Advanced Lane Finding

Advanced Lane Finding Project

The goals / steps of this project are the following:

- 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").
- Detect lane pixels and fit to find the lane boundary.
- 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.

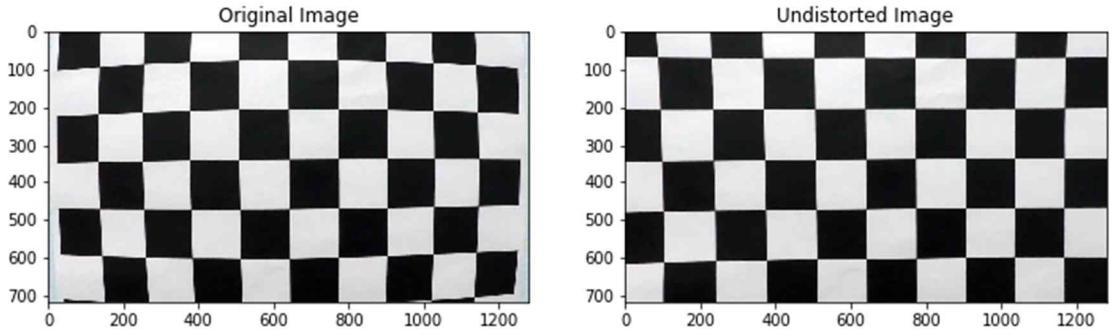
Camera Calibration

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 " advanced_lane_finding.ipynb".

   I started by preparing "object points", which were the (x, y, z) coordinates of the chessboard corners in the world. I assumed that the chessboard was fixed on the (x, y) plane at z=0, such that the object points were the same for each calibration image. The variable objp is just a replicated array of coordinates, and objpoints were appended with a copy of it every time I successfully detected 17 chessboard corners in a test image. I could not extract the corners from the 3 images. However, i continued project because there are not any major problems converted undistorted images. imgpoints was appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

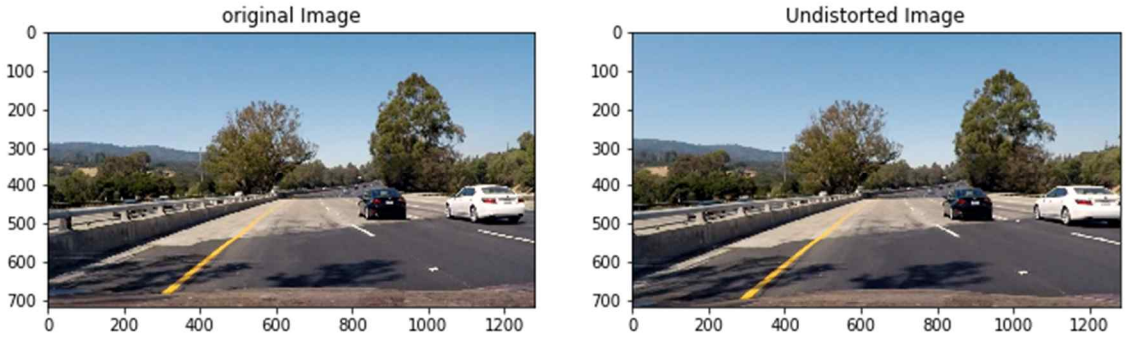
   I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result as follows:



Pipeline (single Images)

1. **Provide an example of a distortion-corrected image.**

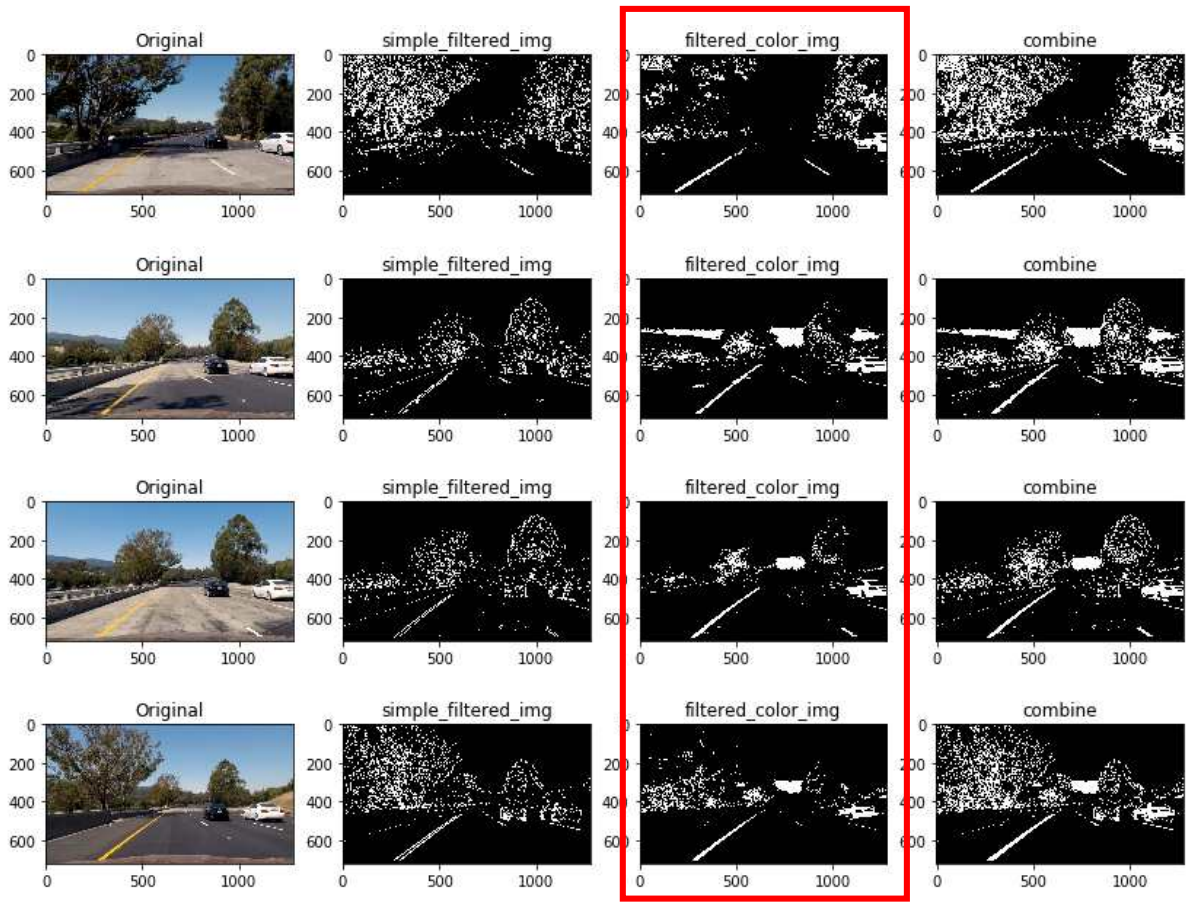
To demonstrate this step, I will describe how I applied the distortion correction to one of the test images as bellows:



2. **Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.**

   I spent a lot of time in the binarization (preprocessing) phase. Sobel, canny, color space, and simple filter (cv2.filter2D) were combined to extract only lane information. However, when I applied several filters to reduce the noise, the lane information also decreased. And since the edge contains many lane-like noise, the color-based filter is finally selected. (It is confirmed that most images contain lane information even if only color filters are used.) After converting the color filter to the HLS color space, the white lane was extracted from the L color space and the yellow lane was extracted from the S color space. The threshold value was determined heuristically. (Color filter steps in the 4th cell in advanced_lane_finding.ipynb file).

   The Test Results are as follows:



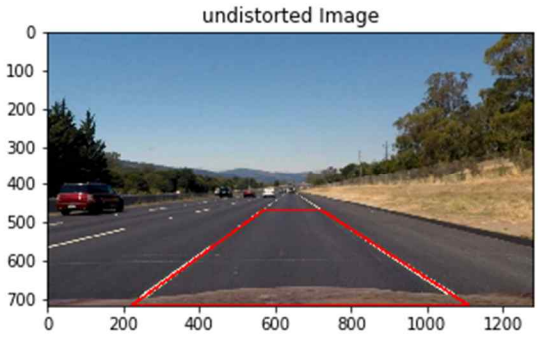
3. **Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.**

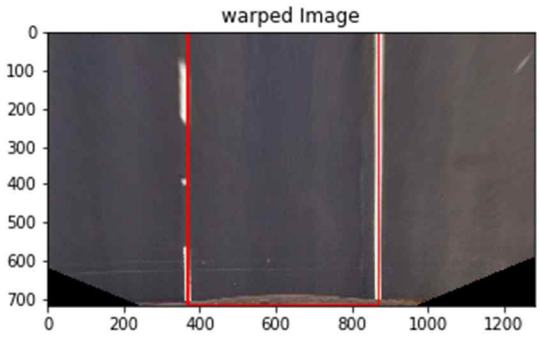
The code that performs the perspective transform is included as a function in the class we created to do camera calibration. (in the second and 5th code cell of the IPython notebook located in " advanced_lane_finding.ipynb) Function uses cv2.warpPerspective and generates perspective transform and inverse perspective transform by generating a homogeneous matrix from dst to src from src to dst.

This resulted in the following source and destination points:

Source	Destination
220, 720	370, 720
570, 470	370, 0
720, 470	870, 0
1110, 720	870, 720

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.

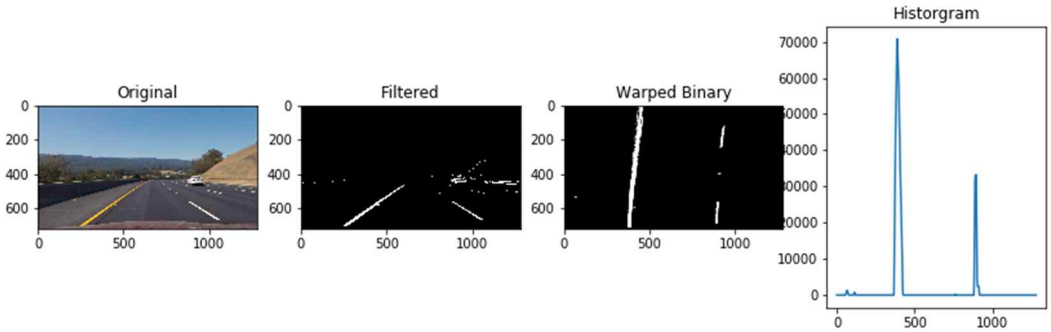




4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

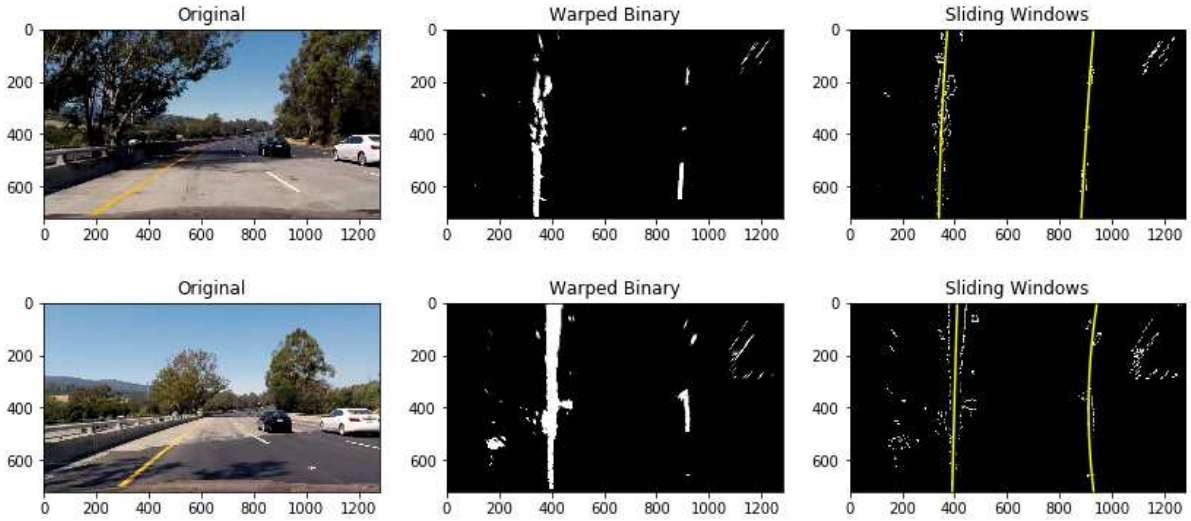
Very Helpful **(Reference: Lession15(Advanced Techniques for Lane Finding))**

I obtained a perspective transform of the the camera images converted into binary images using the warp() function. The lanes lines can be easily separated using histogram. the result is as follows:

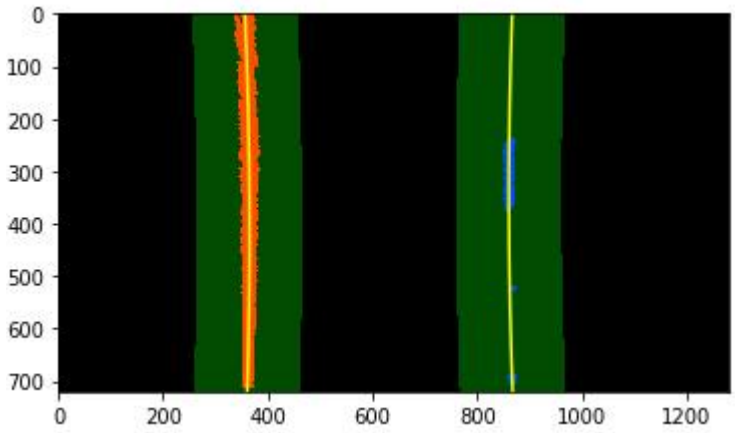


1) Plot the histogram of the binary warped image which gives a plot of the frquency distribution of the non-zero pixels in the image. The histogram provides information regarding the position of the lane-lines. The x and y position where the lane lines are present are identified and stored in the leftx_base and rightx_base.

2) Follow a sliding window search strategy to identify the x and y positions of all non-zero pixels in the image in each window and search within a margin around the leftx_base and rightx_base positions. 9 sliding windows are used to search along the height of the image. This is implemented inside the sliding_search() function in the 7th code cell of the advance_lane_finding.ipynb notebook.



3) A margin search is used to apply the sliding window within a margin of +/- 100 pixels around the leftx_base and rightx_base found in the particular sliding window. This assumption is practical since lane lines are not found randomly across the image. I stored the positions of the left lane line points and the right lane points and fit them using a 2nd degree polynomial. This is achieved inside the margin_search() function in the 8th code cell of the advance_lane_finding.ipynb notebook.



5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

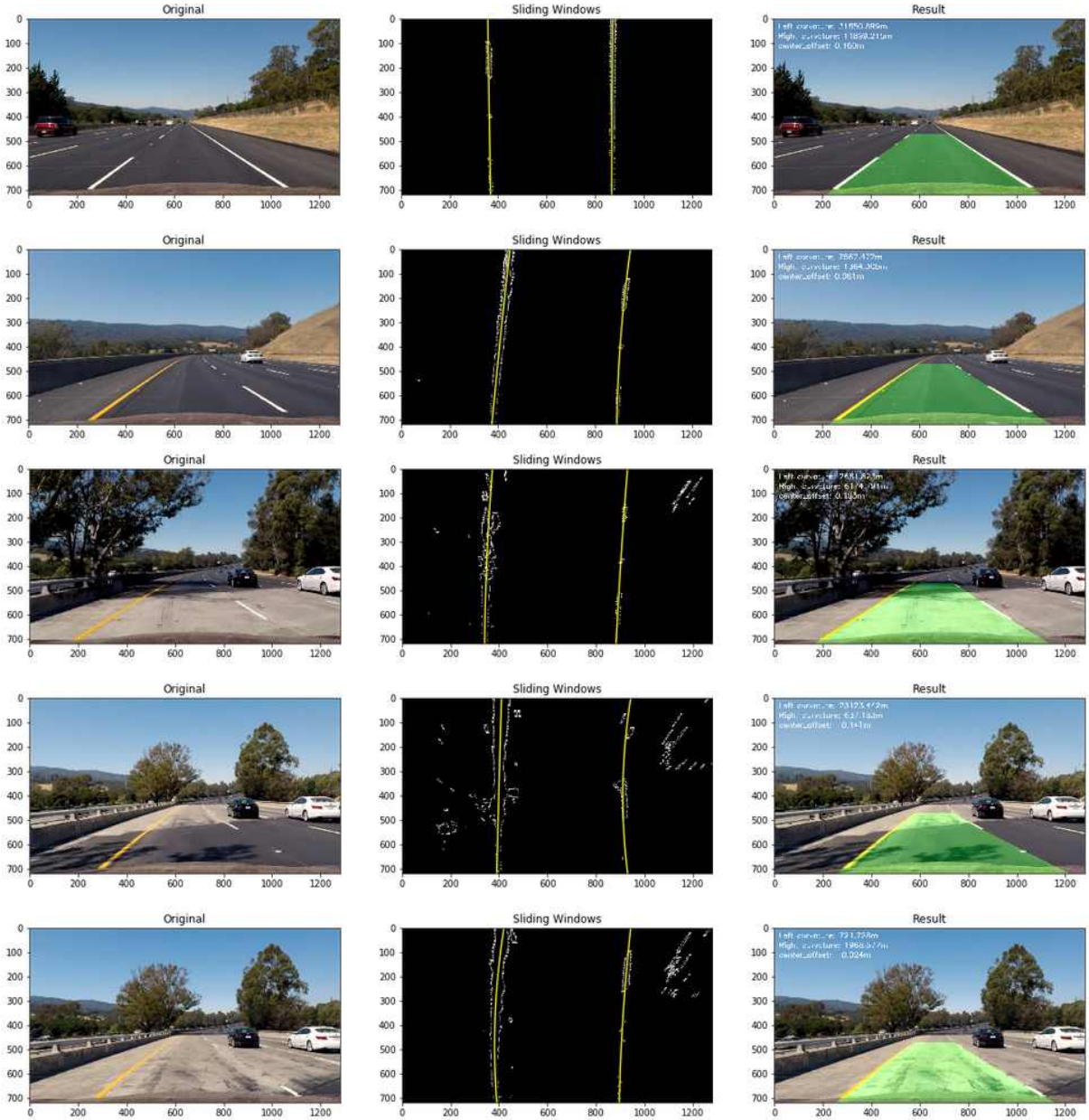
Very Helpful **(Reference: Lession15(Advanced Techniques for Lane Finding))**

For calculating the radius of curvature it is necessary to find a relationship between the distance between 2 pixels in the image and the actual distance between 2 points on the road. Assuming the meters per pixel in the y direction and x direction to be 30/720 and 3.7/(Lane width pixels) respectively the value of the lane line points on the road was obtained. The points were fit into a 2nd degree polynomial. The position of the vehicle with respect to the center was calculated by subtracting the mid-point of the width of image from the mid-point of the x-values of the left lane and right lane. (in the 9th code

cell of the advance_lane_finding.ipynb notebook)

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

   The region between the 2 lane lines was colored in green so that the lane area was identified clearly. Here is an example of my result on the test images:



Discussion

1. **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?**

   At first, I thought it was hard to detect lanes that were robust to lighting changes based on color models. So I tried to detect the lane based on the edge but the edge detection included pavement, shadow, and outliers as well. So I made a binary image by combining color model and edge in the process of binarization. This is because color model based and edge based lane detection are supposed to complement each other. (Lane information not detected on a color model basis can be found on an edge basis, and lane information not detected on an edge basis can be found in a color model.) However, when searching the whole image for every frame, various filters are added every time an error occurs and the code becomes dirty.

   Therefore, instead of continuing to update the preprocessing process, I decided to make the preprocessing process a candidate for the lane based on the color model, and added the margin_searching part and the average filter part.

   And, when using np.polyfilt in fitting lane, we confirmed that it does not fit properly when outlier points are included. I think it's a good idea to use RANSAC to solve this problem. I will expect that the performance will be better if we combine the average filter and RANSAC in the future.