Advanced Lane Finding Write-up

Christopher DiMattia

April 31st, 2020

Summary

The main goal of this project is to create an image pipeline that can find car lanes, the lane curvature and the car offset from the middle of the lane. Multiple computer vision processes and equations are used to accomplish this with the main steps outlined below:

1. Camera Calibration
2. Threshold binary image
3. Perspective transform
4. Lane pixel detection and polynomial fitting
5. Calculation of radius of curvature and determination of car offset relative to lane center
6. Plotting of results on images and video (validation)

All the code is found in: CarND-Advanced-Lane-Lines/Final_Project_Code.ipynb

Camera Calibration (camera calibration rubric #1)

The camera was calibrated to account for distortion from the lens. This was accomplished by looping through images of a chess board at different angles and distances and doing the following:

1. Using the cv2.findChessboardCorners function to create a matrix of internal chessboard corners coordinates
2. Using the cv2.calibrateCamera function to find the camera matrix (matrix to transform from 3D to 2D) and distortion coefficients (matrix to transform image back to "normal" to accord for radial and tangential distortion).

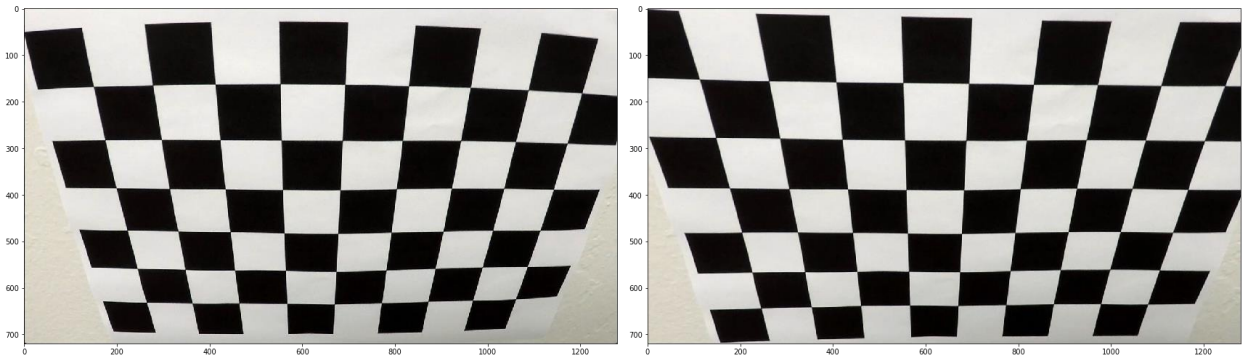

Figure 1. Un-calibrated (left) and calibrated image (right)

Pipeline (Test Images) Rubric #1

Figure 2. Distorted (left) and undistorted test image (right)

Threshold Binary Image (Pipeline (Test Images) Rubric #2)

The undistorted image is processed by filtering out and removing any pixels that are not “very” yellow or white so only yellow and white lane pixels are left. Then only the area of interest is kept. The area of interest is a quadrilateral in front of the car where the lane lines would typically be seen.

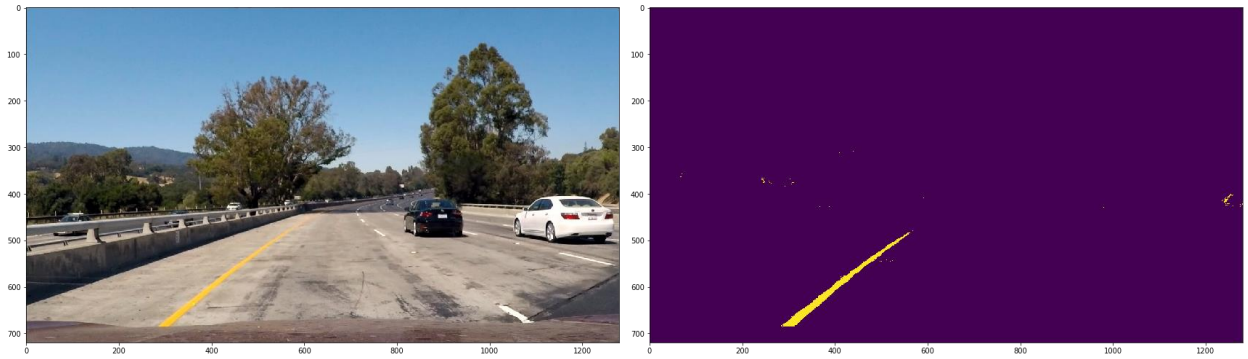

Figure 3. Test image (left) and masking everything that isn’t yellow (left)

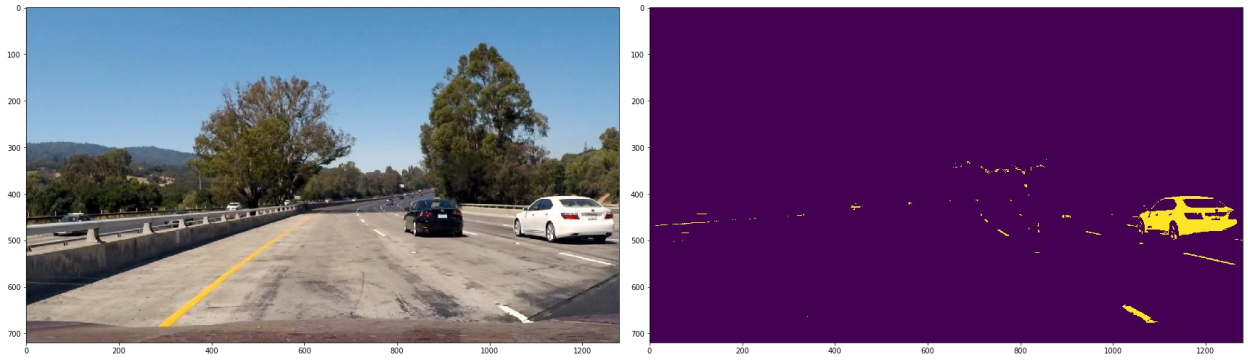

Figure 4. Test image (left) and masking everything that isn’t white (left)

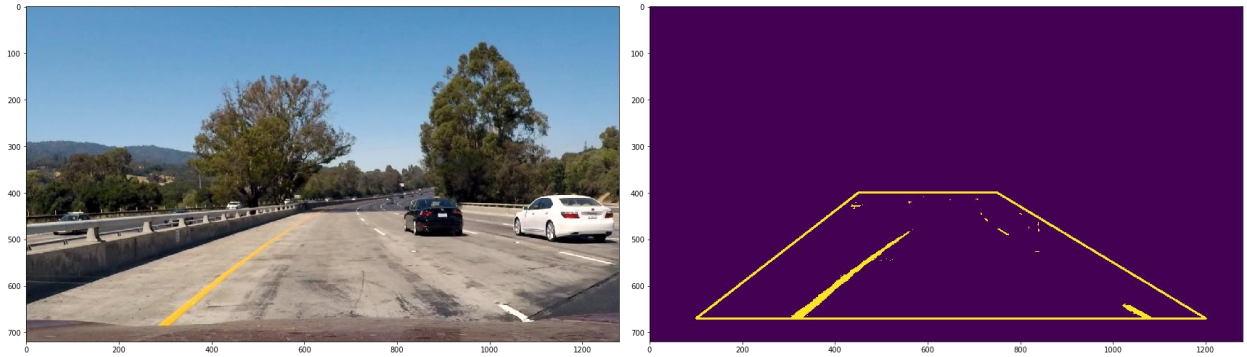

Figure 5. Test image (left) and combined yellow and white images with only area of interest (left)

Perspective Transform (Pipeline (Test Images) Rubric #3)

The binary threshold is then transformed from the normal perspective to a "bird's eye view" using a perspective transform. This is accomplished by identifying four points on the untransformed image of a completely straight lane and four of the same points had they been taken from a bird's eye view. Then using the unwarped points as inputs the cv2.getPerspectiveTransform function returns a matrix that transforms all imagines from the unwarped to the warped via the same transformation. Then the cv2.warpPerspective function is used with the matrix and image as inputs to get the final perspective.

Figure 6. Test image (left) and perspective transformed image (right)

Lane Pixel Detection & Polynomial Fitting (Pipeline (Test Images) Rubric #4)

The lane pixels have to be identified as either left or right to fine the lane. The first method assumes no pixels are detected and the second finds pixels based on a previous frame.

<u>Assuming no prior pixels (method #1: find_lane_pixels)</u>

1. Create a histogram summing the non-zero pixels detected in the x direction but only for the bottom third of the image
2. Find the x position with the most points on the left and right side. These are now the starting points to begin searching for pixels

3. Create a window of interest to add pixels to a final list. Make the width slightly larger than a lane and the height is based on how many windows you wish to create.
4. Loop through windows starting at the bottom adding non-zero pixels to a final list. If the amount of non-zero pixels in a window is large enough re-center the next window on the average of the current window. The windows should follow the lane as the windows "move up" the image, adding the left and right lane pixels to a final list.

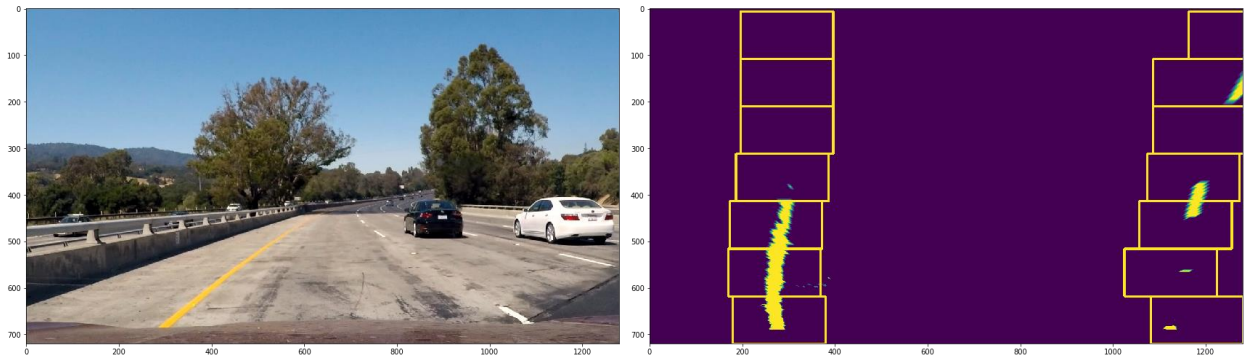

Figure 7. Test image (left) and the binary threshold (right) with boxes showing where pixels were found.

Assuming prior pixels (method #2: search_around_poly)

1. Using the pixels from a previous search, create a polynomial with np.polyfit.
2. Add any pixels around the created polynomial with a decided margin of error
3. Sum the non-zero pixels and return as the final output

Fit polynomial

Using the pixels detected in either method #1 or #2 the final list of x and y coordinates are created using np.polyfit to find the polynomial coefficents which are then plotted against all the y values in an image which in this case are from 0-720.

Calculation of radius of curvature and determination of car offset relative to lane center (Pipeline (Test Images) Rubric #5)

The radius the car is driving is found using the below equations

$$(1)\ f(x) = Ax^2 + Bx + C$$

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

Using the A and B values found from before, simply plug them into the equation 2 to determine the radius. The A and B values from the left and right lanes can be averaged, but the accuracy isn't good enough in this program to make a significant difference.

To determine the offset, use equation 1 to determine the x values for the left and right lane (using a y value of 720 which is the bottom of the image). Then assuming the middle pixel is 640 which is the middle of the image calculate the offset using the below formula.

Figure 8. Final Pipeline image (Pipeline (Test Images) Rubric #5)

Final Video & Discussion

The final video can be found in the Jupyter directory but I posted a youtube link below as it hasn't worked in the past. (Pipeline (video) Rubric #1)

https://www.youtube.com/watch?v=NGPyc-OpwwA&feature=youtu.be

There are multiple areas for improvement and reasons the pipeline could be prone to failure which I list below. (Discussion Rubric #1)

Areas for improvement

- The binary thresholding is extremely simple and could use more filters. The filters could change based on overall lighting or the total amount of other RGB/HSV detected in an image. For instance the filters for day and night should be different or if there is a red tint due to a sunset.
- The binary threshold could use more types of filters such as HSV instead of just RGB which may be better

- The binary threshold seems like it could use a combination of RGB, HSV and different thresholds to determine if the lane lines exist. For instance if you have two separate filters you could make the thresholds very tight/stringent and then search around pixels that are accepted in both. This way you can maintain strict rules, but you can still search around them.
- You could incorporate complex Hough transforms for find curved lines
- The perspective transform could be used with a perfected centered car as I noticed the car appears to be slight off center to the lane which might lead to a warped perspective transform.
- When finding the lane pixels and the polynomial coefficients you could make assumptions about the maximum curvature based on the speed of the car. In theory you won't have an extremely sharp turn at 90 mph and if you do you are either speeding, misidentified the lane lines or have some other major problem. So with that in mind you could set the polynomial coefficients to have maximum and minimums if you're confident in the GPS system that tells you where you are and how fast you're going.
- You could research for new lane pixels not based only on the number of pixels detected but also if the polynomial you detected is greatly different than the previous frames. In this project I only re-searched for lane pixels if there were not enough even if the curve was drastically different than the last frame.
- You could have multiple cameras to cross check your values as bumps in a road might drastically change the calculated curvature

<u>Areas prone to failure</u>

- Images with different lighting as the binary threshold won't work. Too bright and it will over detect and too dark and it will under detect
- Images with unclear lane lines such as snowy conditions, dirty roads, etc as the binary threshold won't work
- Any significant angle/elevation change will completely mess up the perspective transform as it assumes a flat road
- Any images with merging lanes will be highly confused by lane lines that split
- Any images with cars in front could have the lanes highly obstructed and might fail depending on the cars position