Advanced Lane Finding Project

Contents

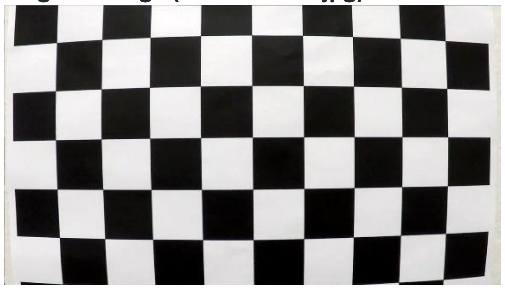
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the **second** code cell of the IPython notebook located in “**CarND-Advanced-Lane-Lines"**

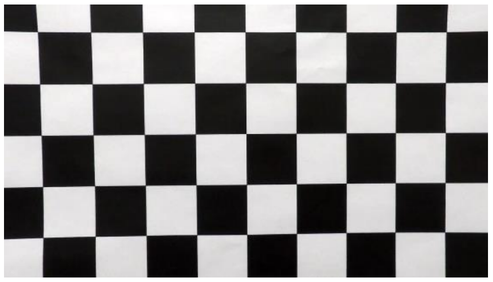
I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. The chessboard grid size is 9 rows and 6 cols. 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.

I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort ()` function and obtained this result:

Original Image (Calibration1.jpg)

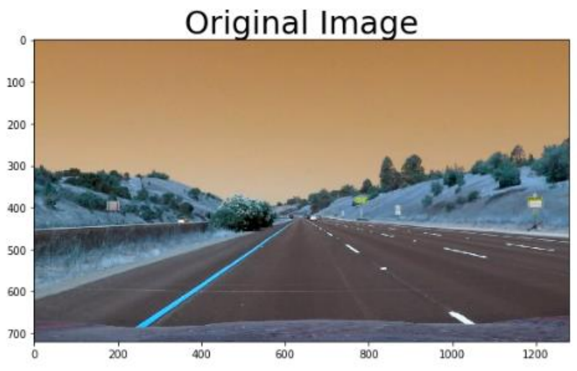


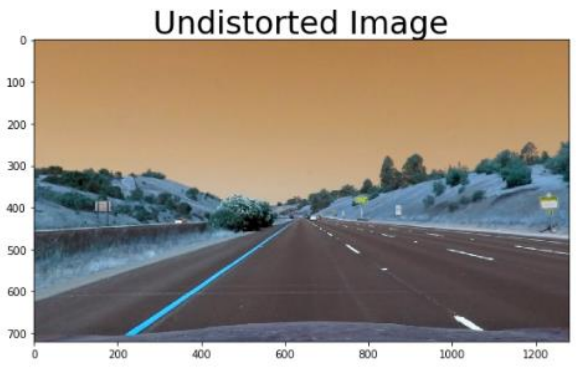
Final Image



2. Apply a distortion correction to raw images.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one. We use the camera matrix, distortion coefficients and use *cv2.undistort()* to apply on the raw images and result is below.

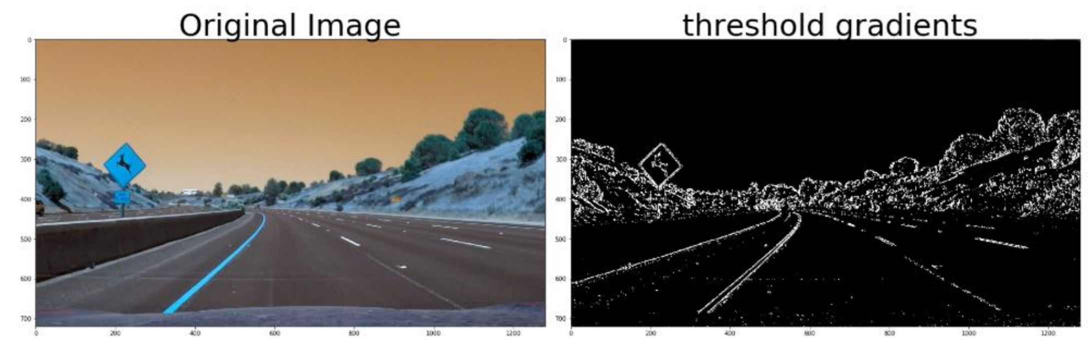




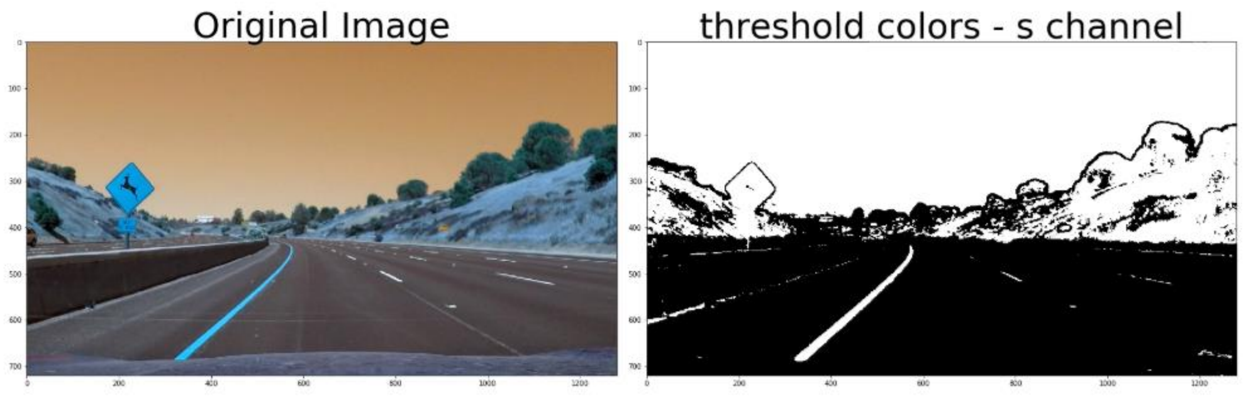
3. Use color transforms, gradients, etc., to create a threshold binary image.

From **code cell 6** there are 2 functions that performs the following actions to create a binary image

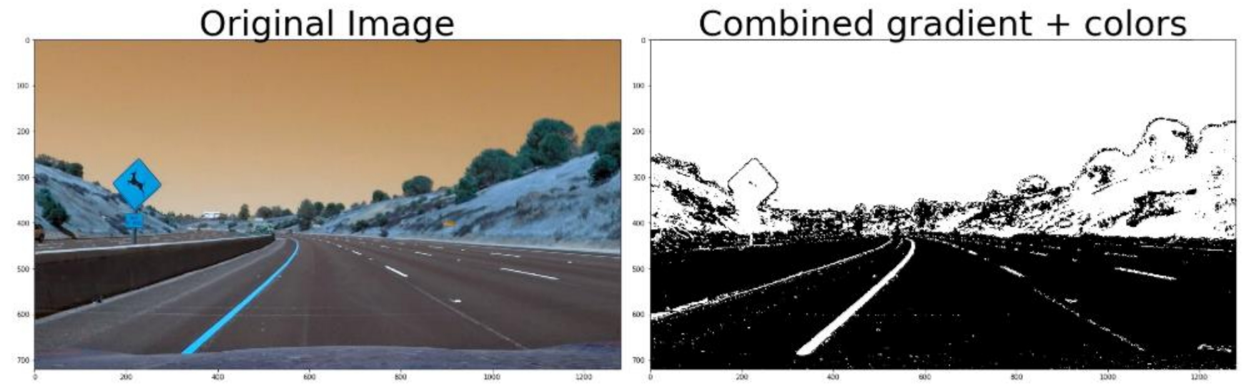
1. **sobel_operation**
   Converting the original image to gray, I applied open CV's sobel operator in the X direction to obtain the following result.



2. **hls_operation**
   Using open CV "COLOR_RGB2HLS" converted the source image into HLS color space.



Now the result of these 2 functions is combined to get the following binary image.

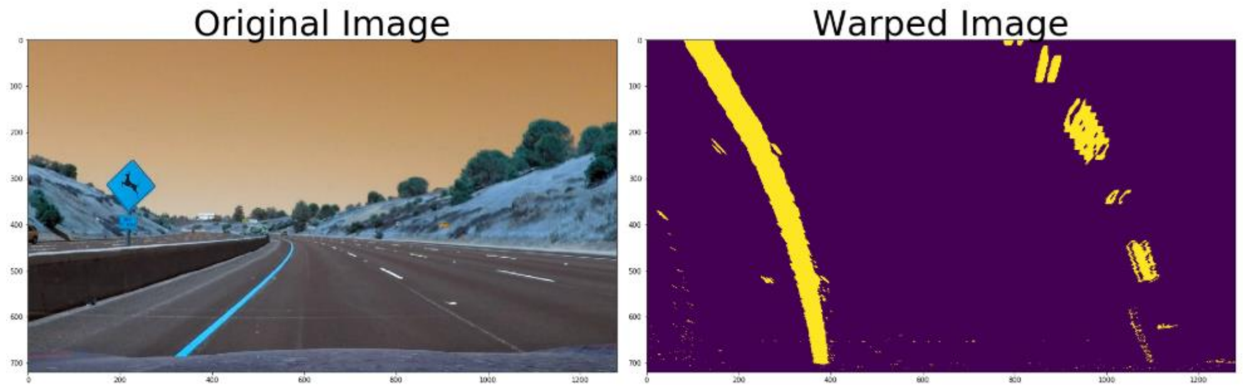


4. Apply a perspective transform to rectify binary image ("birds-eye view").

From **code cell 6** the function *getBinaryWarpedImage* takes the binary image along with SOURCE and DESTINATION points as stated below, along with MATRIX obtained by calibration camera in step 1 to produce a binary warped image.

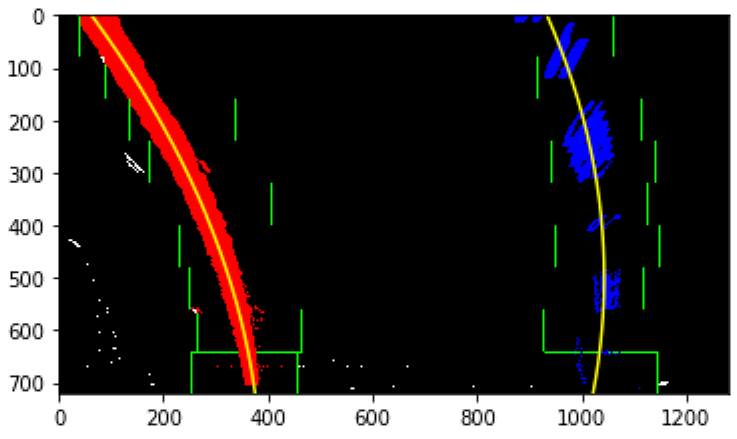
```
src = np.float32([[210, 710],[600,453],[680, 453],[880,710]])
dst = np.float32([[300, 710],[300, 0],[880, 0],[880,710]])
```

This is hardcoded for all images and obtained after a result of trial and error on all the test images. 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.



5. Detect lane pixels and fit to find the lane boundary, identify ROC calculation and the position of the vehicle with respect to center in the lane?

In **code cell 18 fitPolynomial** function accept the binary warped image and fits the polynomial line.



It's divided into 3 major sections separated by "#########################"

The top section calls 2 functions

> **Find Lanes –** Sliding window search to find the lanes in a binary warped images and get the coefficient of polynomial, the positions of the both the lanes. This is used in the first image only and in the images that has "Shades of tree". Reasons explained the last section of the document.
>
> **getLaneInfo –** The "look ahead" filter that uses the co-efficient from previous step and fits the polynomial returning the new coefficients and the positions of the both the lanes.

The middle section uses the coefficients from the previous section and calculates the Radius of curvature in m and position of the vehicle with respect to center in the lane. You can find them in the section marked by the below comments.

################### ############ ROC ##

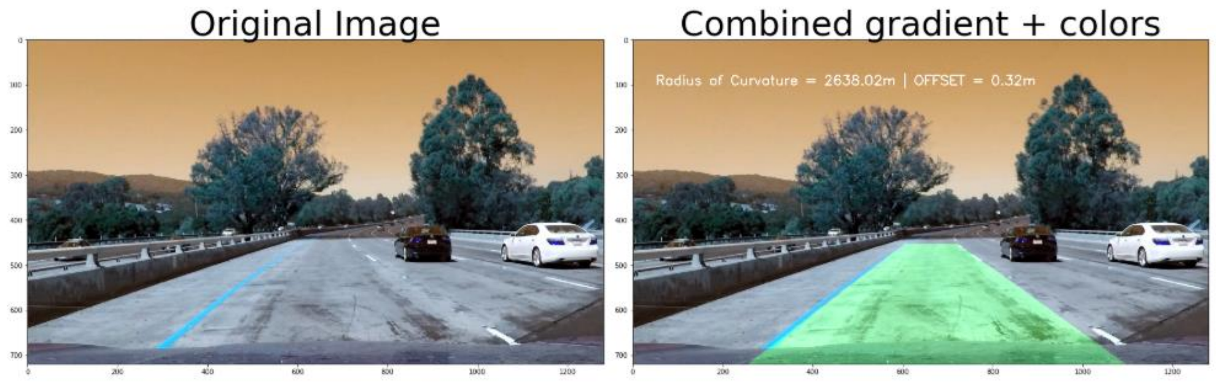
##### ############# position of the vehicle with respect to center in the lane###################

Next, the values are written on top of image using openCV's put text.

The last section draws the area between the 2 lanes whose positions are identified from the top section

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

This is an example image which shows lane identified.

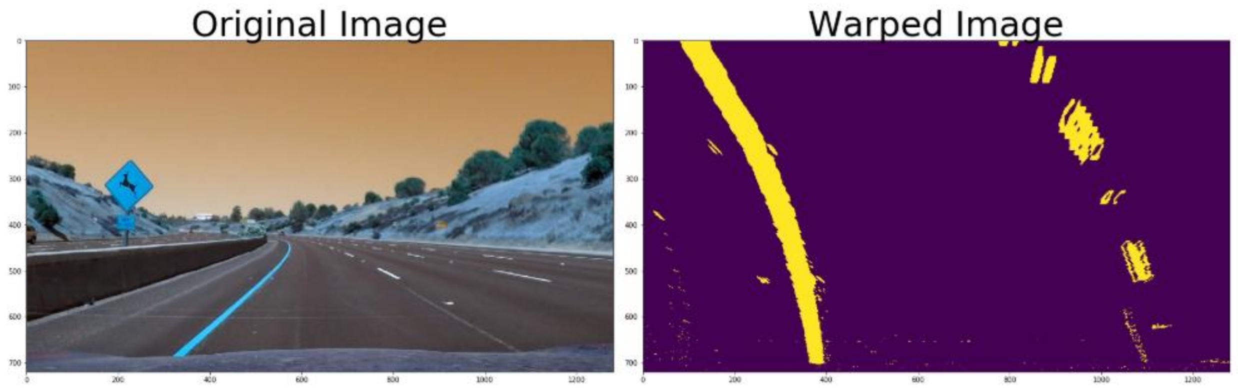

Original Image
Combined gradient + colors
Radius of Curvature = 2638.02m | OFFSET = 0.32m

7. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust

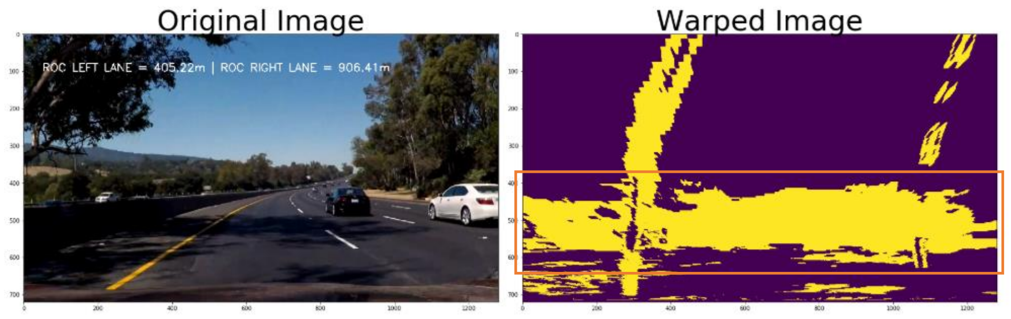
The following challenges were faced during the development of the project.

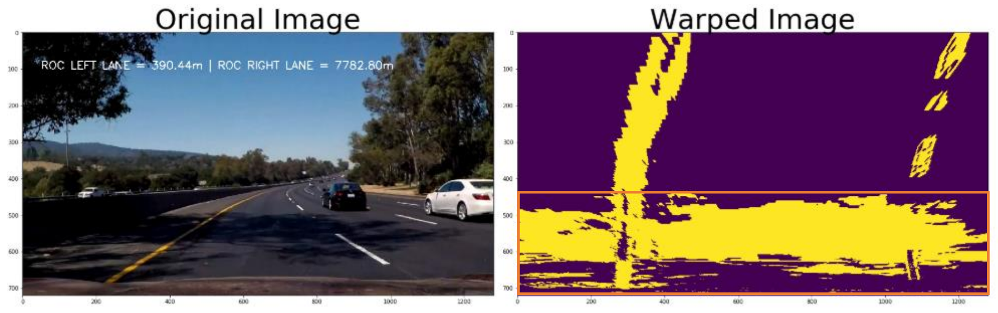
Find the SOURCE and DESTINATION points to *perspective transform* the binary image. This was trial and error process to get the points right.

Additionally there is a section in the video where "tree shades" creates a lot of noise and causes the identification of the lanes difficult. For e.g. for most of the video the image to fit a polynomial looks like



But precisely from **FRAME 1030 – FRAME 1057** there tree shades causes trouble





1. To counter this, I added a function <u>**isShadesPresent** that detect a shade in an image</u>. This is easily know by calculation the density of white pixels in an image.

2. Next after identification, again through trial and error found a <u>different SOURCE and DESTINATION points to fit the lane.</u>

3. I found that we have to <u>RESET the **fitpolynomial** function</u> i.e. start the sliding window search to fit the polynomial for each of the images with shades and not rely on the "look ahead" filter.

Possible Improvements

1. Make the lines smooth by the suggested taking the averaging the position of the lines.
2. Calculate the SOURCE and DESTINATION dynamically somehow for all possible frames that work.