CarND Term1 - Project4: Advanced Lane Finding Project

In this project, the goal is to write a software pipeline to identify the lane boundaries in a video from a front-facing camera on a car. This can be achieved by the following steps:

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.

2. Apply a distortion correction to raw images.

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

4. Apply a perspective transform to rectify 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.

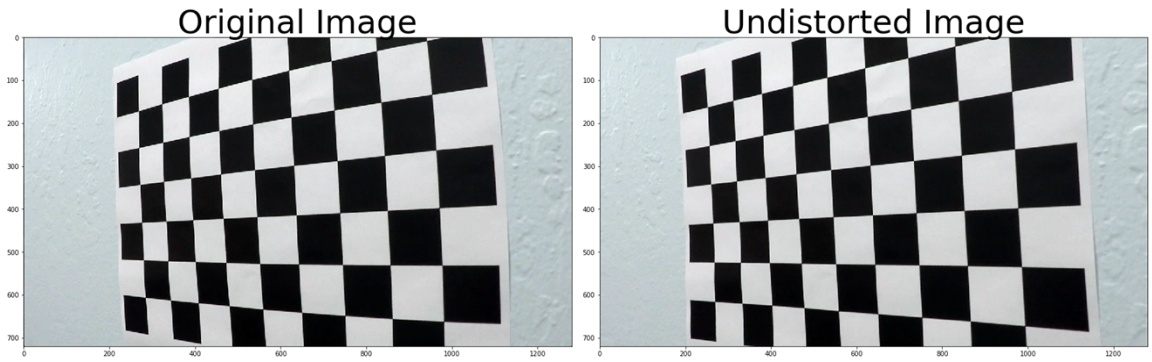
7. Warp the detected lane boundaries back onto the original image.

8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Camera Calibration and applying Distortion correction

I started by preparing "object points", which are the (x, y, z) coordinates of the chessboard corners in the real world. Here I was 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 were appended with a copy of it every time I successfully detected all chessboard corners in a test image. Imgpoints were appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

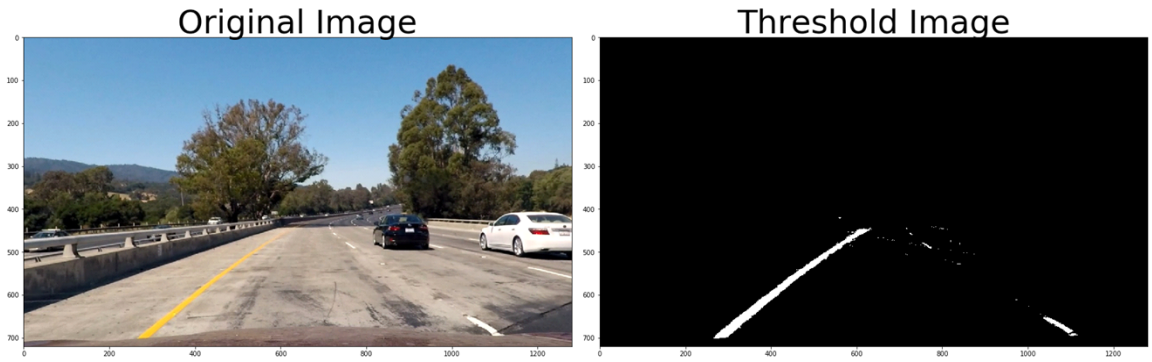
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:



Gradients and Color transforms to create a thresholded binary image

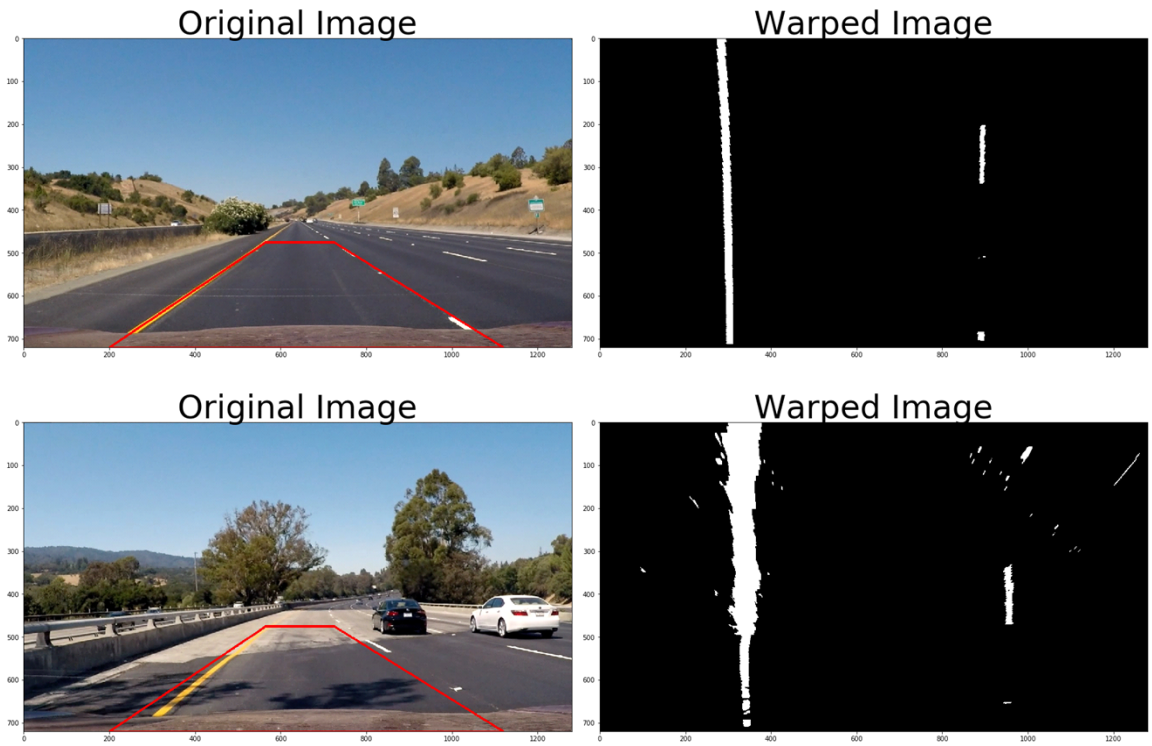
Just the same way Canny edge detection was used during the first project to detect edges, here I used a customized combination of gradients and color transforms to detect lines in the area of interest. I used the same code from my Project1 to define the 'region of interest'.

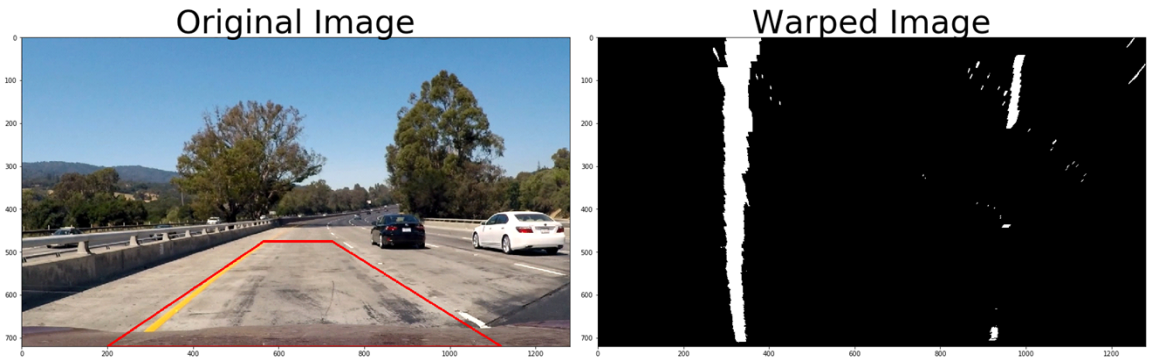
I first started with a combination of the following thresholds as discussed during the Udacity lessons: Sobel threshold, Magnitude threshold, Direction threshold, S-channel-threshold. I was able to detect lines but many unnecessary lines were also coming up, so as suggested in some Udacity discussion forums, I used an L channel threshold to avoid dark pixels and R-G threshold to detect yellow lines in a better way. After playing with the threshold limits for a while, these are the results I got:



Applying Perspective Transform

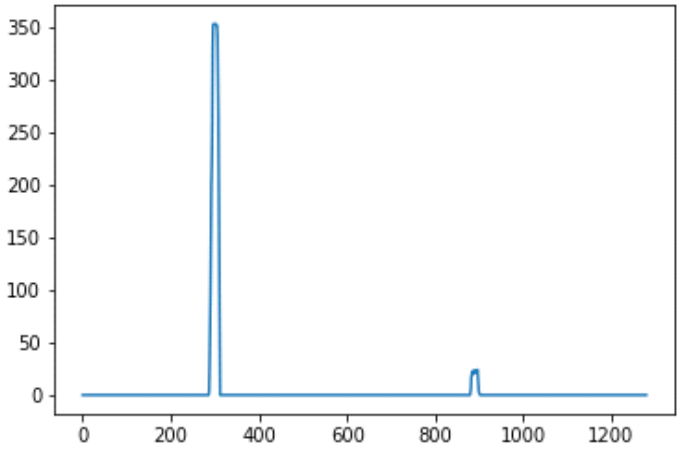
In order to find out the curvature of the lane lines to better detect and draw them on the images, a perspective transform needed to be applied on the above threshold image and the result can be seen below:



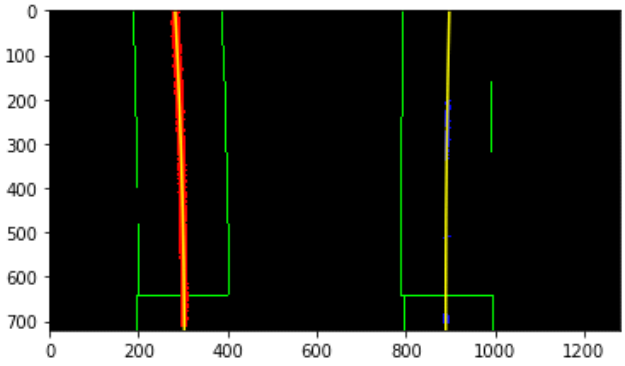


Detecting Lane Pixels and fitting them to a Polynomial

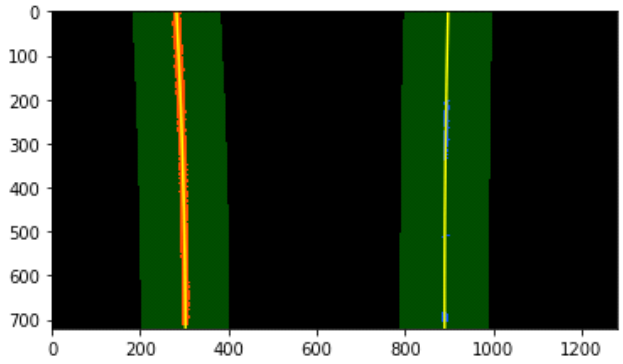
Used a histogram and sliding window search as described in the Udacity lessons to identify the non-zero pixels in the Warped binary image and plot them. The histogram data helped in identifying the position of the two major lines in the perspective transofrm. Used that data to fit a polynomial curve for the left and right lanes. Here are the results of the test image 'straight_lines1':



Histogram



Visual representation of Sliding Window search



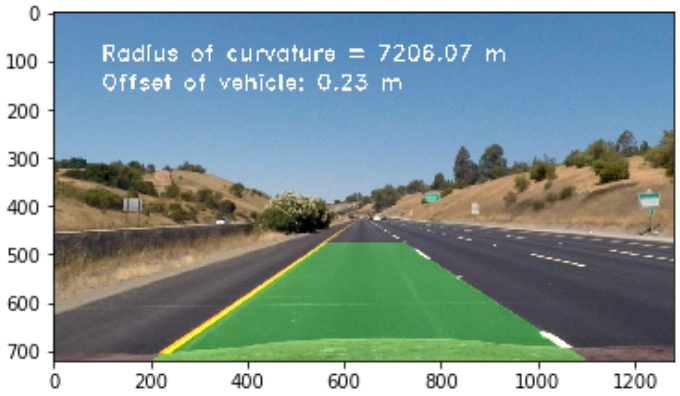
Left and Right Lane lines in the Perspective transform view after identifying their positions through Sliding window search

Radius of Curvature and Vehicle Offset

Defined functions to determine radius of curvature of the left and right lanes and Vehicle offset from the center of the two lanes. The difference in the radius of curvature of the two lane lines was later used to weed out bad results, i.e., results where the left and right lanes are not parallel(approximately) to each other. The rational being, the two lane lines are always almost parallel to each other on roads whether they are straight or curved.

Output Final Processed Image

Applied all the functions above on a test image and the result is shown here with average radius of curvature of the two lane lines and offset from the center displayed on the image:



Pipeline for producing lane lines area in the video

I had initially started with just the above mentioned functions and the video looked good until it started seeing wight areas on the road and shadows. Neither the 'predict lines ' function nor the 'sliding window search' function were able to detect lines on some portions of the road. Also, the lines produced on some frames were out of bounds.

So, I then implemented a function to store data of previous good images and take the average of them if either the left or right lines are not properly detected through the lane detection functions. I used the difference(500m) in radius of curvature between left and right lanes in meters to detect whether the lanes can be considered good or bad. I ignored any lane lines with a difference of more than 500m.

One more parameter I had to play with to reduce the wobbling of the lines in the video is the 'number of frames' used to take average in case of bad results. A high number (started with 20 frames) generalized too much and couldn't properly adjust to the changing curvature of the road. I finally brought it down to 6 'previous good frames' to have a relatively smooth transition in the video from straight roads to curved roads. The 'project video' is attached along with this write-up.

Further discussion

The pipeline didn't perform that well on the challenge videos and I see that the following changes could help make it perform better:

i. Adjusting the Magnitude and Color thresholds to better detect the lane lines.

ii. Performing at least one more sanity check (distance between the lines?) and combining it with difference in radius of curvature to better define good vs bad results.

iii. Adjusting the number of good frames to average over to find a good trade-off between maintaining continuity and not overly generalizing the lane lines.