**Advanced Lane Finding Project**

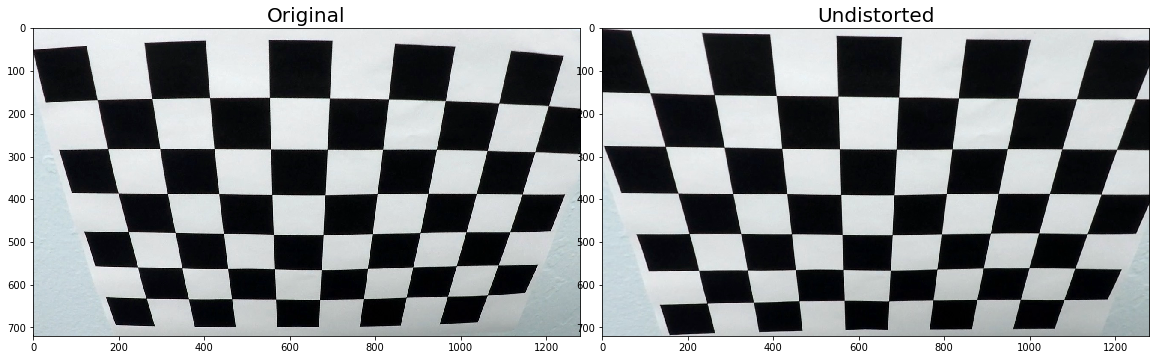
The goals / steps of this project are the following:

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 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.

Results

Undistorted Chess board



Undistorted test image



Transformed Road Image



Binary Image (Transformed) output images



Fit visual



Output Videos

[Project Video]: ./project\_video\_output.mp4

[Challenge Video]: ./challenge\_video\_output.mp4

[Harder challenge Video]: ./harder\_challenge\_video\_output.mp4

1.**Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.**

We used ‘findChessboardCorners’ and ‘calibrateCamera’ from OpenCV for this purpose

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 `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` 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 `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:

**2. Pipeline (single images)**

Calibrate the camera using chess board images 🡪 Use the calibration coefficients to undistort the road image 🡪 Do a perspective transform 🡪 Apply gradient threshold (x direction threshold, gradient, direction threshold) 🡪 Apply colour threshold [HLS and LAB {choose the thresholds based on the image we have}] 🡪 Get histogram and the peaks of histogram will be starting point for our line 🡪 Use sliding window and fit a polynomial 🡪 Try to find lanes using the prior polynomial fit 🡪 Check if the prior polynomial still fits if not start with sliding window again

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

Code is in cell 6 and 7 in the jupyter notebook

I created this function

*def unwarp(img, src, dst):*

*h,w = img.shape[:2]*

*# use cv2.getPerspectiveTransform() to get M, the transform matrix, and Minv, the inverse*

*M = cv2.getPerspectiveTransform(src, dst)*

*Minv = cv2.getPerspectiveTransform(dst, src)*

*# use cv2.warpPerspective() to warp your image to a top-down view*

*warped = cv2.warpPerspective(img, M, (w,h), flags=cv2.INTER\_LINEAR)*

*return warped, M, Minv*

*h,w = example\_img\_undistort.shape[:2]*

Source points

*src = np.float32([(575,450),(720,450),(258,682),(1070,682)])*

Destination points

*dst = np.float32([(450,0),(w-450,0),(450,h),(w-450,h)])*

I have actually visualized the output by manually inserting points (commented in cell 7). There is a scope for improvement in identifying the source and destination points



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

We used two functions here ‘sliding\_window\_polyfit’ and ‘polyfit\_using\_prev\_fit’

We identify the lane lines and tried to fit it in a second order polynomial. The starting point was identified using peaks of histogram. The starting point is identified by diving the width into two, but there was an unnecessary line.