Advanced Lane Finding Project

The goals / steps of this project are the following:

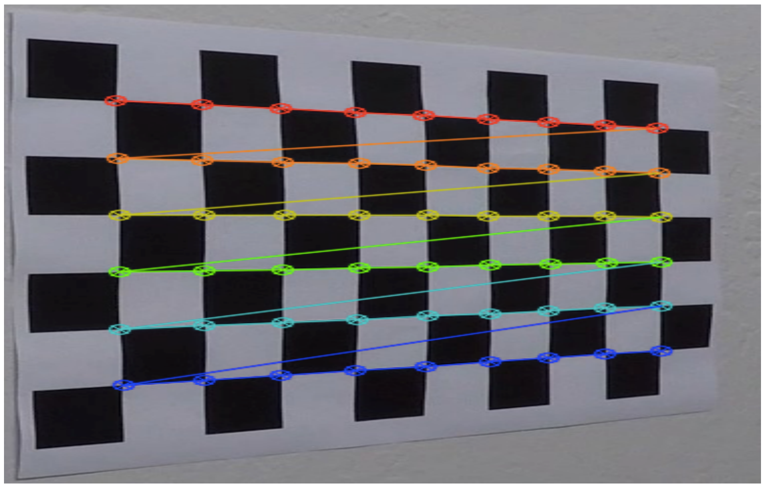
1. Compute the camera calibration matrix and distortion coefficients using the given set of chessboard images.
2. Apply a distortion correction to raw images of the road.
3. Create a thresholded binary image using color transform and gradient.
4. Apply a perspective transform to warp the binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Create a pipeline to undistort, find threshold, apply perspective transform, find & fit lanes, determine the radius of curvature, warp the detected lanes back to original and display the lane boundaries.
9. Test the pipeline with a test image and then on a video.

Rubric Points:

Camera Calibration (Lines [1-3] in the notebook):

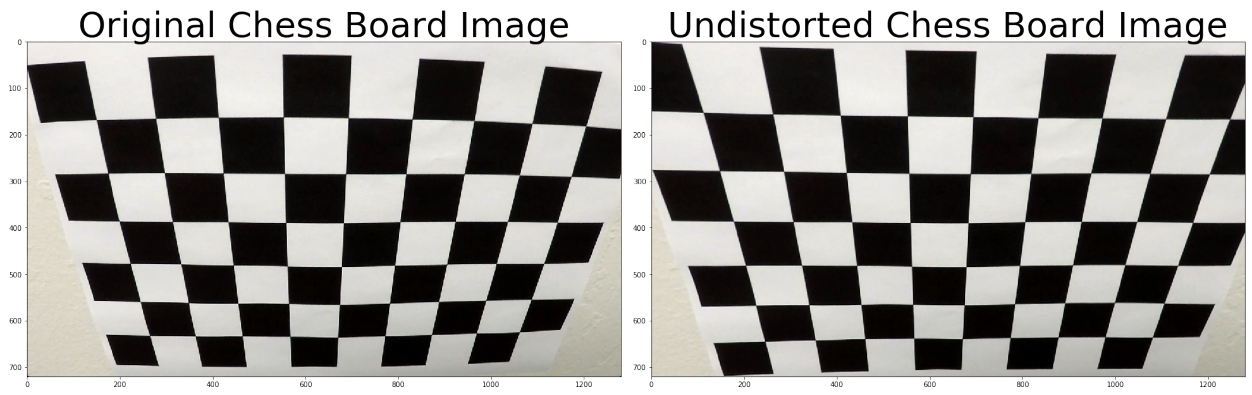
I have implemented the entire code in Jupyter Notebook. At first, I setup two empty arrays to hold the object points and image points. And then I prepared object points by creating 6*9 points in an array (corresponding to 6*9 corners in the chess board) each with 3 columns (x, y, z co-ordinates) and initialized the array to all zeros. The z co-ordinate is always zero since points are on a flat image plane. Using 'mgrid' function I generated the co-ordinates for 6*9 grid size.

Next, to create the image points I used OpenCV function 'findChessboardCorners' to detect the corners on a distorted camera image of a chessboard. The function 'findchessboardcorners()' returns the corners found in a grey scaled image. I'll append the detected corners to the image points array. Also, using the function 'drawChessboardCorners()' I'll draw the detected corners on the image. The output looks something like the one below showing the image points–



I followed the above procedure on all other calibration images and appended the image and object points to the arrays.

After iterating through all calibration images, I used the OpenCV function 'calibrateCamera()' which takes in object points, image points, image shape. The function mainly returns the distortion co-efficients, camera matrix to transform 3D object point to 2D image points.
Next, I used 'undistort()' function which takes image, camera matrix and distortion co-efficents to undistort the image.



Applying Distortion correction on a raw image (Line [4] in the notebook):

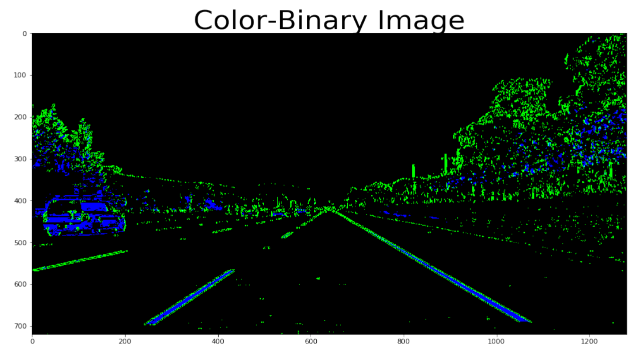
By feeding the raw image along with the object points and image points (from the camera calibration) to the 'undistort()' function the below undistorted image was obtained.
Note: The difference in the undistorted image can be noticed near the rear end of the white car.

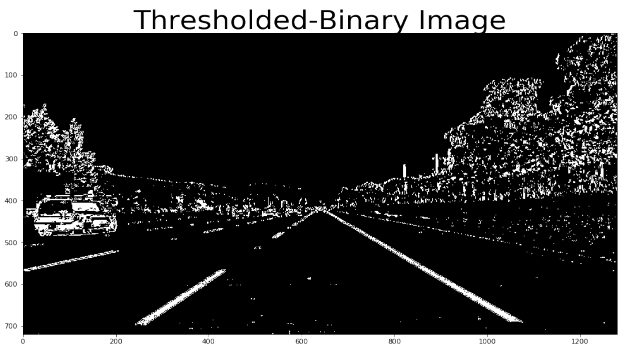
Create a thresholded binary image using color transform and gradient (Line [5-6]):

The following steps were used to create a binary threshold image –

1. The undistorted RGB image was transformed to HLS color space using 'cvtColor' function
2. The s-channel was extracted from HLS color space
3. The undistorted RGB image was converted to gray scale
4. Sobel operator was used to take the gradient w.r.t x; Absolute value of sobelx was calculated and scaled to 8-bit image
5. With threshold range (20,100) a binary array equal to the size of a scaled sobelx was created by setting all the elements in the array that are within the range to 1
6. Similarly using s-channel and a threshold range (170,255) another binary image array was created
7. A combined binary image is formed by overlapping the above two binary image arrays

The combined thresholded binary image is shown below -

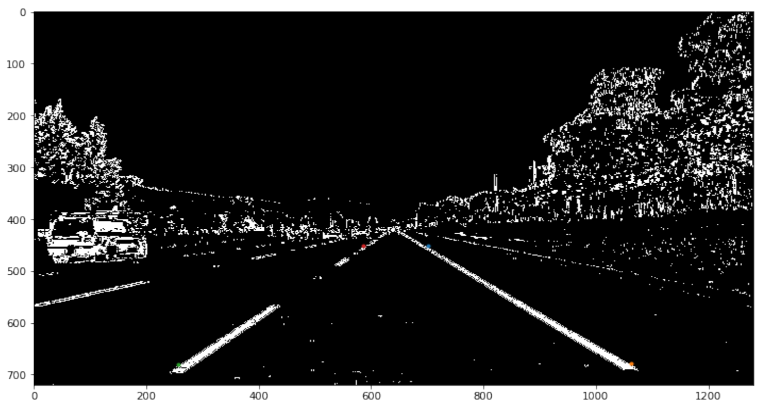




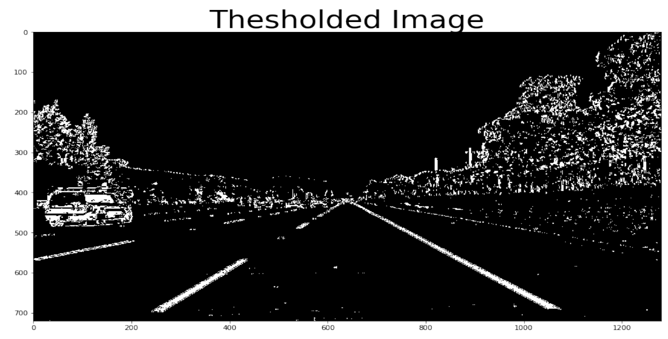
Apply a perspective transform to rectify binary image ("birds-eye view")(Line [9]):

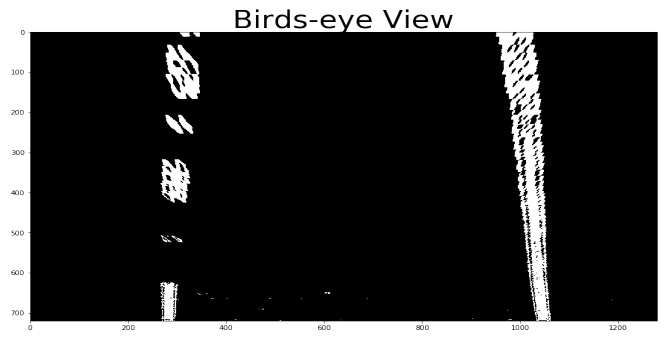
The first step in working towards warping an image was to select four points around the lane lines to apply perspective transform. I chose the following source and destination points -

```
source = np.float32(            destination = np.float32(
      [[1065,0],                                  [[701,451],
       [1062,678],                                 [1065,714],
       [265,714],                                  [256,681],
       [586,451]])                                 [265,0]])
```



The perspective transform (M) was calculated using ‘getPerspectiveTransform()’ function which takes in the source and destination points. Finally, the thresholded binary image was warped using ‘warpPerspective()’ which takes the image and perspective transform as input.
‘Birds-Eye View’ is shown below.

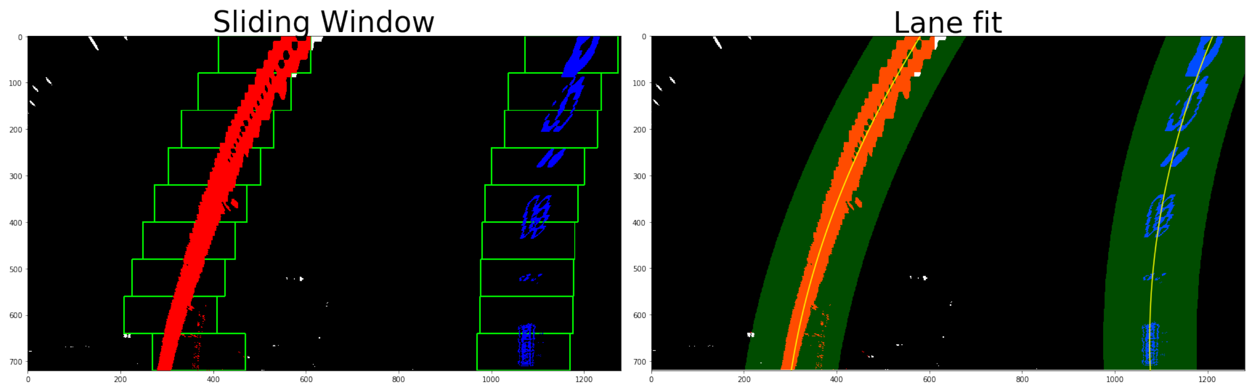




Detect lane pixels and fit to find the lane boundary(Line [11-12]):

Implemented ‘Sliding windows’ to search for lane line pixels on a warped binary image.
The histogram taken along the columns, the peaks were identified and peak positions were used as a starting point.
First, the number of windows of sliding windows was chosen and the size of each window was calculated.
The first window was placed at the peak positions and the co-ordinates of non-zero pixels were identified.
As the windows slid from bottom to top their positions were adjusted to the mean value of the non-zero pixel region. Fitted a second order polynomial to left and right line pixel positions.

Sliding window and Lane fit output shown below -



Determine the curvature of the lane and vehicle position with respect to center. Line [16]:

Used the polynomial co-efficients A & B (calculated from lane fitting) in the below equation to determine the radius of curvature of left and right lane.

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

The measurements were converted from pixel space to meters using the below scaling–
ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/700 # meters per pixel in x dimension

Offset:

The offset between the center of the image and the lane center was calculated as follows –

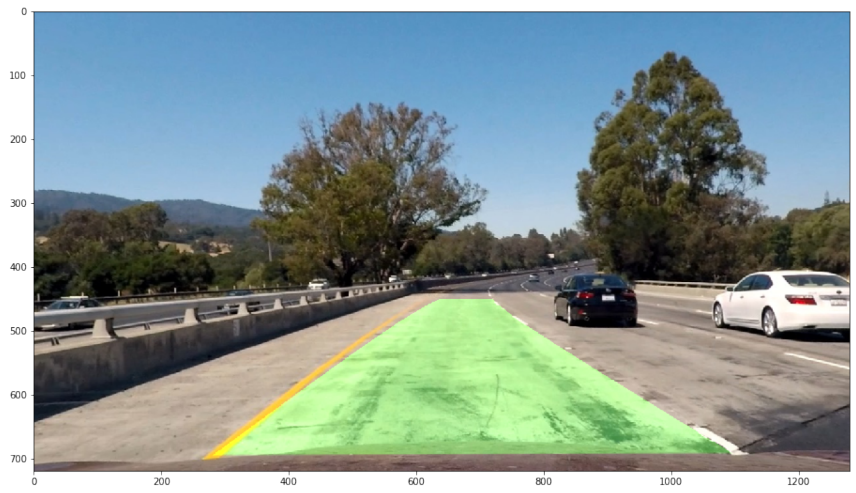
The center of the image (Image_Center) was calculated by using “image.shape[1]//2”
The lane canter (Lane_Center) was calculated by determining the distance of left lane pixels and right lane pixels from the left corner of the image.

Offset was calculated using the below formula:

Offset = Image_Center – Lane_Center

Warp the detected lane boundaries back onto the original image (Line [13]):

Using inverse perspective transform(Minv) the detected lane boundaries were warped back to the original image using 'warpPerspective()' function. Warped image shown below-

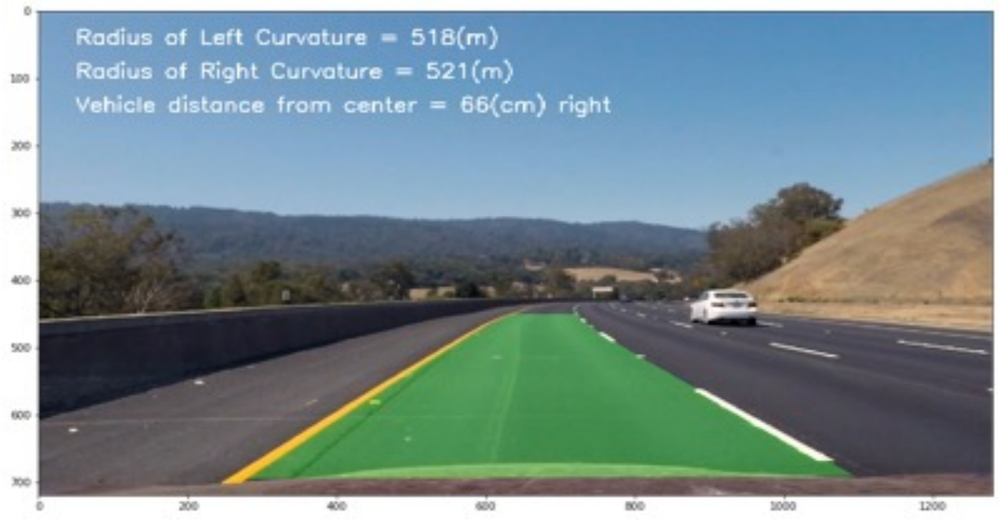


Create a pipeline to undistort, find threshold, apply perspective transform, find & fit lanes, determine the radius of curvature, warp the detected lanes back to original and display the lane boundaries. (Line [18] in the notebook):

A pipeline called "process_image" was defined to do the following actions:

1. Undistort the image
2. Create a thresholded binary image
3. Apply perspective transform to get the 'bird-eye view'
4. Find lane pixels and fit a second order polynomial
5. Determine radius of curvature of the lane lines and,
6. Finally warp the detected lines back to the original image

Pipeline output showing the lane area, radius of curvature and offset value:


Radius of Left Curvature = 518(m)
Radius of Right Curvature = 521(m)
Vehicle distance from center = 66(cm) right

Discussion

One of the shortcomings in this implementation is that the lane lines are not perfectly determined when the vehicle approaches an area with high brightness. The thresholded binary image needs to be improved in finding more lane pixels.

The other aspect where there is need for improvement is in determining the radius of curvature.

In the project video output “project_video_output.mp4”, the radius of curvature value is way too high between 0:12 and 0:23s. This could be improved by analyzing the polynomial fit function on the lane pixels and the characteristics of the lanes on the road (like are the lanes too straight??!).