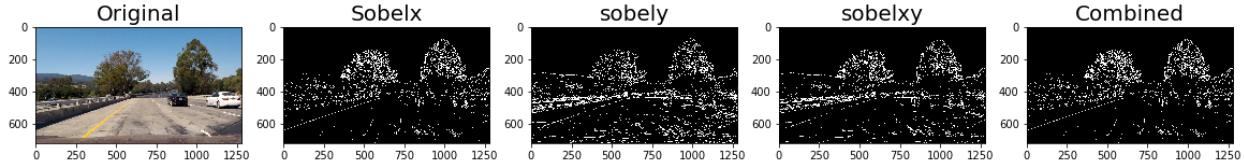
In the third code cell I calculated the required calibration matrix and distortion matrix using cv2.CallibrateCamera. My cal_undistort function does the required undistortion for the original image.



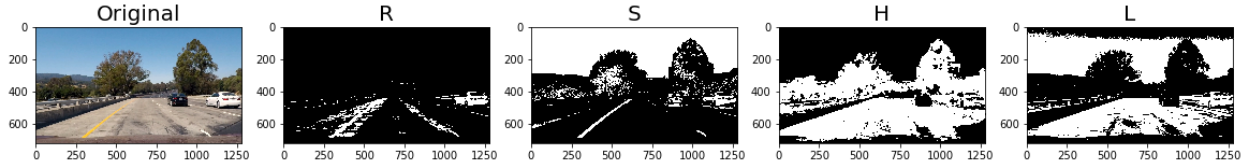
Pipeline (single images)

1. Example of the undistorted image from the test images.

2. Different sorts of gradient thresholds applied to test image to choose from.

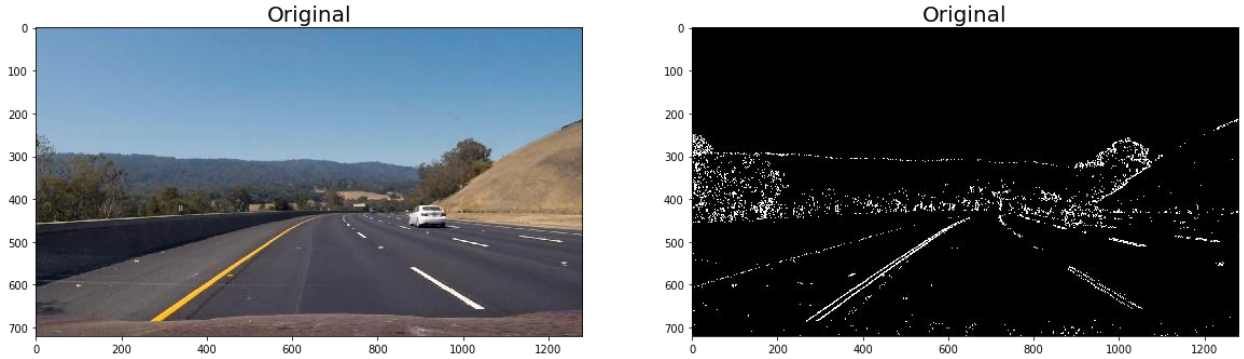


3. Test image being separated into different color space and color channels to choose the best from.



Found out that R and S channel along with sobelx is the best combination to work with.

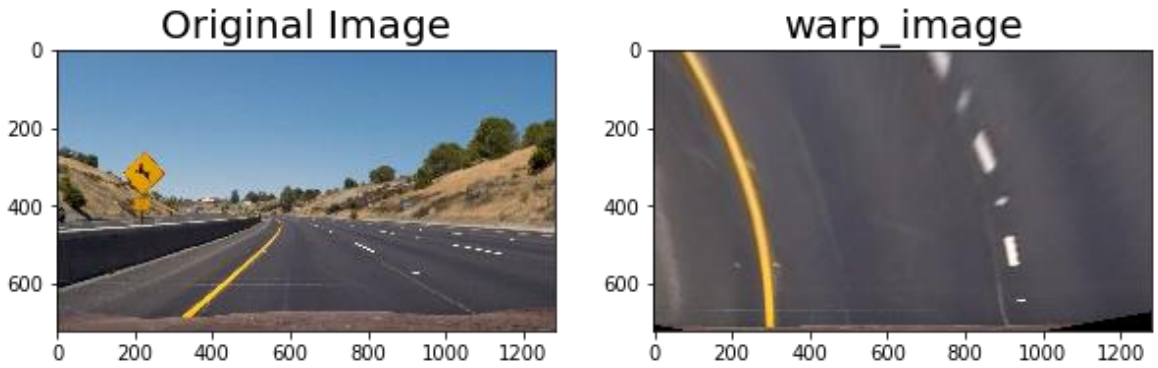
4. Example of an image been created using color transforms, gradients or other methods.



5. Applying Perspective transform:
   The function warp_img applies the required perspective transform. The following source points and destination points were chosen by eye balling the image:

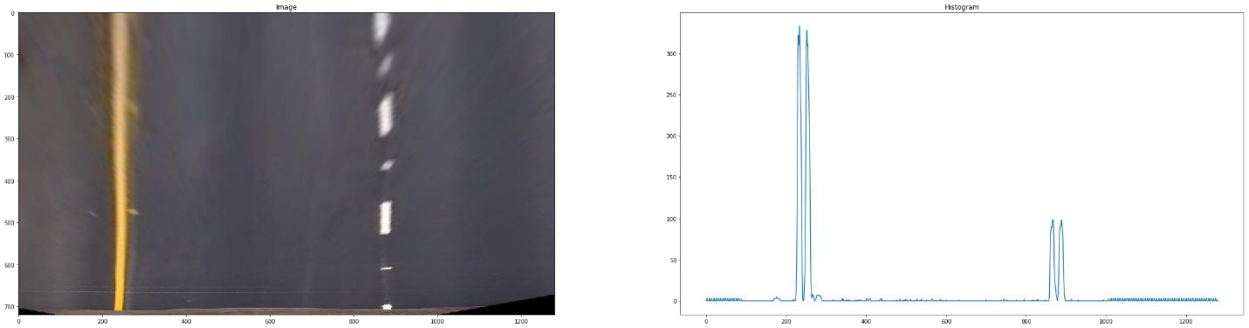
   src=np.float32([[220,720],[1110,720],[722,400],[570,400]])
   dst= np.float32([[320 ,720], [920 ,720], [920 ,1], [320 ,1]])

Below is the example of a warped image:

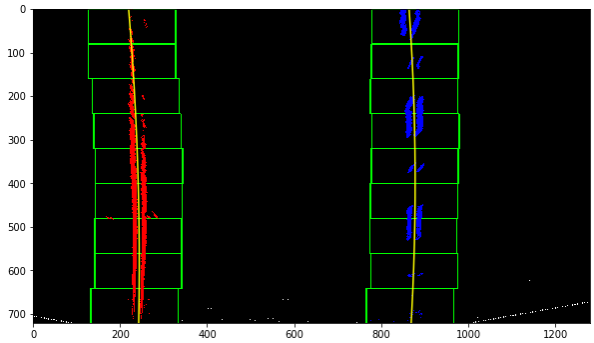


6. Applying

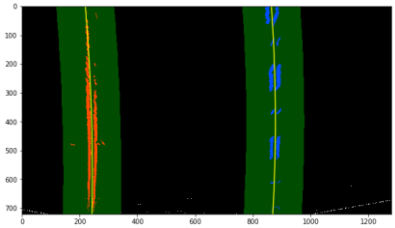
   Histogram function to obtain the two peaked on the binary warped image:



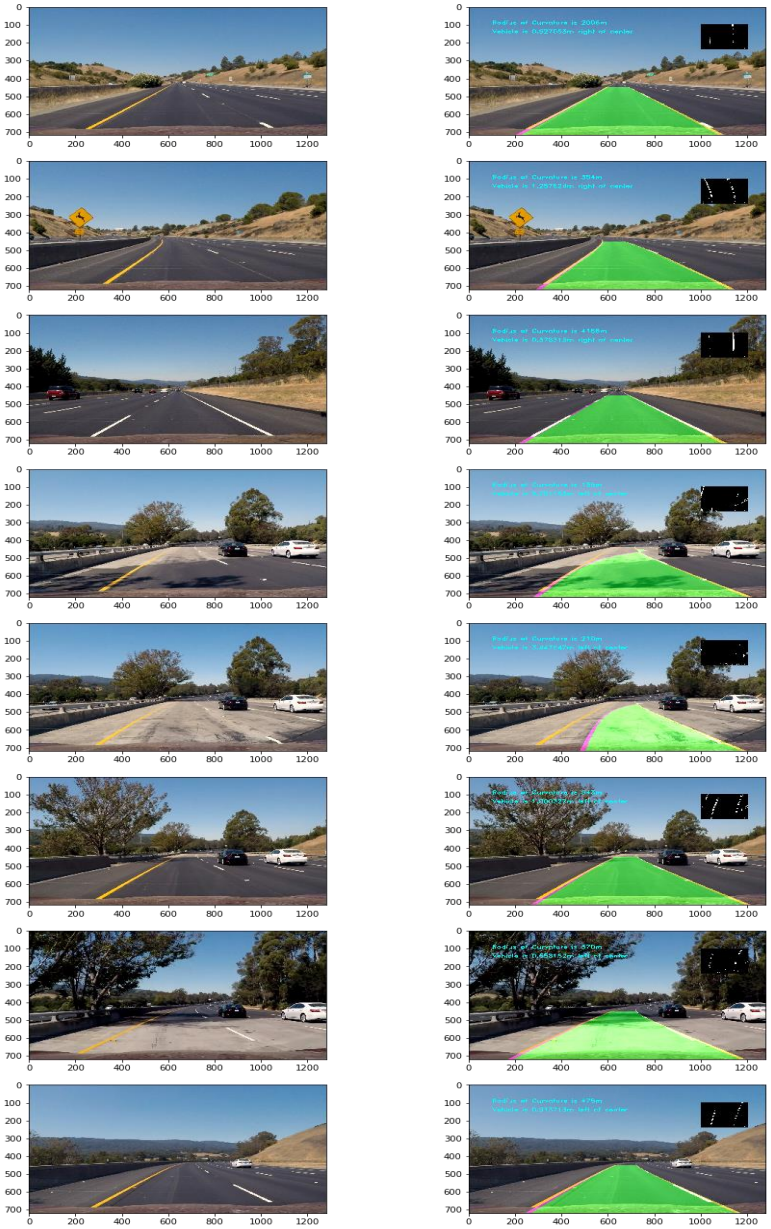
7. The window_search function applied the sliding window technique to the warped binary image after apply the histogram function.



8. And the fitting a polynomial after the sliding window search:



9. The images are then fed to the pipeline and the outputs are shown below:



FINAL VIDEO OUTPUT

The final project video output is in its respective folder. The pipe line suffers errors as soon as the video reaches an area with shadow however the results are not at all catastrophic. And the lane recognition is

perfect in other areas. You can also view it here:

Harder video challenge: In its respective folder named harder_challenge_video_output.mp4 you can also view it here: