# Advanced Lane Finding on the Road

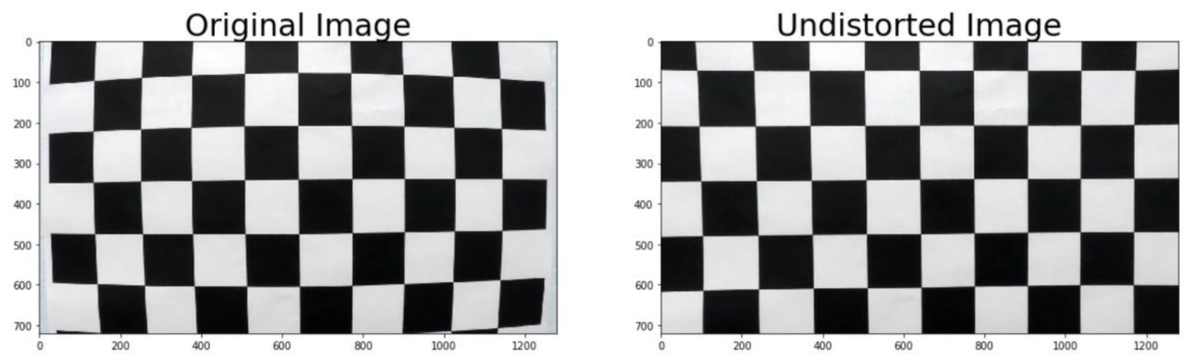
## Goals

- 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 threshold 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

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 obj_points will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. image_points 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 obj_points and image _points 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:

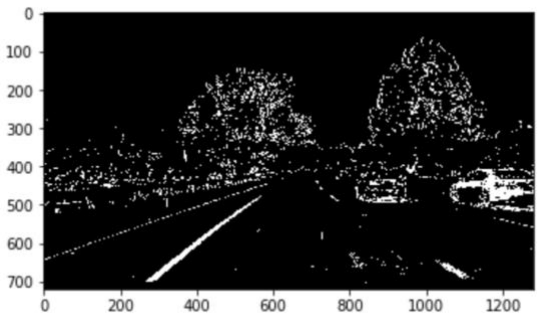


## Pipeline (single images)

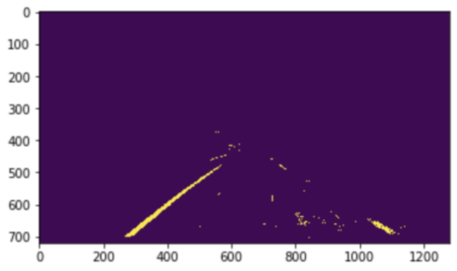
Pipeline consists of six steps and they are as follows

1. Undistort the image using the values you calculated in camera calibration phase.

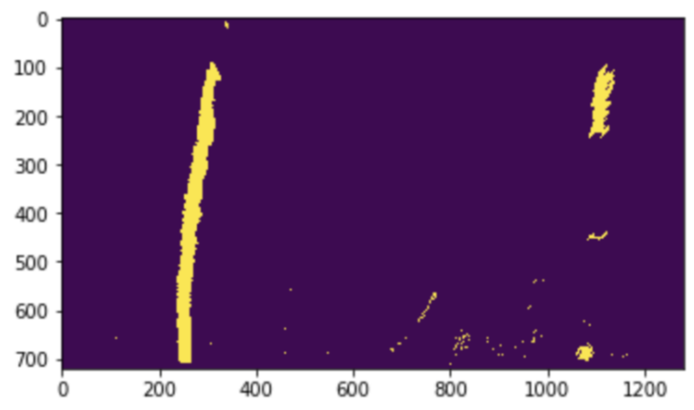
2. Thresholding
   a. Convert to HLS color space and separate the S channel.
   b. Convert to gray scale
   c. Find gradients in x direction.
   d. Threshold x gradient and color channel.
   e. Combine the two binary thresholds.



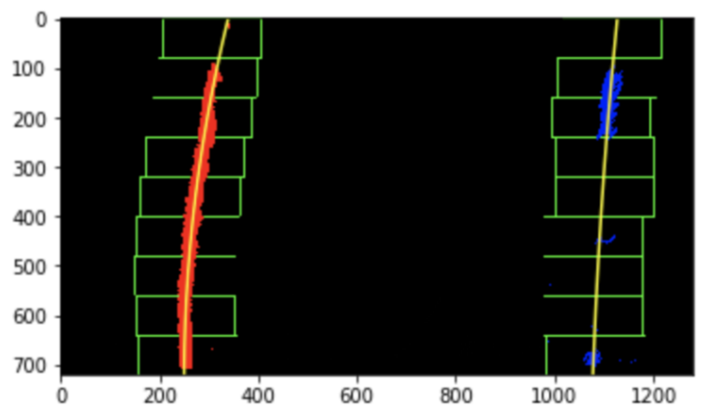
3. Extract the region of interest.
   a. Usually the road appears below in the image and does not appear above.
   b. Also, it appears towards the center of the image.
   c. So, extracting relevant portion of image helps us to eliminate all the edges in rest of regions in the image.



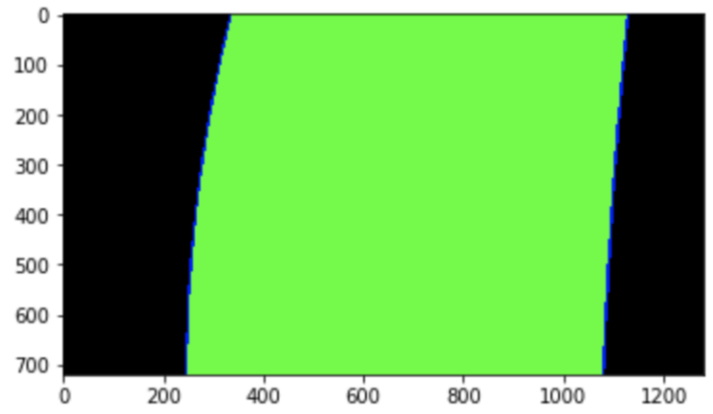
4. Apply perspective transform to get bird eye view



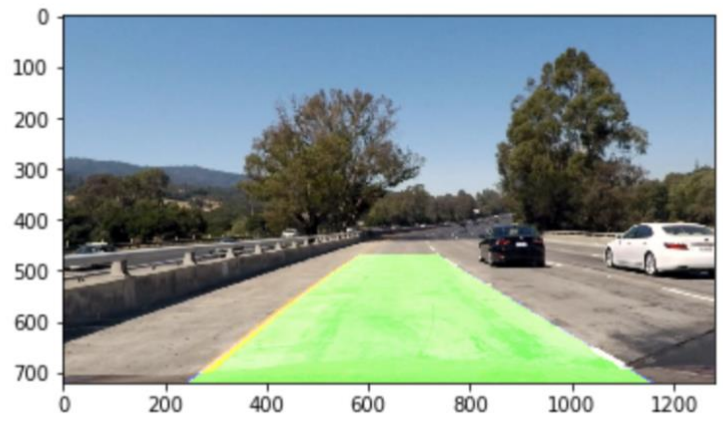
5. Find lane pixels
    a. Take histogram of bottom half of image.
    b. Find peaks in left and right halves of the histogram.
    c. Choose the hyper parameters for number of sliding windows, width of the sliding window, minimum no of pixels found to re-center window.
    d. Identify x and y positions of all non-zero pixels in the image.
    e. Step through windows one by one.
        i. Identify window boundaries in x and y
        ii. Draw the windows on the visualization image
        iii. Identify the nonzero pixels in x and y within the window
        iv. Append these indices to the lists
        v. If you found > minpix pixels, recenter next window on their mean position
        vi. Concatenate the arrays of indices
        vii. Extract left and right line pixel positions
6. Fit Polynomial



7. Measure curvature of pixels
8. Highlight the area between the lanes



9. Display the highlighted area on top of image



## Link to output on video

https://drive.google.com/file/d/1MyQarhEFBFu9vWtCbYfiDZ9lmlgmqgV_/view?usp=sharing

## Discussion

- Had hard time finding the right src and dst points for perspective transform

## Improvements

- I see that the algorithm shakes a liitle bit when the lane color changes and could be improved in that a bit