Advanced lane detection

Introduction

Following are the steps of this project.

- Compute the camera calibration matrix and distortion coefficients from a set of chessboard images.
- Correct distortion on the images
- Apply color and gradient thresholding to focus on lane lines
- Apply perspective transform to obtain bird's eye view images
- Detect lane pixels with sliding widow technique
- Compute the curvature of the lane and vehicle position with respect to center of the road
- Warp and draw lane boundaries on image as well as lane curvature information.

Camera Calibration and Image Distortion Correction

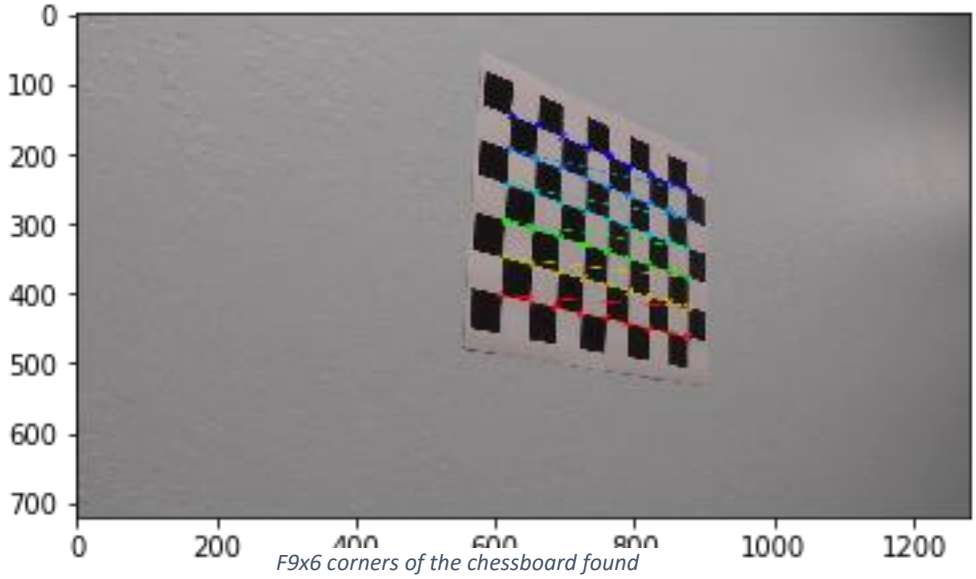
Camera calibration is the process of measuring all the important characteristic of o camera in order to understand how the three dimensional world get mapped down onto two dimensional sensor inside your camera and ultimately to the camera's output image.

To calibrate camera you need the how 3D point in the world get mapped down onto 2D.

The first step is to choose calibration pattern. In this project we will be using chessboard pattern. The purpose is to provide a known geometry that you can take pictures of from different angles using your camera and use it as a reference when mapping 3D world point to 2D image point.

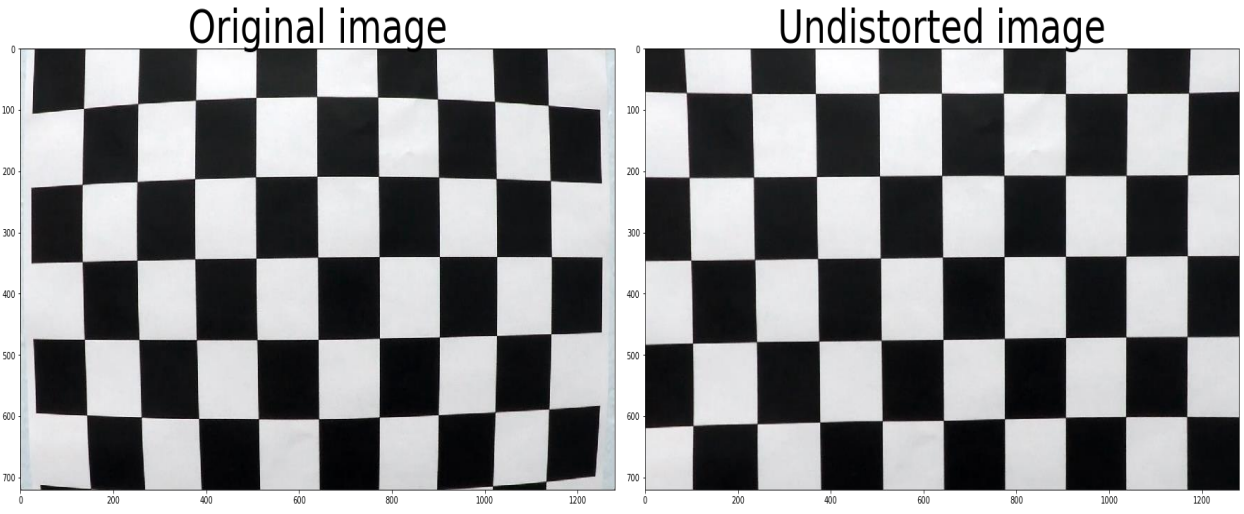
In this project we will be using OpenCV; an open source computer vision library to perform camera calibration and distortion correction in three main steps:

1. Use CV2. **findChessBoardCorners()** to find corners in the chessboard images and aggregate arrays of Image points (2D image plane point) and object points (3D world points)



F9x6 corners of the chessboard found

2. Use the OpenCV function cv2.calibrateCamera to compute the **calibration matrices** and **distortion coefficient.**
3. Use cv2.undistort() to undistort test images.



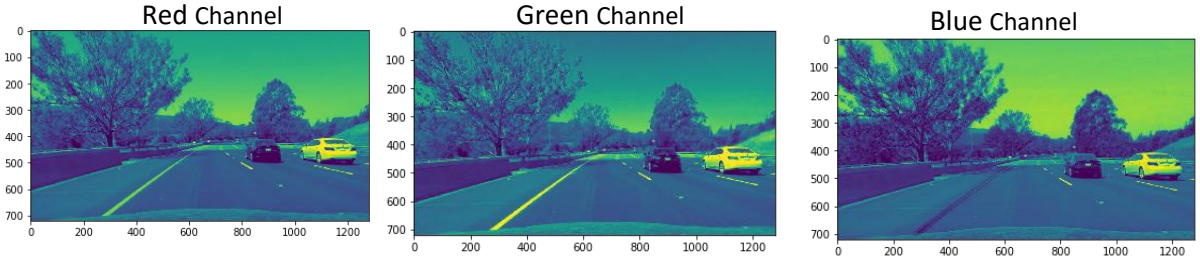
Thresholding

We use the thresholding technique to highlight parts of the image that fall within a specified range, hence help us to detect lane lines on the road images. In this project we combined both color thresholding and gradient thresholding to complement each to other to enable us better detect lane lines.

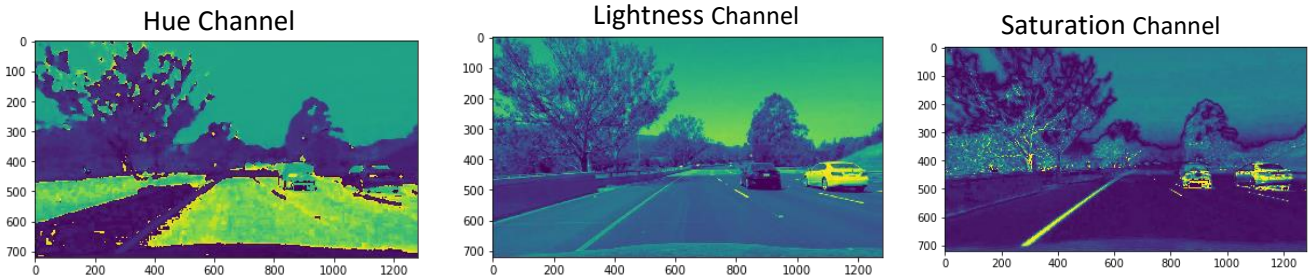
Color Thresholding

Here we explore various color spaces and channels then apply one that increases our chances of detecting the lane.

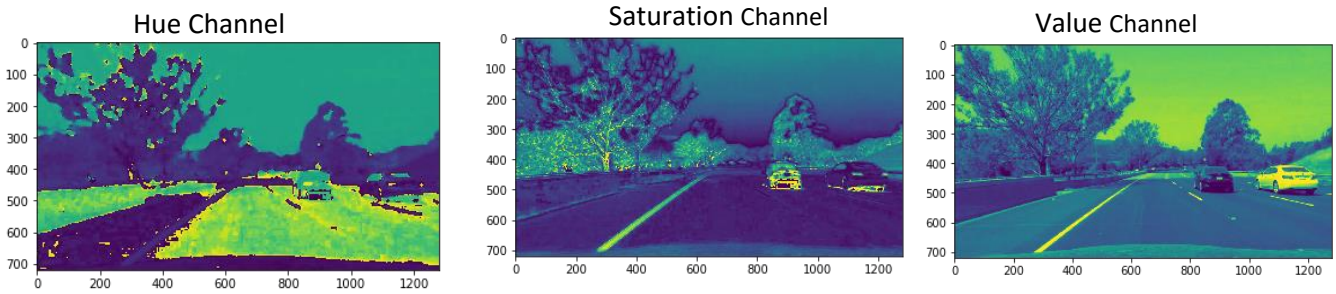
With RGB color space, the blue channel is worst at identifying yellow and white lines while the red channel provide better result.



With HLS color space, the hue channel gives a very noise result while the saturation channel seems to produce the strong result.



Likewise in HSV color space, the hue channel gives a very noise result while the saturation channel seems to produce the best result.

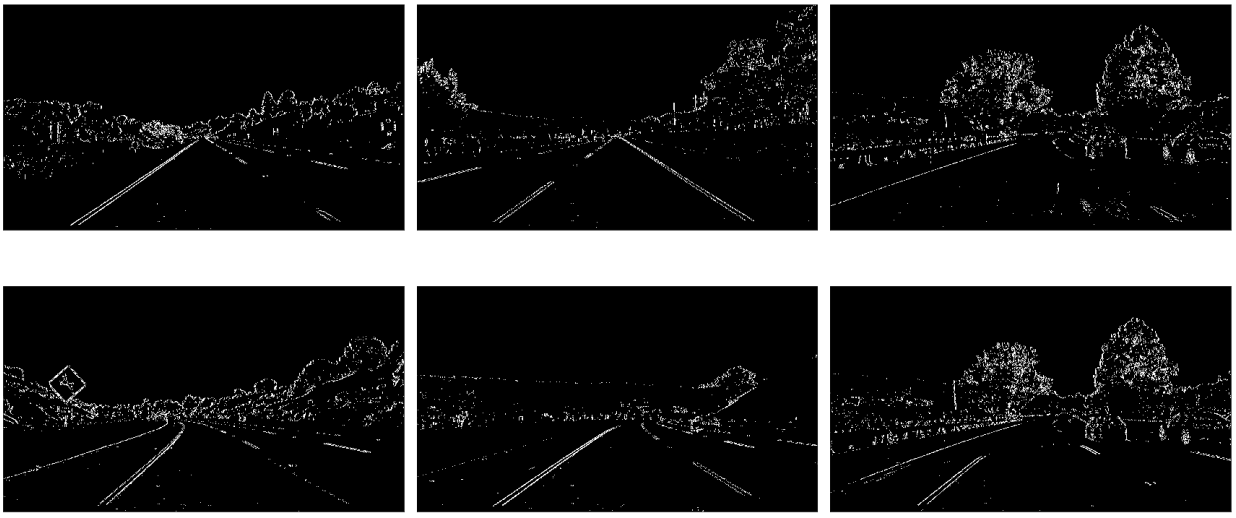


Therefore we choose use saturation channel of the HLS color space because it gives better result at highlighting yellow and white line on the lane.

Gradient Thresholding

We use Sobel operator to compute gradient of the road images. What Sobel operator does is to perform derivative on our images in either x and y direction or both, thereby, tracing outline of the edges that correspond to the most sharp changes in intensity.

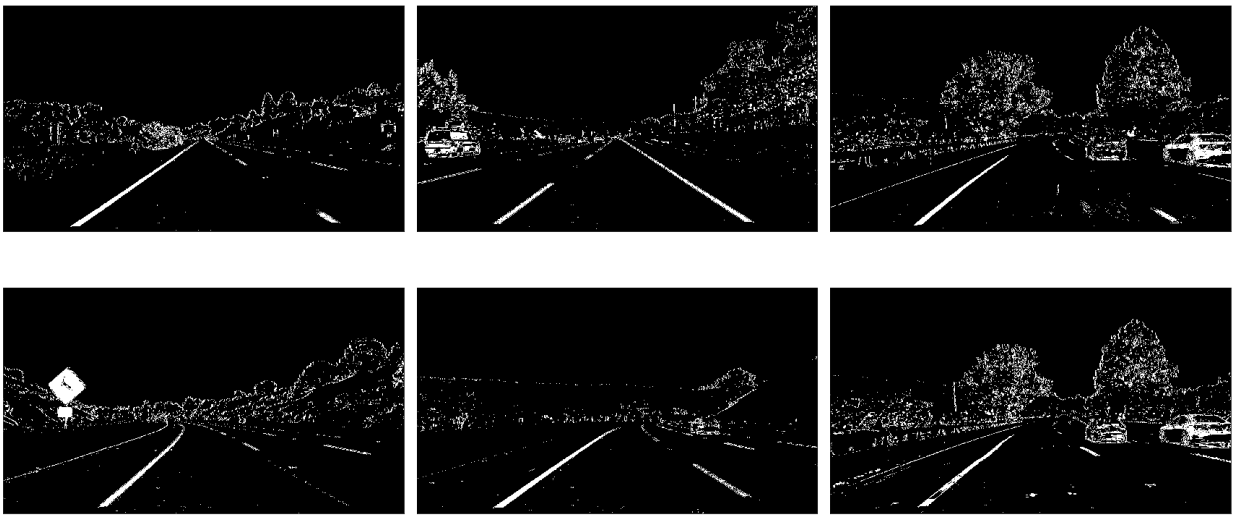
Gradient thresh images



Combining Color Threshold and Gradient Threshold

After performing both color and gradient thresholding, we combined the both to complement each other to detect lane in a robust manner.

Gradient and Color thresh Combined



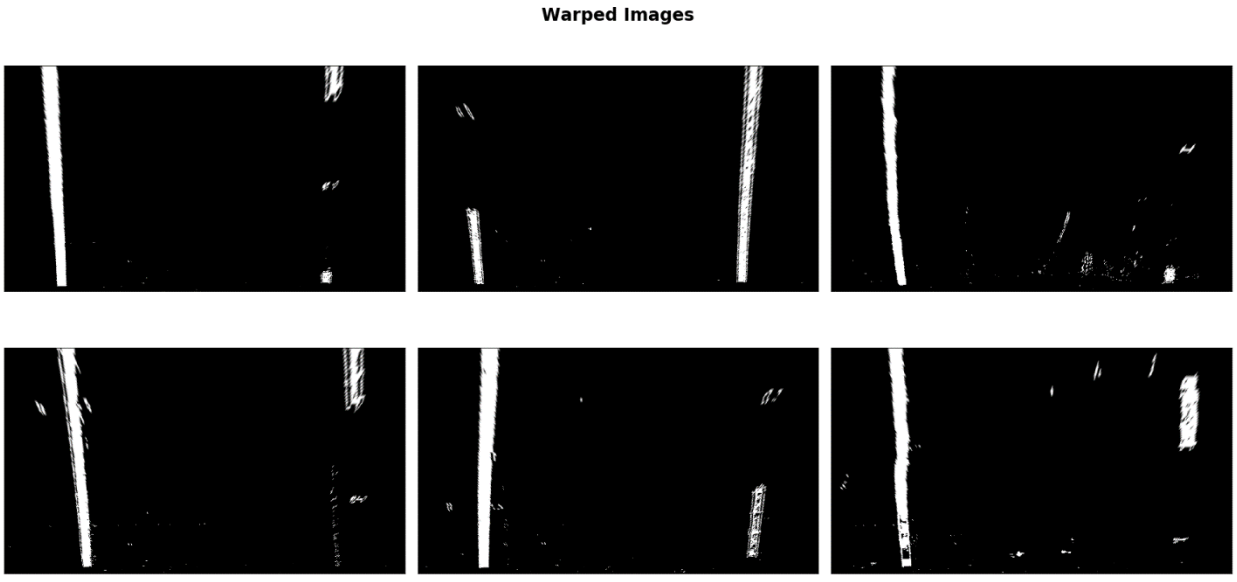
Perspective Transformation

The perspective transform deals with the conversion of 3D world into 2D image. In this project we apply perspective transform to our road images to shift them from the perspective seen from camera to one where you're looking down on the world from above (Bird's Eye View).

To perform perspective transform we need to define four corners that form trapezoidal region that will go through a perspective transform to convert into a bird's eye view.

We perform perspective transform on our road images using OpevCV using following steps:

1. Define source points, four corner of the trapezoidal region on the image
2. Define four destinations points in the same order as source points
3. Use **cv2.getPerspectiveTranform()** to get the transformation matrix
4. Use **cv2.warpPerspective ()** to apply the transformation matrix and warp the image to a top-view.

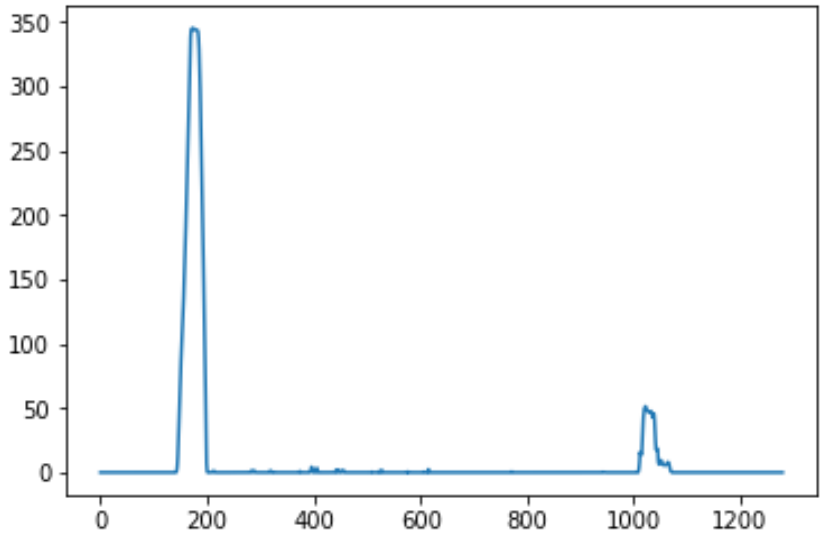


Lane-Finding

The lane detection is achieve using a sliding window technique.

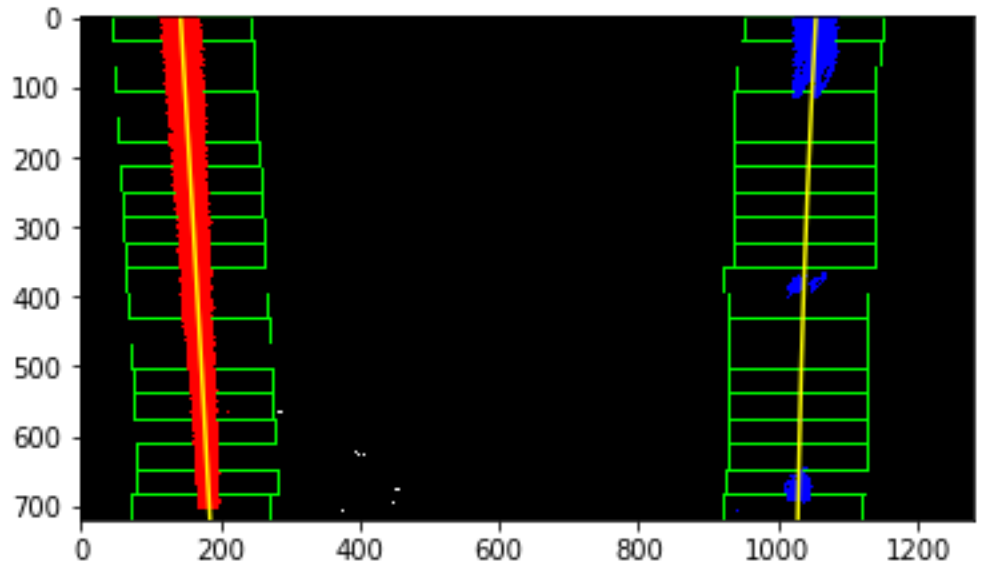
Histogram

We use histogram to find which pixels are part of the lane. We take histogram across the columns of the lower half of the warped binary image. We get a graph with two peaks, the two noticeable peaks of the histogram indicate the x positions of the base of the left and the right lane which we can use them as a starting points.



Sliding windows

The sliding window algorithm divide the image into vertical stack of narrow horizontal slices. One window on top of the other that follows the lane up the frame. The pixel inside the window are selected as pixel of interest and added to the list of points representing lane. We average the x value within the sliding window to give us the base point of the next window above. This process is repeated over and over until we reach the top of the frame. This way we have accumulated all the pixels that we are interested in, we then feed them into the numpy function **polyfit()** to obtain the coefficients of the second degree polynomial.



Lane curvature and Vehicle positions

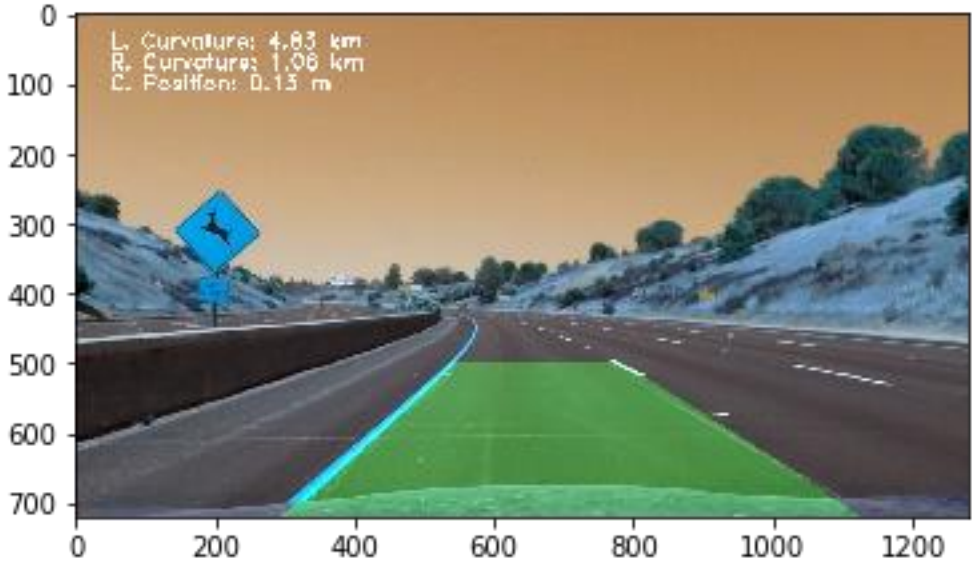
After we fitted the curves for each lane line with a second degree polynomial function; $Ay^2 + By + C$, we get A, B and C which we used them to compute the radius of curvature of each lane line the following equation.

$$F(y) = Ay^2 + By + C \qquad R_{curve} = \frac{[1+(dx/dy)^2]^{3/2}}{|d^2x/dy^2|}$$

To estimate the vehicle position with respect to the center of the road. It is a simple average of the pixel coordinate of the two lanes at the bottom of the image minus the pixel coordinate of the center.

Draw lane Area

Finally, we unwarp the bird's eye view to trapezoidal region we selected to earlier to mark our lane and overlay it on the original image. We also annotate information about lane curvature and vehicle's position.



Conclusion

In this project we built computer vision pipeline for advance lane line detection. We started by correction camera noise and distortion on our road images, we then isolate our lanes from the image using color and gradient thresholding. After that warp the images from camera view to bird's eye view to make it more easier to detect curvature of the lane lines and we used these lane lines pixels to find best fit curves parameters using sliding window and numpy polyifit function. Finally draw the lane region onto the images.

This project can be improved by exploring other color spaces such as YUV, LAB and etc for better color thresholding and apply other relevant advance computer vision techniques not cover by this project.