Advanced Lane Finding

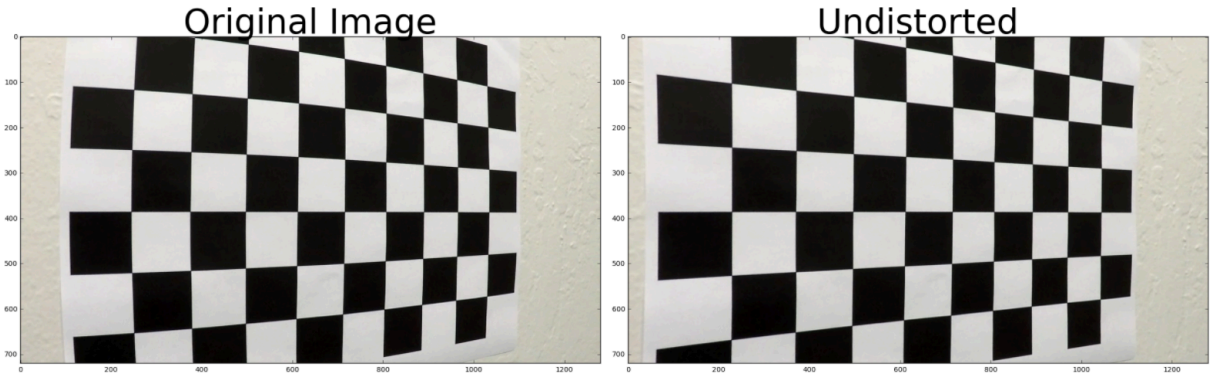
All the questions are answered in reference to the project notebook: "Advanced Lane Finding Submission.ipynb"

Camera Calibration:

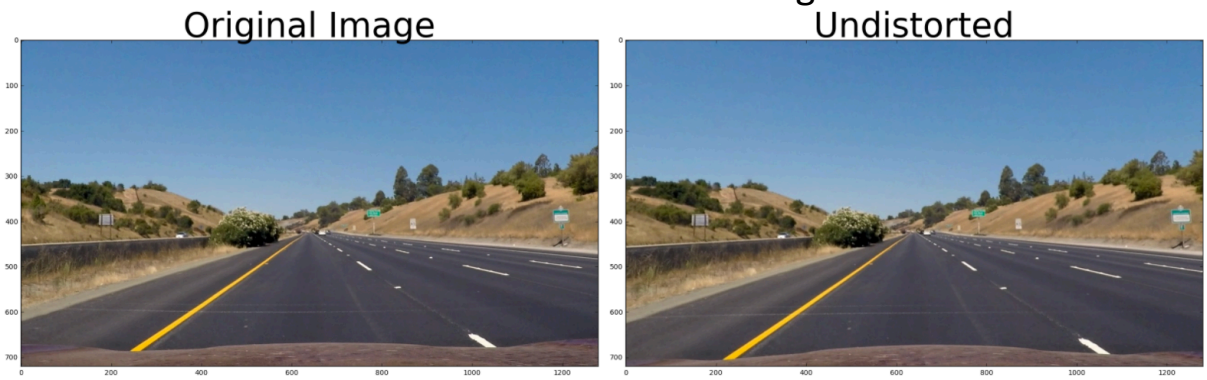
Camera calibration is performed in cell 2 of the notebook.

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.
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.

The output images are as follows:



Iused the calibration to un distort the test images and the results are:



One can clearly observe that there is a change in the image around the hood on each side as expected.

Pipeline (single images)

Pipeline function is written in cell 4 of the notebook.

I used a combination of color and gradient thresholds to generate a binary image. Here's an example of my output for this step.

Undistorted Image

Gradient Image

Iconverted the image to HLS from BGR. HLS transformation helps us to detect color (hue), amount of color (S) and lighting conditions (L). Itook the L channel, applied Sobelx filter on it to obtain the gradient in the x direction. Ithresholded the gradient to obtain relevant pixels. The threshold Iused for this part was (30,150).

Next Iused the S channel to capture color information which I missed in the gradient part. I restricted the S image to a range of values by thresholding. The threshold I used for this part was (170,255).

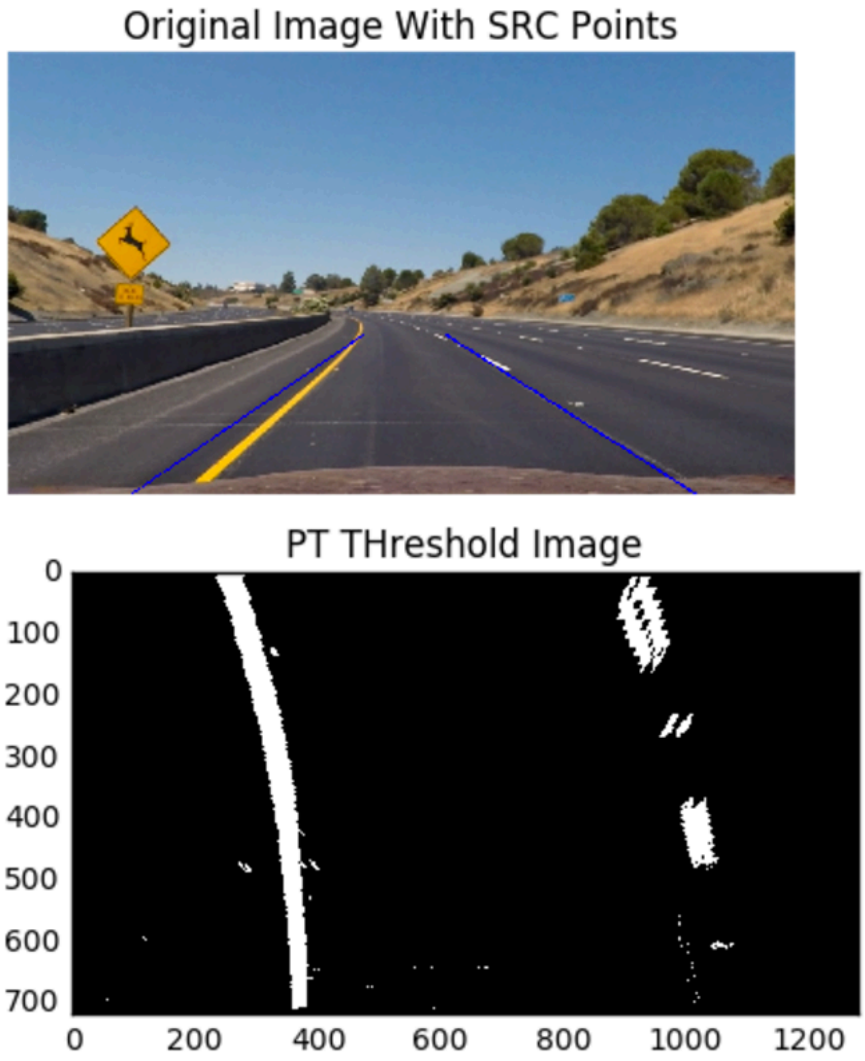
Finally, I combine both the images of gradient and S channel to obtain the final image with lane lines detected as shown above.

Perspective Transform

Code for my perspective transform appears in the *warper function* in the cell 11 of my notebook. Note that the function is a part of *line class*. The source and destination coordinates are mentioned below:

Source Coords	**Destination Coords**
(200,720)	(300,720)
(1120,720)	(1000,720)
(710,460)	(1000,0)
(580,460)	(300,0)

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.



Note that perspective transform image shown above is obtained after the pipeline function on the original image.

Lane Line Detection

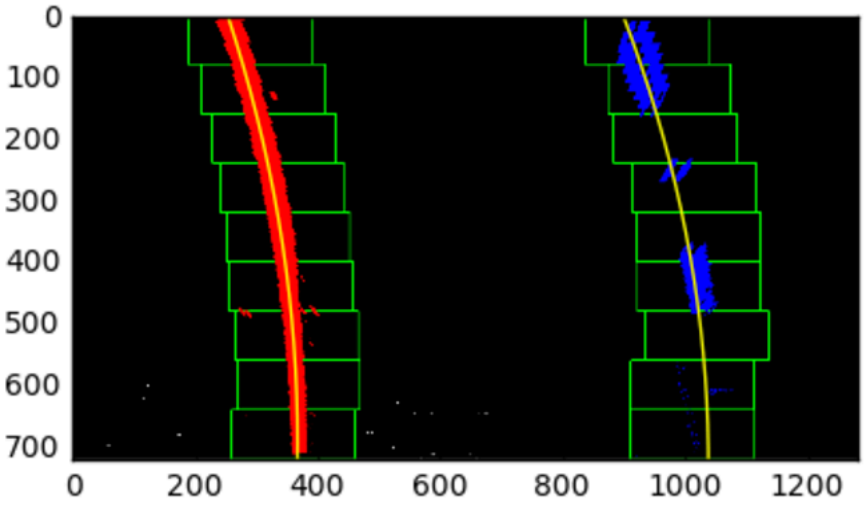
Lane lines are detected in the function detect_lane_lines as part of the line class in cell 11 of notebook. I used the polynomial fit method from the lectures to detect the lane lines.

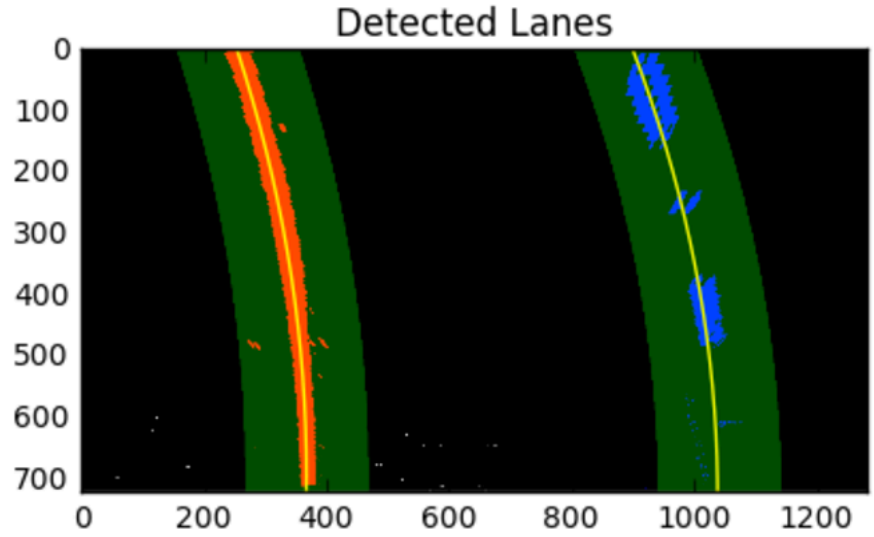
Here are the steps I take for detecting lane lines:

- Parameters:
    - Num windows = 9
    - Minpix = 50
    - Margin = 50
- If previous = False i.e. there is no previous frame detected, I do a brute force search for lane lines. Here are the steps:
    - Calculate the histogram of the bottom half of the image and obtain the peaks for left and right.
    - Use the peaks pixels on the left and right of the image as a starting point for lane search.
    - Obtain all the non-zero pixels within the margin (used 50) of the center pixels.
    - For next window, if you obtain more than *minpix* no of pixels for either left or right lane, use the mean of those pixels as a starting point for center of next window.
    - Use the non zero pixels obtained on the left and right to fit the left and right quadratic curves respectively.

- If previous = True i.e. if there is a previous detected frame, I do a targeted search based on the lane lines in previous frame. For search of center pixels in each window, I use the polynomial coefficients from previous frame to obtain the center pixel for new frame. Rest of the procedure remains the same as above.

Here are the images of detected lane lines.





Sanity Checks

To verify that the detected lane lines are indeed good, I need to perform sanity checks. For a lane to be valid, it has to pass the three sanity checks as mentioned below:

- **Distance Check:** Difference in centers of lane lines as per US guidelines is 3.7 m. This corresponds to 700 pixel in our images. I checked whether the distance between the lane lines is between (650,750) pixels. If yes, then I assume that detected lane lines passes the distance check.
- **Radius of Curvature Check:** Idea is that for detected lane lines, the Roc of left and right lanes should be approximately same. There are several ways to approach this check. However, the way that worked for me is as follows:
    - **Straight Lines**: If ROC of either left or right lane is greater than 10Km, I assumed it is a straight line frame and neglected the ROC check.
    - **Curved Lines:** If ROC of both left and right lane is below 10K, I assumed it is a curved lane. For curved lane if the difference between Roc left and Roc right is less than 1Km, I assume that Roc check passes.
- **Parallel Check:** The left and right lanes should be parallel to each other. For detecting that I just checked whether signs of each of the coefficients for left and right lanes are same or not. If the signs are same, I assumed that lines are parallel. There might be better ways of doing this, but for this case it worked pretty well.

Radius of Curvature Calculation

Function named **get_roc** in cell 11 of notebook contains the code for getting roc. The code there is pretty self explanatory.

Getting the Road Image

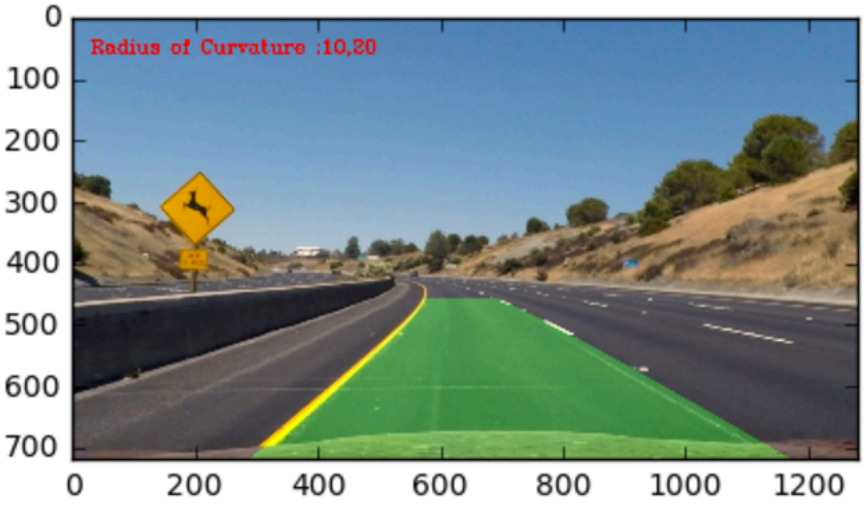
If detected lanes in a frame passes the sanity check, I perform the following steps:

- **Smoothing:** I use the average of the coefficients of this frame and previous 4 frames to get the final coefficients for the left and right lanes. This can be seen in main function in cell 11 of the notebook.
- **Inverse Perspective Transform and Plotting:** Finally, I do a inverse perspective transform and plot on road image using get_road_image function in cell 11.

If detected frames doesn't pass the sanity check, then I perform the following steps:

- I keep a counter of consecutive faulty frame in the video. By faulty, I mean the frames which didn't pass the sanity check. If the number of faulty frames are less than 3, I use the average of coefficients from previous 5 frames as this frame's coefficient.
- If the number of consecutive faulty frame is above 3, then I perform the following steps:
    - If the lane lines were detected using targeted search of previous frame, I use a brute force search for lanes in this frame.
    - If the newly obtained lanes still didn't pass the sanity check, I skip that frame and use the previous frame coefficients nonetheless.
    - Else, I follow the usual course with new coefficients.

Here is my final result on one of the test images.(Video Link)



Discussion

The pipeline is definitely not perfect. It fails on the challenge video. This project required a lot of parameter tuning which makes me wonder if CV methods can work in all the cases. Working on this project has made me realize the importance of Deep Learning methods which are very generalizable.