Advanced Lane Finding

Overall Goals:

The goals / steps of this project were as following (taken from UDACITY template write-up):

- 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").
- 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:

I loaded the images from the 'camera_cal' folder having distorted images of a chessboard pattern. Most of the images (17) had 9x6 corners visible while rest 3 had some corners hidden. I used only those image where all the corners were visible to make the process simpler. Then, I used the openCV functions 'findChessboardCorners' & 'calibrateCamera' to calibrate the camera and generate the camera matrix 'mtx' as well as the distortion coefficients 'dist'. Calling the user defined function 'Camera Calibration' with the chess board images returns the values for 'mtx' and 'dist'. Since this calibration was required to be done only once, therefore these values were stored in a pickle file. Example of the undistorted images is given below.

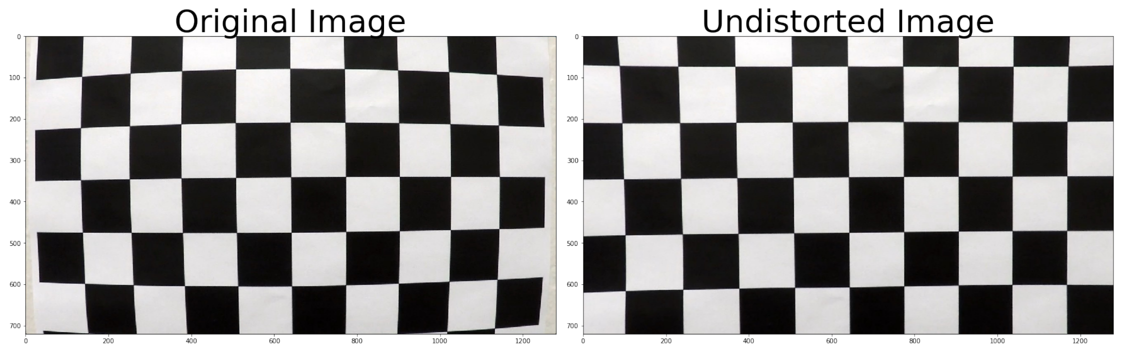

fig1: Figure showing original and undistorted image.

Distortion Correction:

After calibrating the camera, I loaded the test images to further tune my algorithm. I used the 'cv2.undistort' function to to do the job. Calling user defined 'undistort_image' function return the undistorted image. Below is an example showing both the original and the undistorted image:

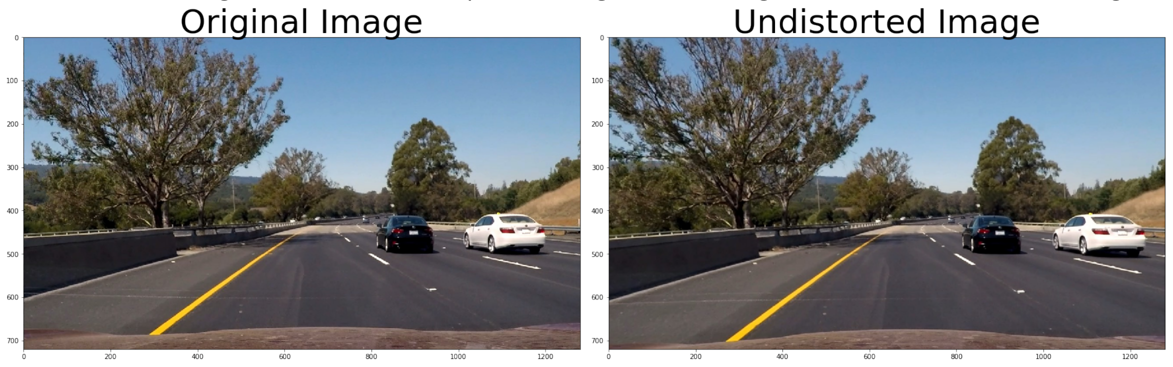

fig2: Figure showing original and undistorted image.

Thresholding:

After getting the undistorted image, I needed to generate a binary image to seperate out the lane lines from the rest of the image. For this I combined two processes. Selection of the two processes was itself a result of an extensive trial and error process.
For first part of the process, I used the sobel function on the grayscale version of the image and applied threshold limits in the x-direction. After several trial and error, I fixed those limits at 30 and 150. Also, a kernel size of 11 worked best for me.
Other than this, applying sobel function on the Red scale as well as the L scale (from HSL color scale) was also tried, but the gray scale seemed to give the best results.
For the second part, I converted the image into a HSL color scale, since different lighting conditions don't have any effect on the Saturation of an image. Again, I applied thresholds to generate another binary image. Setting the thresholds at 175 – 250 gave the best results.
Finally, I combined the results of the two processes and generated a binary image.
(Credit to Paul Heraty for the threshold limits)

An example of the effect of threshholding is shown below:

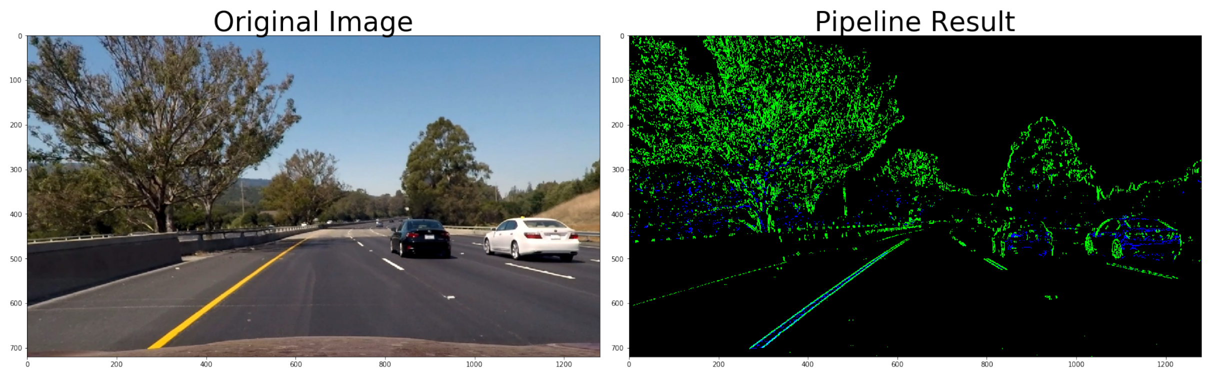

fig3: Figure showing undistorted and threshholded image.

Perspective Transform:

After getting a binary image I performed a perspective transform, to get a warped image as well as the perspective transform and inverse perspective transform matrix. Performing perspective transform is important since it gives the bird eye view of the lanes. Using that we can easily calculate the curvature of the lanes as well as calculate the position of the car with respect to the center of the lanes. Choosing source and destination points for this image was the trickiest part for the perspective transform. I started off with the sample points given in the UDACITY write-up and tweaked the values till it gave me the best results with my overall algorithm. The final values I chose were:

Source Points	Destination Points
(575, 460)	(320, 320)
(190, 720)	(320, 720)
(1127, 720)	(960, 720)
(705, 460)	(960, 0)

Table 1

An example of perspective transform is shown below:

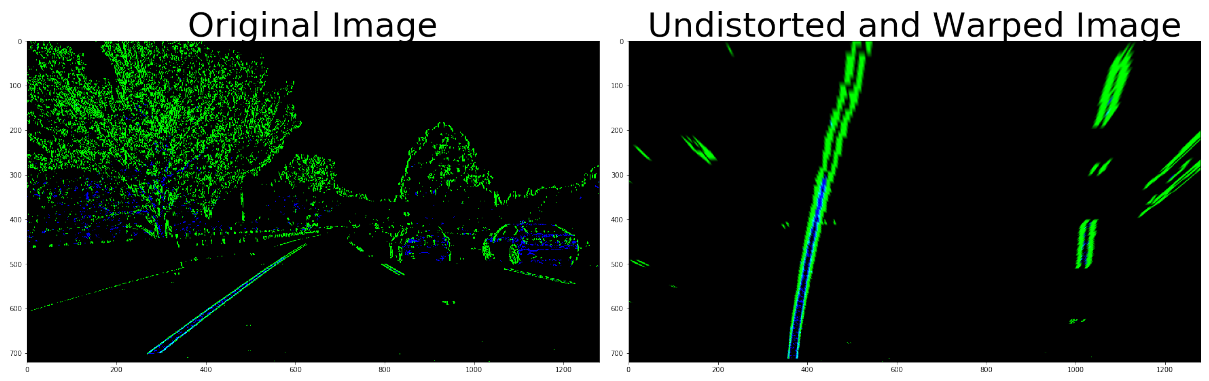

fig4: Figure showing threshholded image and warped image

Lane Fitting:

For finding the best fit lanes from the warped image generated by the perspective transform function, I used the Sliding window algorithm provided by the UDACITY team. The algorithm starts by summing the pixel values from the half length of the image all the way towards the bottom. The lane lines should show a spike at two image locations as is indicated by the picture below.

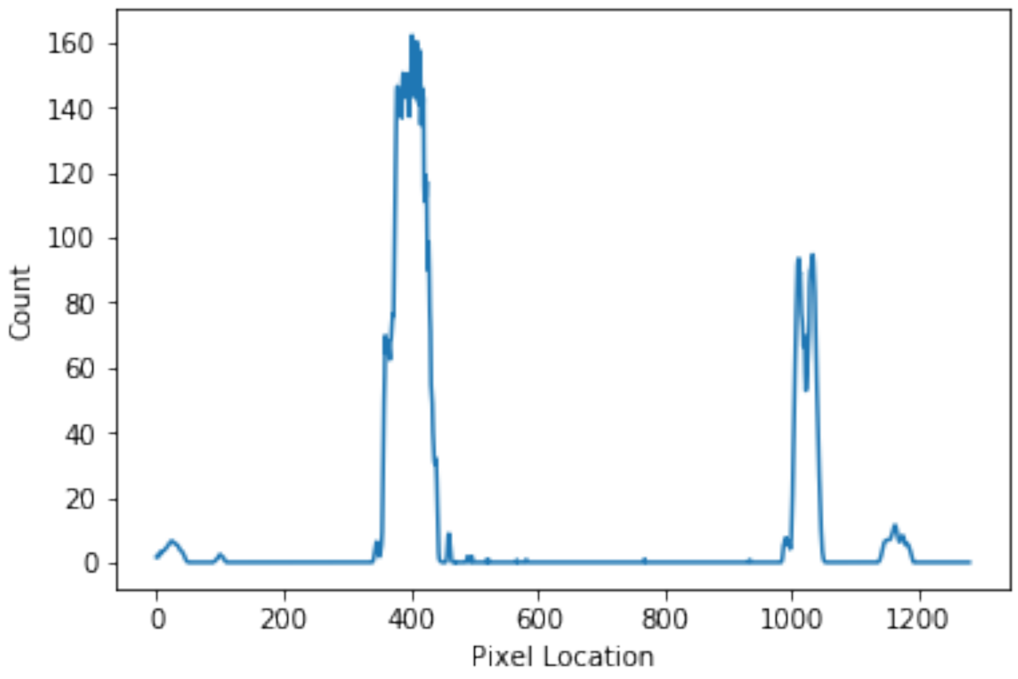

fig 5: Figure showing histogram output

Now, it takes up the two spikes from either side of the half line of the image and form a rectangle around it. Now it moves upwards in the steps trying to locate more pixel spikes in the vicinity of the previous rectangle. When it finds all such points it tries to fit them to a second degree polynomial of the form: $Ax^2 + Bx + C$. These second degree polynomials are the two lane lines.

This process is slightly computationally extensive. I tweaked this algorithm slightly so that it does not blindly search for points every time. Each time the algorithm has found a good lane line, it searches for the next lane line within a defined vicinity of the previous line.
This made the process of finding lanes much faster.
An example of lane lines found by the algorithm is shown below:

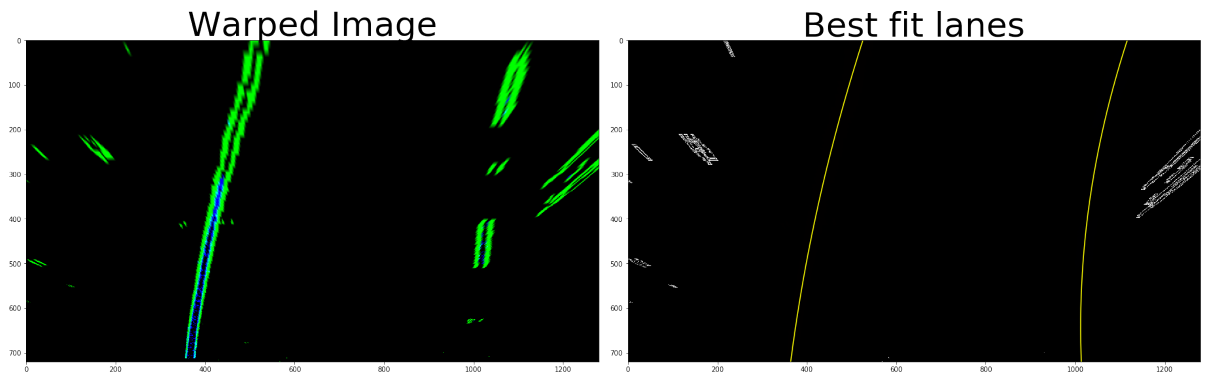

fig 6: Figure showing best fir lanes

Radius of Curve & Distance from the center:

Once the algorithm has located the lane lines, then it calculates the radius of curvature (R) using the formula:

$$R = \frac{(1 + (2Ax + B)^2)^{3/2}}{|2A|} \qquad \text{eq (1)}$$

where A & B are the polynomial coefficients we fit earlier.

For calculating the distance from the center, it was assumed that the width of the lane is 3.7 m. Center pixel location of the image was then calculated. The the midpoint of the pixel location of the two lanes was calculated, and it was subtracted from the screen middle pixel value to get the desired distance in pixel value. It was then converted to cms.

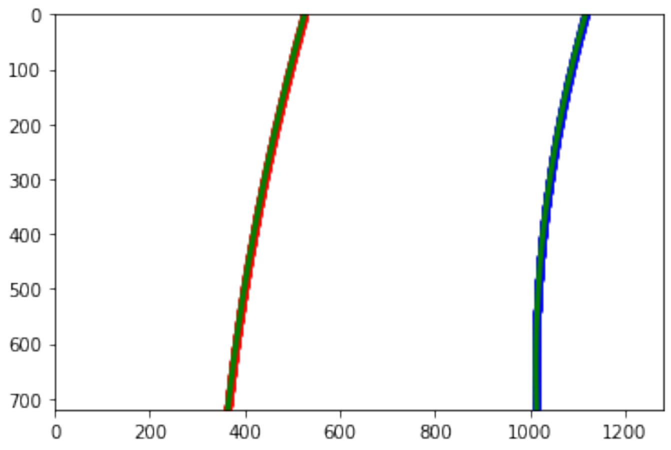

fig 7: Figure showing output after finding radius of curvature

Final Image:

Finally, a shaded area was added to the image identifying the lane. Also, the radius of curvature for the two lanes along with the distance from the center value were added to the final image which looked something like this:

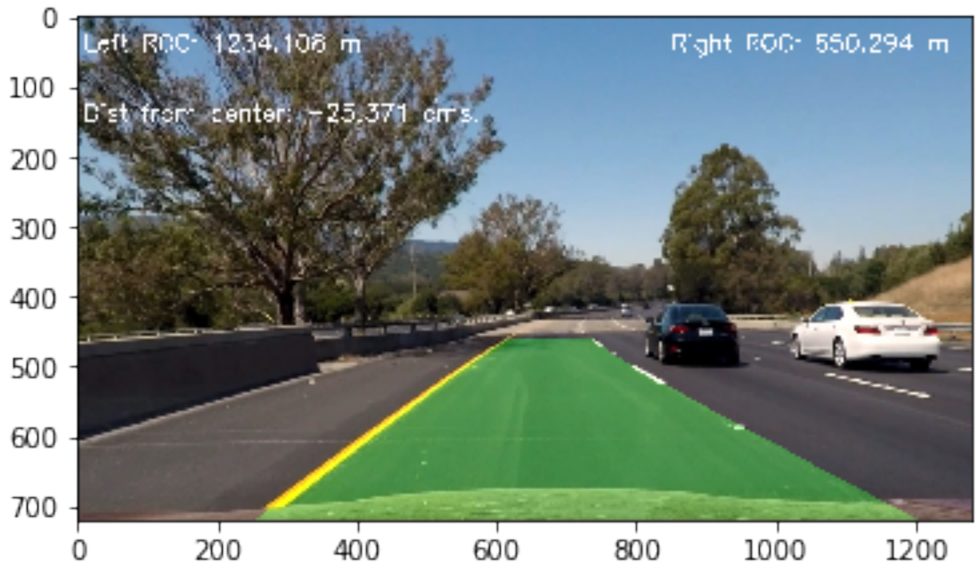

fig87: Figure showing final output

Checking for Spurious Lane Values:

Since the entire algorithm was applied to a video, it was necessary to check for spurious lane values calculated by the algorithm. For, this I stored the last lane polynomial coefficients for both the lanes and compared them against the current coefficients calculated by the algorithm. If the sum of the differences was less than 5, then the current values were accepted after applying a smoothening function which add 90% of current value and 10% of previous value. It prevents any jerky behavior on part of the algorithm. If the difference is greater than 5, then the last best lane is used for current frame.
Finally, everything was put together in a function called process_image and applied to the 'project_video' to generate 'project_output' video. The output video is part of the attached zip file. It can also be seen in the attached Jupyter notebook named 'Project_4.ipynb' in the very end.

Conclusion:

The final video generated shows that the algorithm finds the lanes pretty well, even in varying lightning conditions as well as different turns. However, there are potential loopholes and possibilities of failures in this algorithm:

- The source point for perspective transform are fixed. There needs to be an algorithm which can detect them dynamically.
- The algorithm will fail if there are any sharp turns. To mitigate this, instead of taking sobel threshold only in the x direction. Magnitude of x and y thresholds should be considered.
- The algorithm will fail if there are vertical cracks on the roads or vertical variations of road color (vertical with respect to the image generated by camera). The algo needs to take intoaccount the expected road width to overcome such an issue.