# Advanced Lane Finding

## 1. Purpose :

The purpose of this project is to use computer vision techniques to identify the Lane lines on road, calculate the radius of curvature and vehicle position with reference to lanes.

## 2. Project Inputs:

The inputs for the project are provided by Udacity. The inputs include following items –

a) Chess board images for Camera calibration
b) Test images for road conditions
c) Videos for project and for challenges.
d) Folder structure in Jupyter notebook

## 3. Project Outputs :

i) To fulfill the project rubric following artifacts are expected –

a) Distortion – corrected image
b) Example of Binary image result after applying color transforms, gradient or other methods
c) Example of Perspective transformed image
d) Output image with identified lane markings, radius of curvature and Vehicle offset
e) Output video with identified lane markings, radius of curvature and Vehicle offset
f) Code for identification of lane markings on test images
g) Code for identification of lane markings on videos

ii) To fulfill the project rubric following explanations are expected –

a) How the camera matrix and distortion coefficients are calculated ?
b) How the color transforms, gradients or other methods are used to create a thresholded binary image ?
c) How Perspective transform is performed ?
d) How lane-line pixels are identified and mapped with polynomial ?
e) How radius of curvature and Vehicle position is calculated ?
f) Explanation of problems faced during the project.

4. **Description** :

**a) Camera Calibration**

The *Function1: cal_undistort()* is defined to provide an undistorted image from calculated obj points and image points. The CV function *cv2.calibrateCamera()* is used to extract the matrix and distortion values from the Chess board images.

The matrix and distortion values are then stored in distortion pickle file named as *wide_dist_pickle.p*

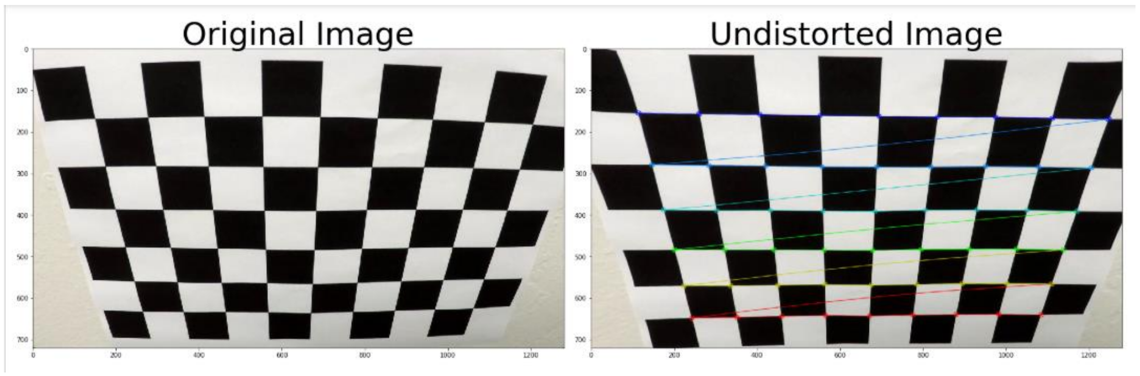
************

The *Step1: Camera Calibration and Distortion correction*, is defined to read the camera calibration images and store the undistorted calibration images. The number variables *nx* and *ny* are defined to provide inside corner values for function *cv2.findChessboardCorners()*.

The image for calibration is then converted to gray scale with function cv2.cvtColor() and is passed to function *cv2.findChessboardCorners()* for getting corners and image points.

The detected corners and images points are then drawn on calibration image with function *cv2.drawChessboardCorners().*

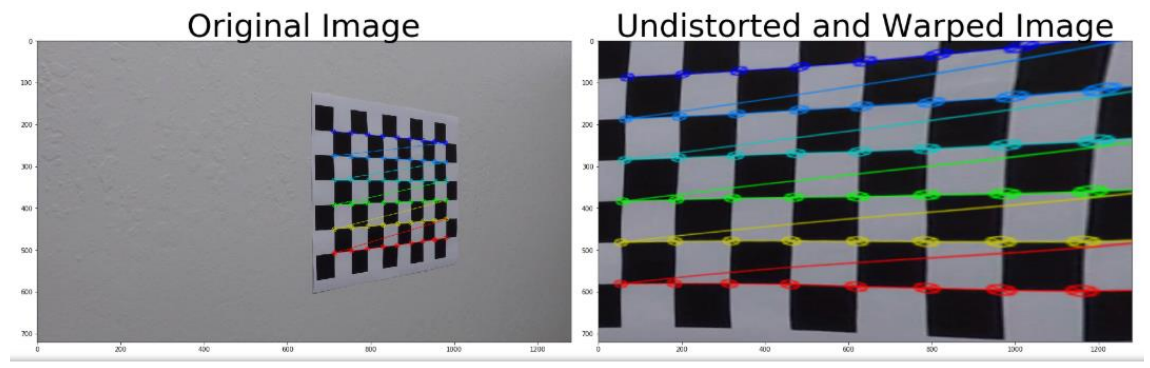
*cal_undistort()* is then called in loop from *Step1* with above steps to get the calibrated images in loop and store them in a separate folder.



The Undistorted images are then converted to warped images by calling *Function2: corners_unwarp().*

The function *corners_unwarp()* utilizes *cv2.getPerspectiveTransform*() to get the Perspective transform matrix. The Perspective Transform Matrix *M is then stored in Warp_Pers_pickle.p. The Warp_Pers_pickle.p* is not used further as the transformation for the test images is based on fixed corners identified based on pixel locations on lanes.

However below is an example of warped images for Chess board pattern.



**b) Pipeline – Test images**

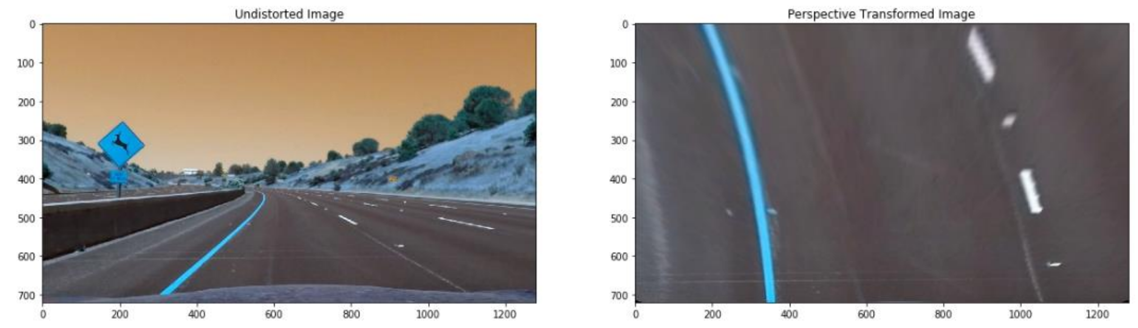
*Step 2 : Plot Undistorted images for Test images,* defines the steps needed to distortion correction for Test images. The matrix and distortion values which were calculated using the Chess board images for Camera calibration are recalled from *wide_dist_pickle.p* file.

Function *cv2.undistort() is used for distortion correction.* The distortion correction image is then saved in *"output_images/Undistorted_Test_Images"* folder.

**c) Pipeline – Perspective Transform**

Step 3: This section is to provide Perspective transformation to Undistorted images, defines the steps needed to transform the image to have a Bird's eye view. As per the project Ruberic this step is expected after Color and Gradient Transform. But to simplify the lane detection I have applied the Perspective transform prior to other steps. The Source and Destination points are defined based on the visual observation of image, and then function cv2.getPerspectiveTransform() function is used to get Transformation matrix and Revers Transformation matrix.

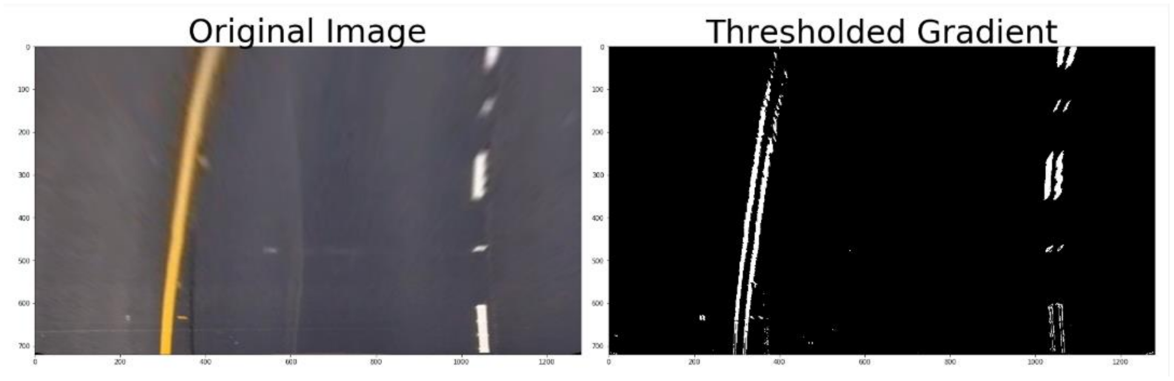
Then function cv2.warpPerspective() is applied to get the bird's eye view.

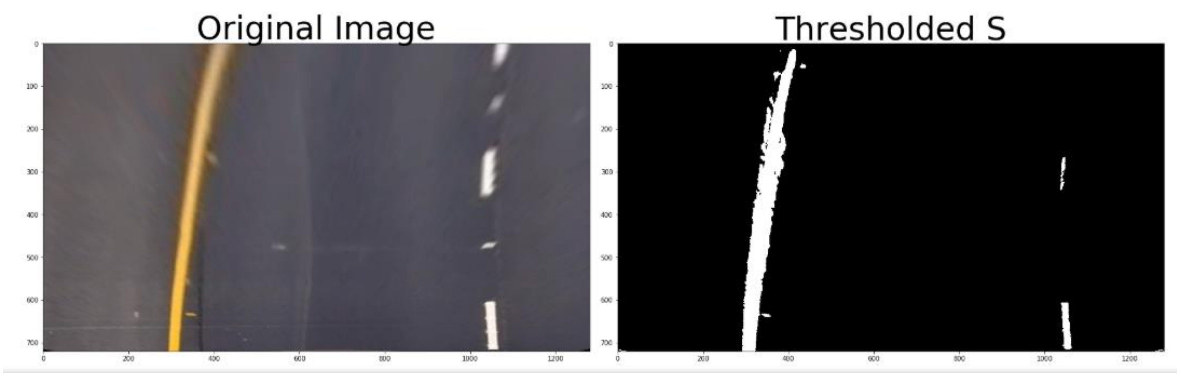


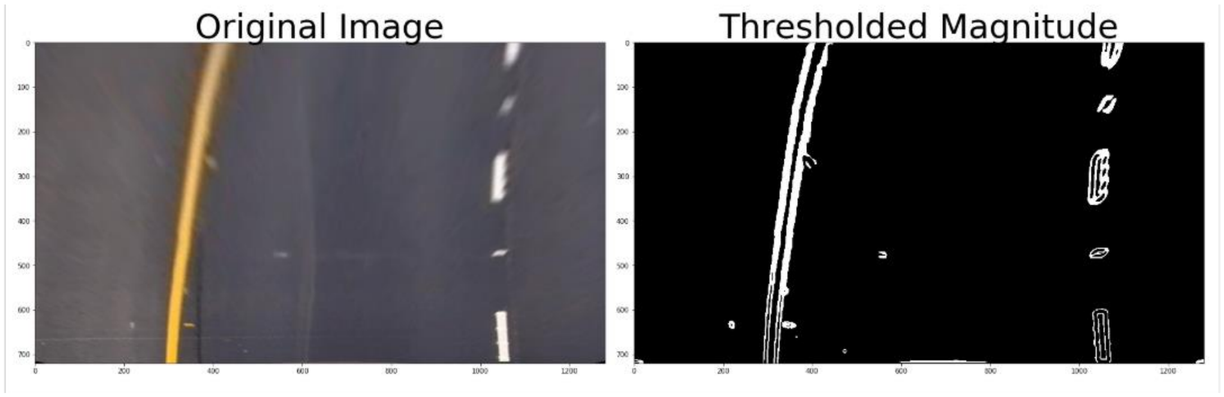
**d) Pipeline – Color and Gradient Transform**

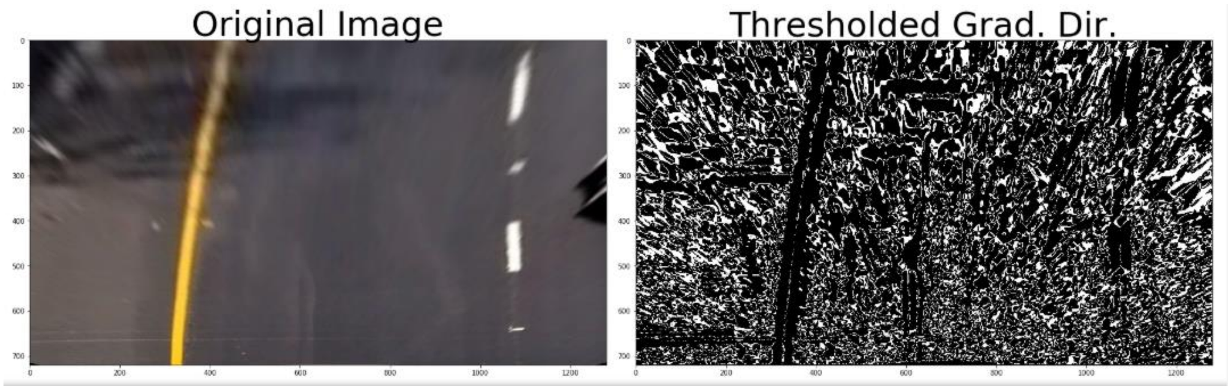
In Steps 4, 5, 6, 7 Absolute Sobel, Magnitude Threshold, Direction Threshold and Combined thresholds are implemented for understanding of concept. For video processing Step 8 with HLS thresholds is used for Lane identification.

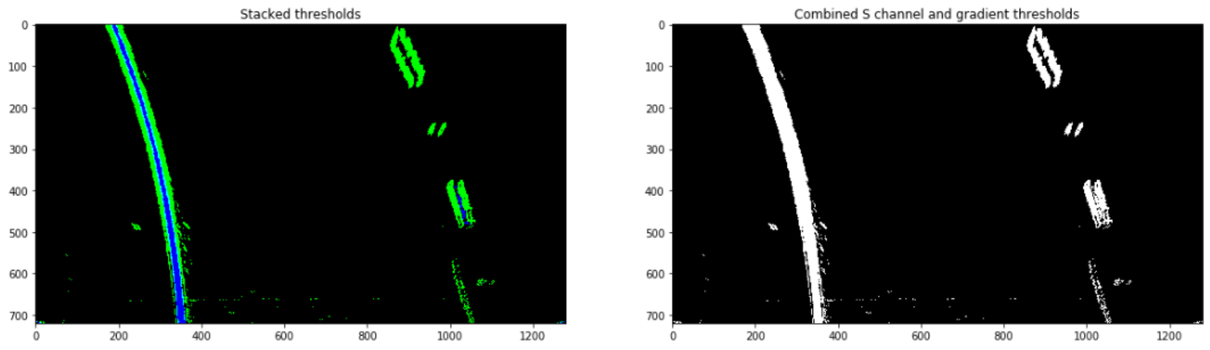
The Undistorted Warped image is converted in to HLS format by calling *Function 6: hls_select().* The image is converted into hls format with function *cv2.cvtColor(img, cv2.COLOR_RGB2HLS).*







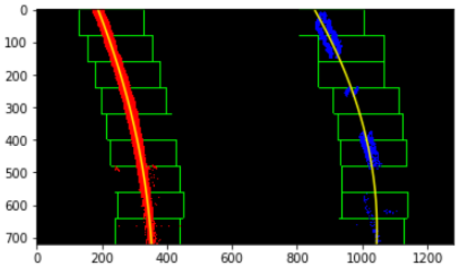




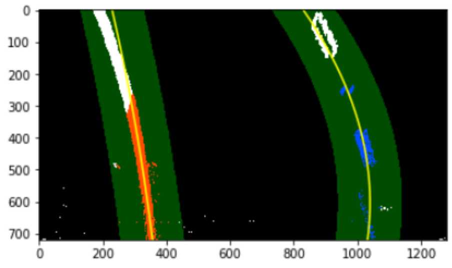
**e) Pipeline – Lane-line pixel identification**

The Lane lines can be identified with two different methods as defined in the chapters. I have defined functions *fit_polynomial()* and *find_lane_pixels()* in one cell. And *search_around_poly()* with *fit_poly()* in another cell. I defined global parameters for the right and left lane pixels and curvature related values so that they can be excessed from pipeline code.

Below is an example of output from *fit_polynomial()* function which uses the sliding window to track the curvature of lanes based on lane's mean position. Image is displayed below only for covering the project specifications. In the project final code this function is not used, instead *search_around_poly()* is used.



Below is an example of output from *search_around_poly().* In this function we take help of the left fit and right fit pixels detected from the sliding window function *fit_polynomial()* and detect the curvature within a certain defined margin. The margin is taken as 100 from the examples included in Quiz.



**f) Pipeline – Radius of curvature**

The Radius of curvature defines the characteristics of turns and bends of the road and impacts the visibility and controllability of the vehicle.

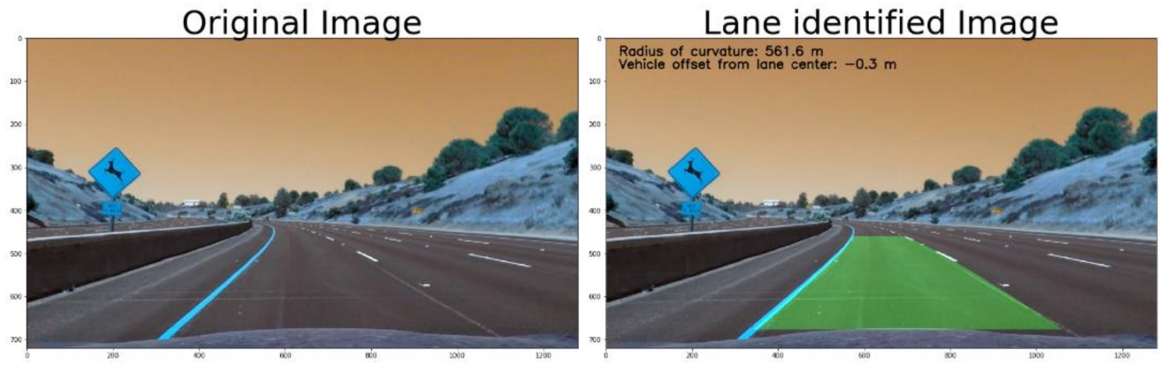
In this case the Radius of curvature is calculated by converting the values extracted from image in pixel space to world space. I utilized the meters per pixel in x and y dimensions, which was given as *ym_per_pix = 30/720*, and *xm_per_pix = 3.7/700*. These values where then used with the x and y coodinates in *polyfit()* function to calculate curvature followed by equations for Radius of curvature.

**g) Pipeline – Vehicle position**

The vehicle position from the center is determined by knowing the position where on the x plane, both the left lx and right rx lines crossed the image near the driver. Then the xcenter of the image is taken as the width/2. The offset was calculated such *(rx -*

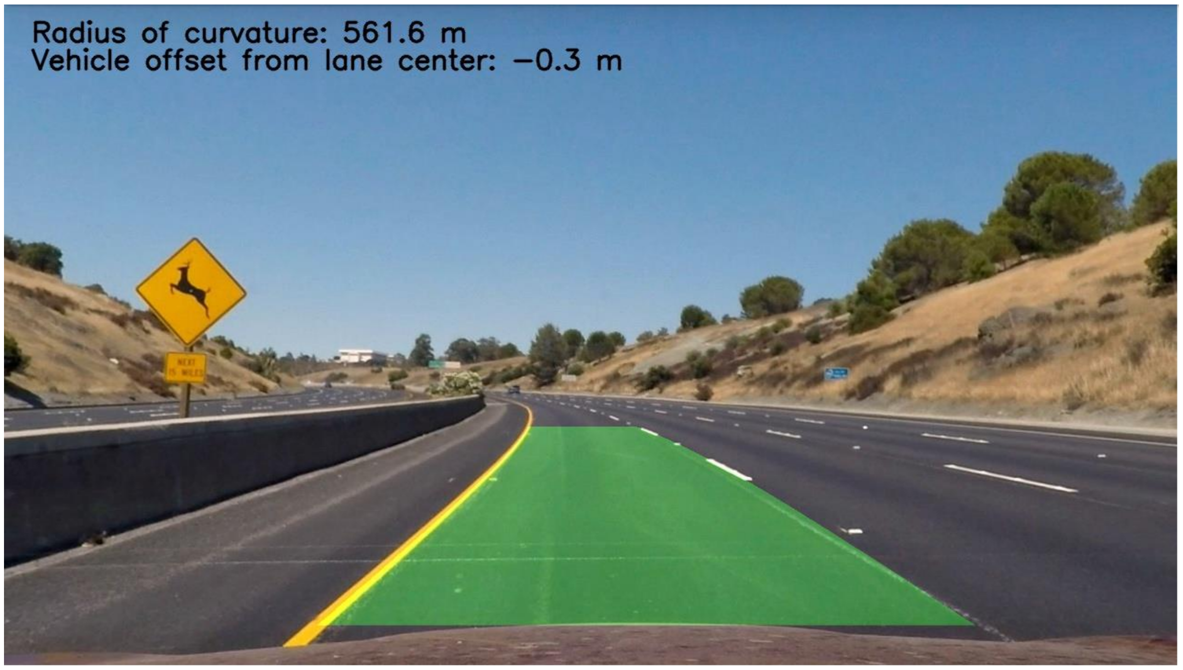
*xcenter) - (xcenter - lx)* before being multiple by *xm_per_pix*. Again for this calculation I used the *left_fit and right_fit* global variables.

The Curvature and Vehicle offset information is superimposed on the Lane identified image with *cv2.putText()* function.


Original Image
Lane identified Image
Radius of curvature: 561.6 m
Vehicle offset from lane center: −0.3 m

**h) Pipeline – Example of result for output image**

Following is an example of output image for the pipeline applied on the Test images. The other images can be found in folder *output_images/Result_Images/ Result_Img_*.jpg*


Radius of curvature: 561.6 m
Vehicle offset from lane center: −0.3 m

**i) Pipeline – video**

The pipeline video is available at *videos_output/project_video_Output.mp4.* The video is following the specification, however at two places there is sudden glitch for fill area. But that will not cause the vehicle to leave its path as the markings get back on the lane after one or two frames. I also tried the pipeline for challenge video but it didn't worked at all. Need to spend more time for challenge videos.

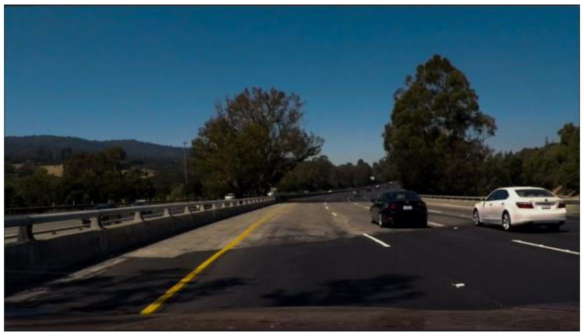
## 5. **Discussion** :

### a) **Implementation related** -

In relation to the implementation, currently there is issue with the following lane markings at two places. At the end of one of the white patches the lane mask is going little outside toward left. At another place the lane mask is going out for one or two frames when the black vehicle on right takes over. This may be because of white or bright color of car wheel getting wrongly detected as lane marking.
Another this is related to the lane curvature, it is little hard to understand if the detected lane curvature is correct or not.

For lane detection in very bright and dark places, HDR images may be more useful along with the Color and gradient transformations. The HDR takes images with multiple exposures and comes up with and image of balanced exposure, so it can be better for lane detection in very bright and dark areas.

Below is an example of test image with normal exposure and under exposure. We can easily see the difference in visibility of lane markings. With HDR it can be further improved.



### b) **Workspace related** –

I faced problems related to *Jupyter* workspace. The workspace do not support files of large size that well. Once I executed the code and had all the images as output then it was not possible to save the file. Every time I need to restart the kernel and clear all outputs before saving. Due to this I lost almost 15days and had to rewrite the code as initially reason for this behavior was unknown to me.