ND013 - Self-Driving Car

# 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.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Use color transforms, gradients, etc., to create a thresholded binary image.
- 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.

## Source Code

The main executable source code is ‘advancedlanelines.py’. The process flow is generally separated into 3 parts. First part is to extract the camera calibration coefficients using the images with checkered box pattern. Second part is to threshold the image and find the lane lines from the test images. Third part is to apply the laneline detection to a video.

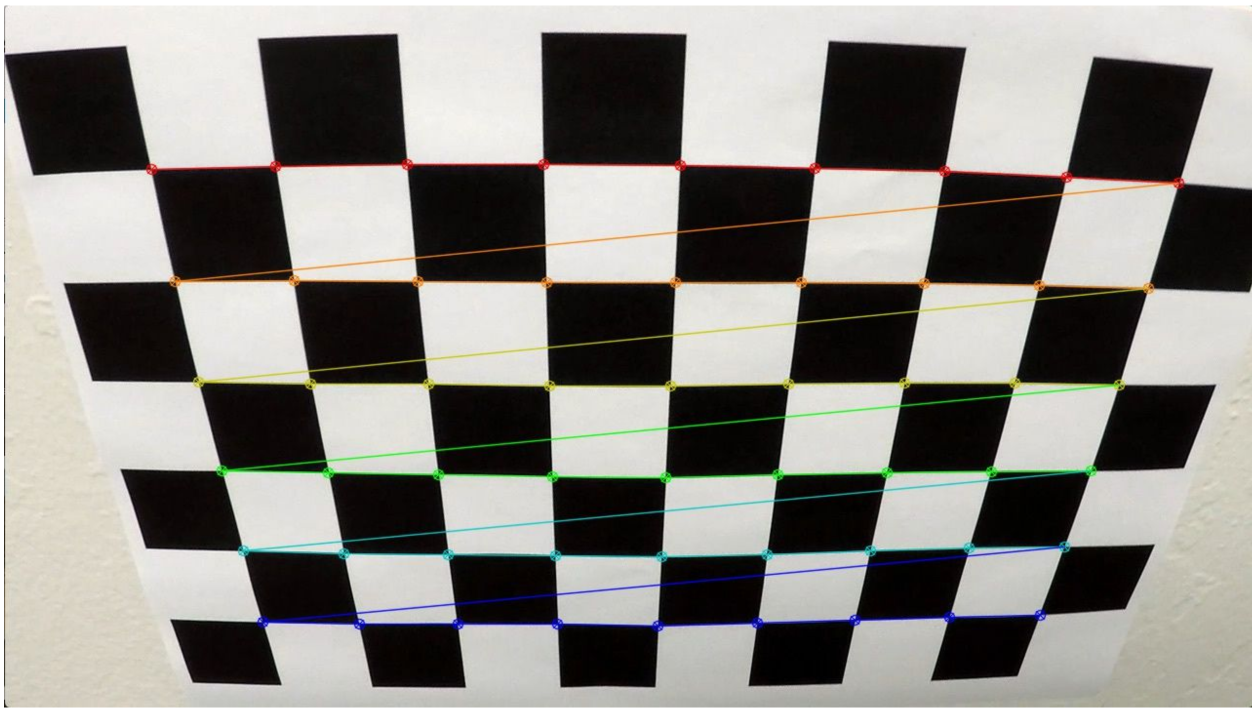
## Camera Calibration

The camera calibration code is under the function ‘calibrate_camera’. It takes all the images from the ‘camera_cal’ folder. ‘cv2.findChessboardCorners’ is then used to find the

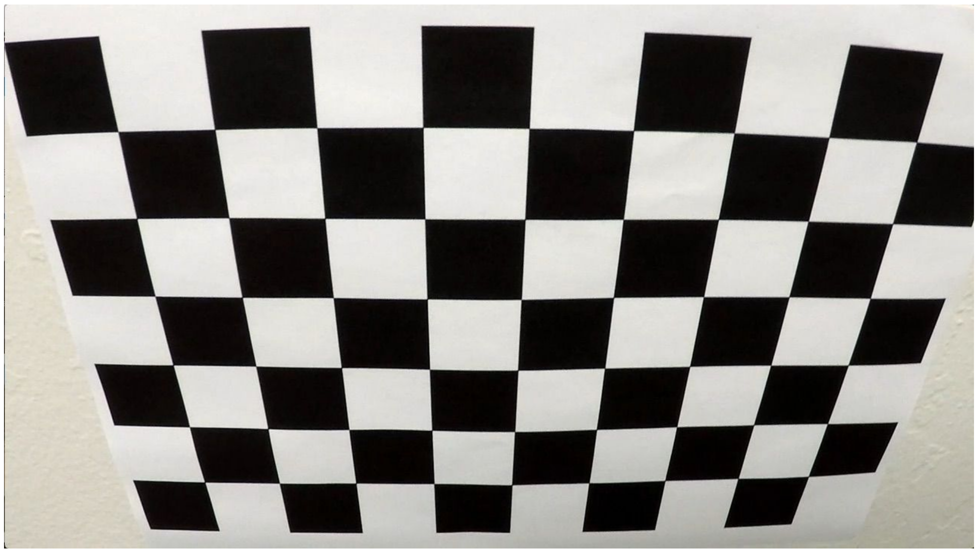
9,6 checkered box pattern from the images. If found, the corners will then be stored into an array and then refined using ‘cv2.cornerSubPix’. For visualization purpose, I use ‘cv2.drawChessboardCorners’ to draw the detected corner on top of the calibration images. The camera calibration coefficients can be found by using the function ‘cv2.calibrateCamera’ and the corners detected earlier. The coefficients are stored into a pickle file for future usage so that I do not need to recalibrate all over again if I rerun the whole process.

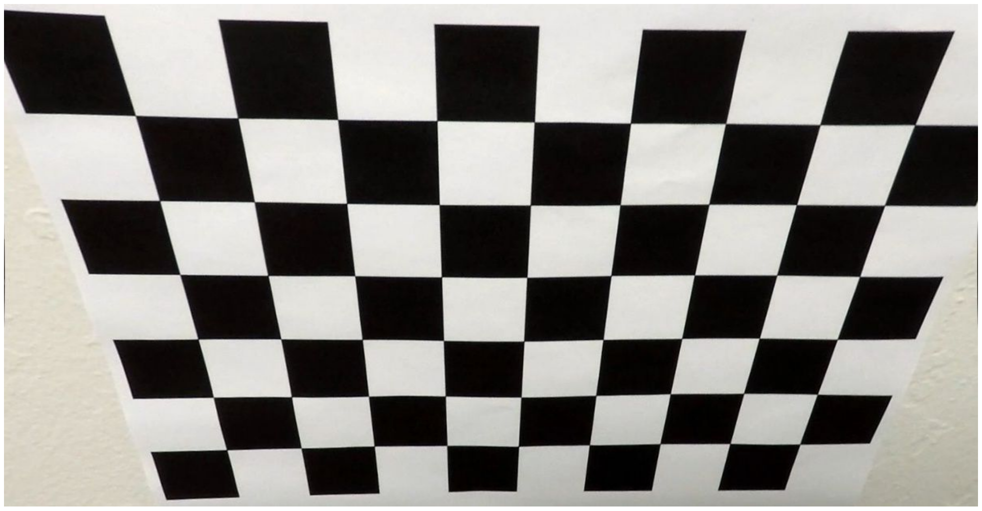
In order to verify the process, every camera calibration images and test images go through one time undistortion process for visual inspection.

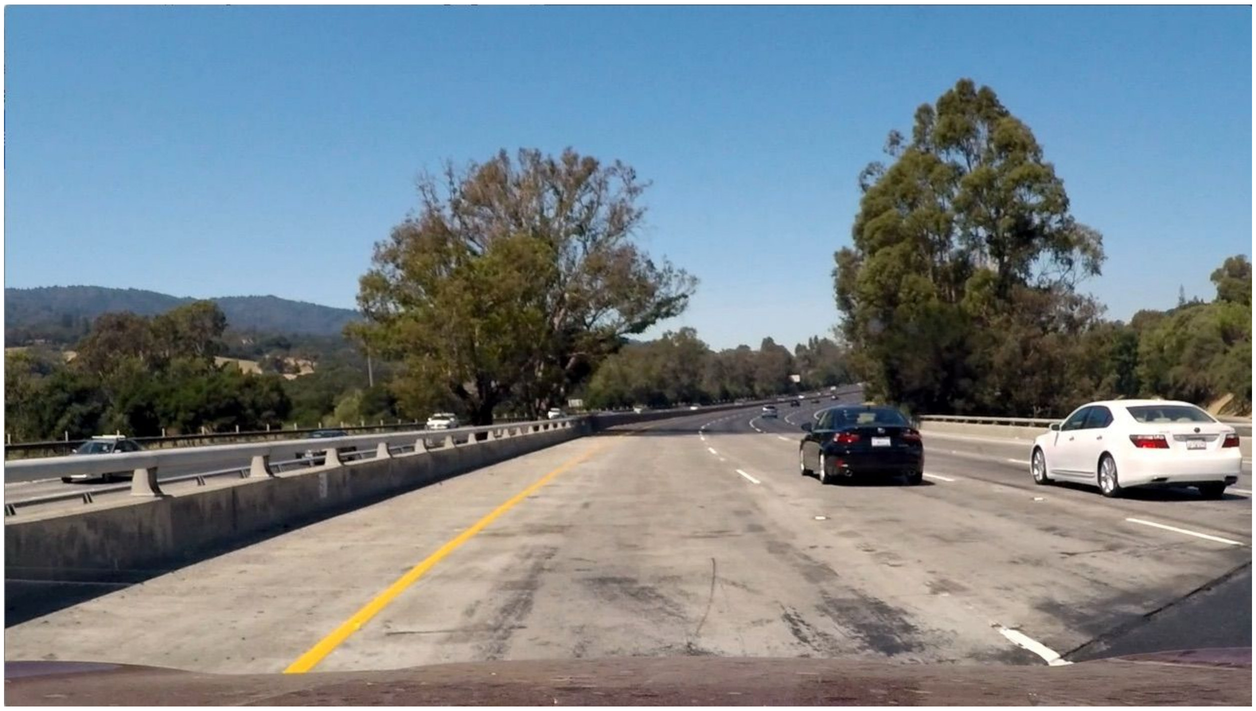
## An example of calibration results



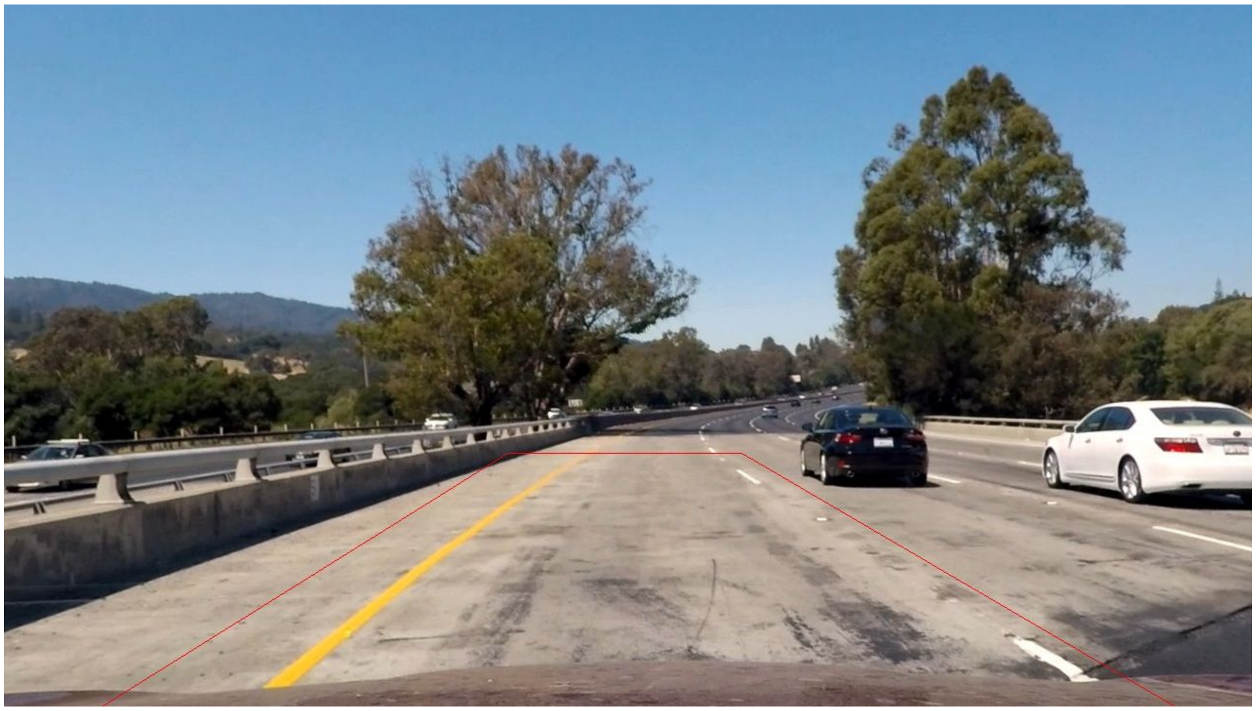
Finding the corners in calibration2.jpg


calibration2.jpg before calibration


calibration2.jpg after calibration



test1.jpg before calibration



test2.jpg after calibration

# Finding Lane Lines

## Perspective Transformation

After applying undistortion based on the camera coefficients, a perspective transformation is applied to get the bird's-eye view of the road. The transformation area can be seen from the previous image as drawn with red line. It is done at the function 'transform'.

## Thresholding

Using the bird's-eye view, I created 2 types of thresholding to cater for different situations.

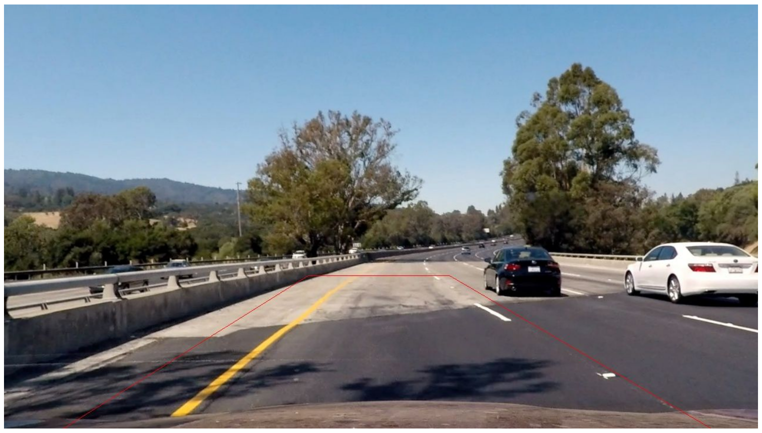
### Type 1

The type 1 thresholding can be found at function 'threshold_type1'. Generally, it combines the mask of 3 thresholds: magnitude of sobel with threshold (15 - 130), yellow mask using HLS colorspace (18,10,10 - 80,240,240), and white mask using HLS colorspace (S> 100 and h < 90). This threshold performs well in detecting the left line especially in yellow.

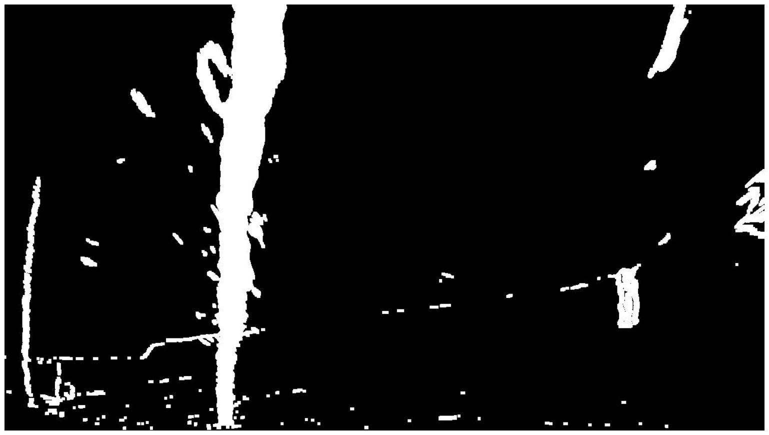
### Type 2

The type 2 thresholding can be found at function 'threshold_type2'. It combines Sobel in X-axis threshold with (50 - 150), magnitude of sobel with threshold (50 - 255), and directional sobel with threshold (0.75 - 1.15). They are combined with color threshold on S in HLS colorspace (170 - 255) and color threshold on R in BGR colorspace (200 - 255).

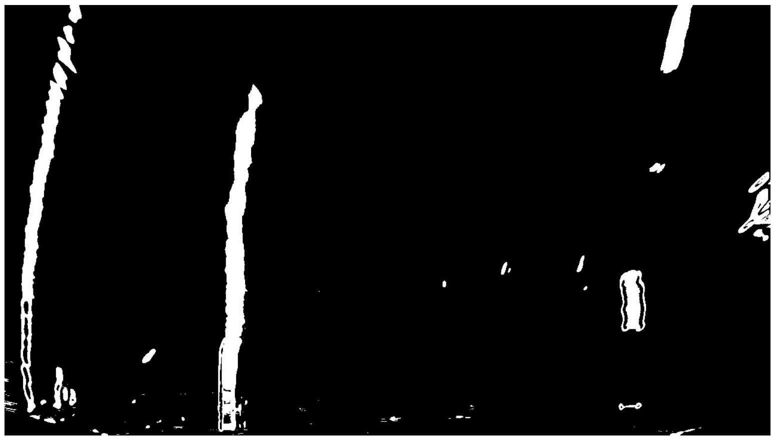
### An example of threshold results



test4.jpg



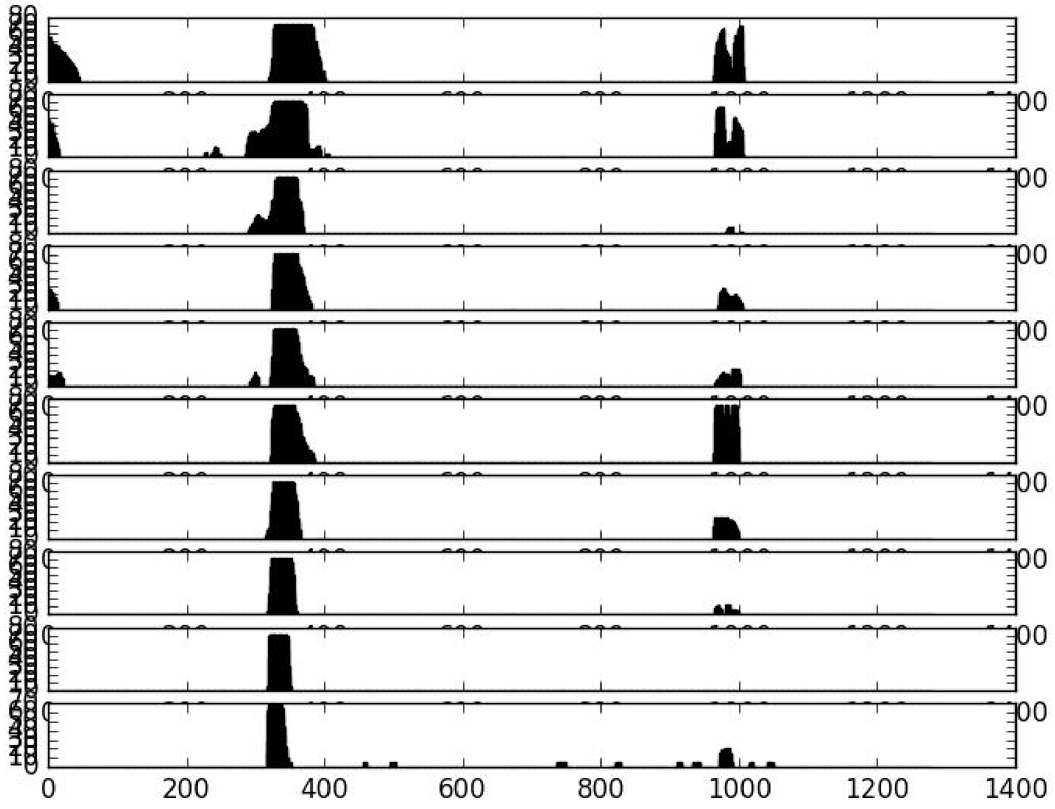
Type 1 threshold on test4.jpg



Type 2 threshold on test4.jpg

## Finding Centroids

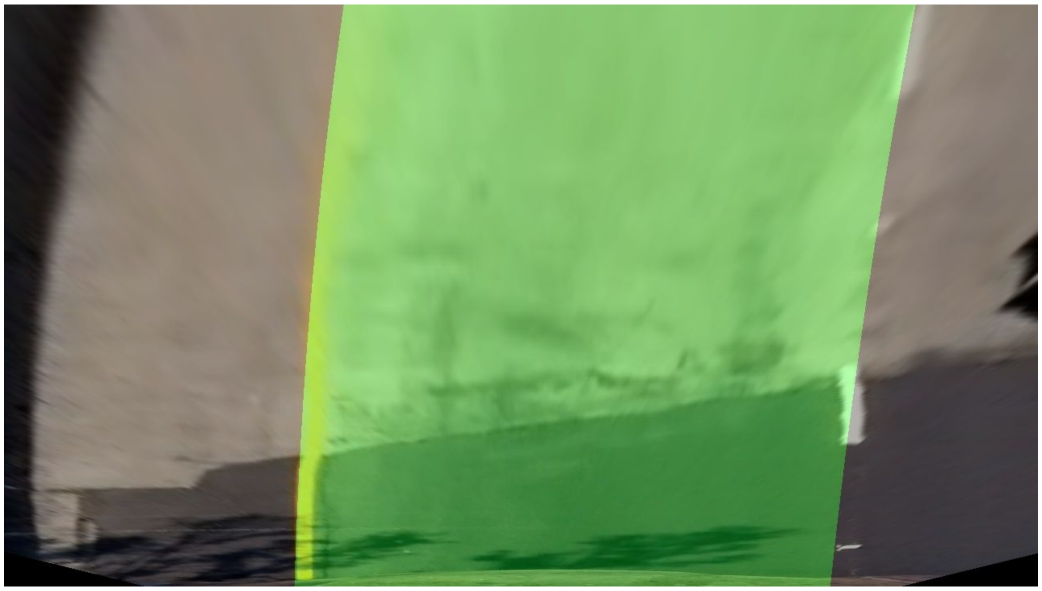
In order to look for centroids, each threshold image is separated into 10 horizontal regions of left and right (Total 20 regions). The peaks are identified from every regions through histogram.



## Curve Fitting

I have 2 sets of histogram from Type 1 and Type 2 thresholding. They are fitted with second order polynomial. By comparing the error rate (residuals) and number of centroids, I am able to get the best fit (choosing from Type 1 and Type 2) for every image.

### Results of Curve Fitting



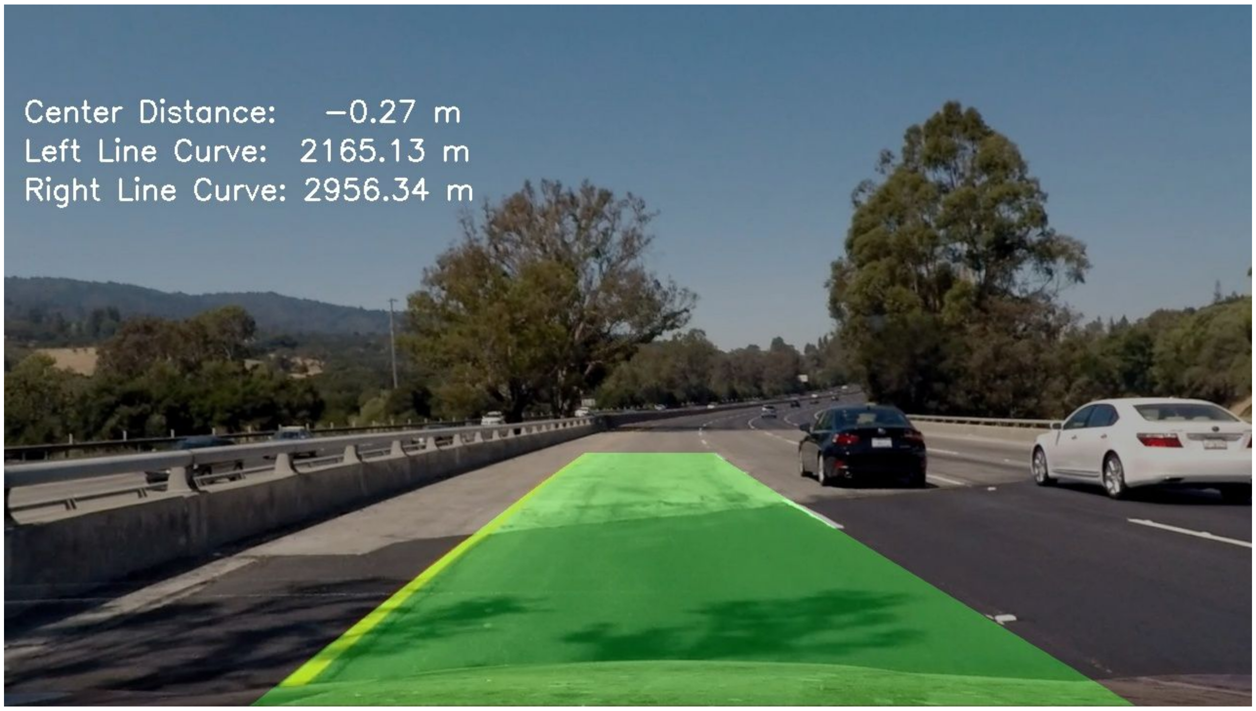
Curve is fitted on test4.jpg

## Curvature of the lane

To determine the curvature of the lane, I follows the code from Udacity. The process can be found in ‘calculate_radius’.

## Final Result

The mask of the fitted curve is then transformed inversely and plotted on top of the calibrated image. The details of the lane curvatures and center distance is written on top of the image too.



Final result of test4.jpg

## Video

Using Moviepy, the whole process is reiterated on top of the image frames extracted using fl_image function. The pipeline can be found at function 'video_pipeline'.

The output of the result is uploaded at youtube:

https://www.youtube.com/watch?v=k94FUQPfGaQ&feature=youtu.be

## Discussion

If given more time, a better algorithm like sliding window can be use to better determine the centroids to discard the excessive noise. More threshold tuning can be done to better capture the lane lines in various condition like snow and rain. The method is good only for car staying in flat road. If the car is overtaking, the whole lane line detection will fail. A better algorithm to detect drivable area need to be determined by calculating other objects like car and human. It might fail for other country too because I depended very much on the yellow color for left line detection whereas many other countries only use white paint for the laneline.