Advanced Lane Finding

The goals / steps of this project are the following:

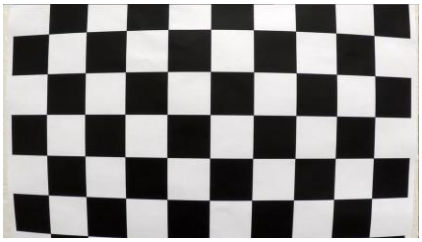
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

Part 1: Compute the camera calibration matrix and distortion coefficients given a set of chessboard images, apply distortion correction to raw images.

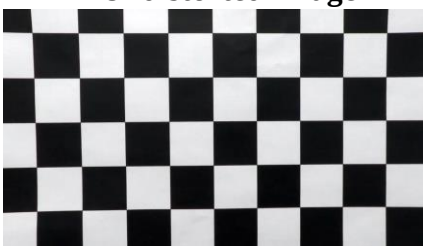
For this step, I created arrays to store 2d and 3d points – **imgpoints** and **objpoints**. Found corners for every images using **cv2.findChessboardCorners()**. After finding the corners, I calibrated the camera using the calibration matrix and distortion coefficients.

After the calibration, I used the calibration matrix and distortion coefficients for distortion correction – using **cv2.undistort()**

Original Image



Undistorted Image

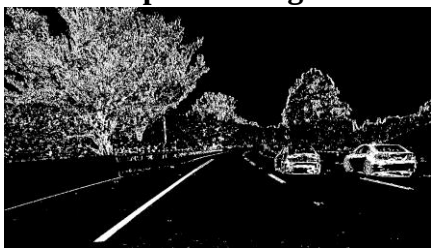


Part 2: Use color transforms, gradients, etc., to create a thresholded binary image.

To detect the lanes, I used the same algorithm which was used in the lecture. Converting the image to HLS form, using l_channel for gradient thresholding and s_channel for color thresholding. Using the appropriate thresholding constraints, I combined both the binary thresholds.

Original Image

Pipeline Image



Part 3: Apply a perspective transform to rectify binary image ("birds-eye view").

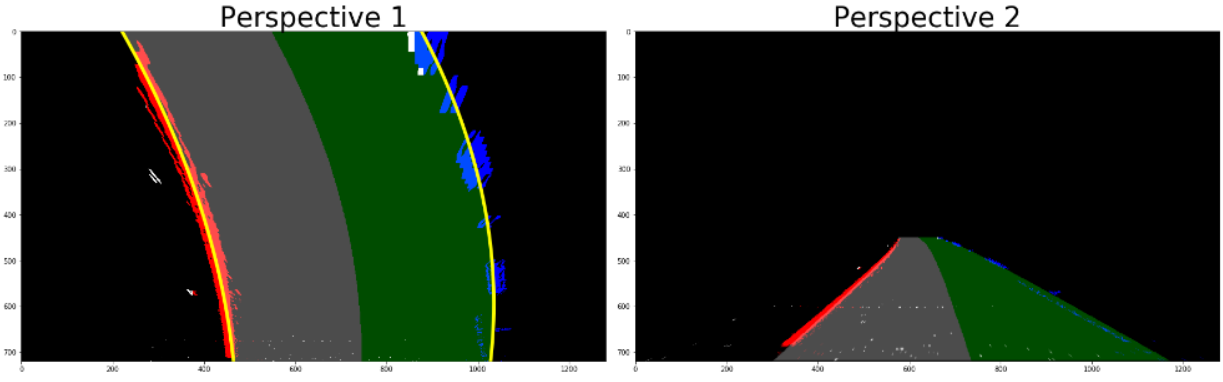
To find the region of interest, I used the same vertices which I used in the previous lane finding project.

For the perspective transform, the source vertices were similar to the vertices used for the region of interest (with some tweaking) but for the destination vertices, I used brute force.

Part 4: Detect lane pixels and fit to find the lane boundary.

The lane markings were found using **Histogram Peaks** method. Used **Sliding Window** method to find the polynomial corresponding to each lane and extended the **Search from Prior** method to highlight the whole lane.

After detecting the lane boundaries, I reversed the perspective transform by interchanging the source and destination vertices.



Part 5: Determine the curvature of the lane and vehicle position with respect to the center.

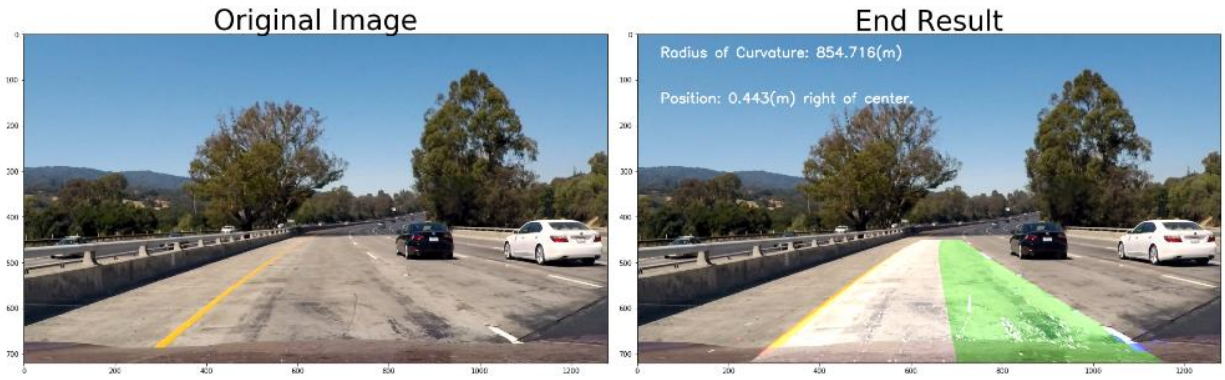
To measure the radius of curvature of the lane, I used the method that was in the lecture. For the vehicle position, I find calculated the center of the lane – to be specific, the x coordinate of the lane center.

Used the center of the image as a reference and calculated the deviation from the image center i.e subtracted the coordinate corresponding to the image center and coordinate corresponding to lane center.

If the deviation comes to be positive, the vehicle is to the left of the lane center and of the deviation is negative, the vehicle is right.

Part 6: 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.

After detecting the lanes and estimating the curvature radius and vehicle position, I used **cv2.addWeighted()** to wrap the detected lane boundaries back onto the original image.



I implemented the pipeline for the video. The project_video.mp4, the lane boundaries are detected with few deviations. But for the challenge videos, the pipeline needs to be modified more.

Apart from a few deviations in the video, the lane boundaries are detected perfectly.

Improvements:

- A better thresholding method can be implemented.
- Pipeline for video can be made more robust.