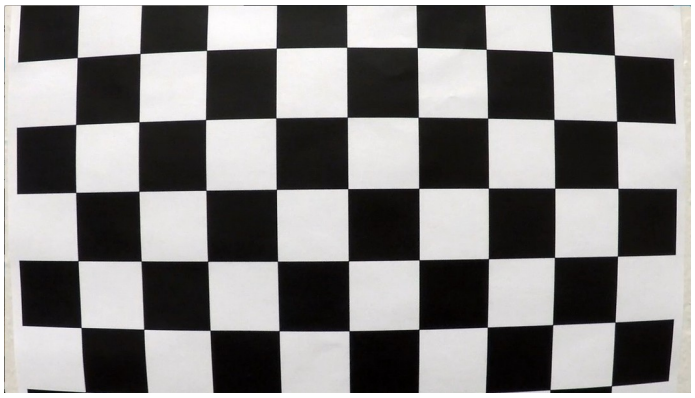
Advanced Lane Finding Project **Rahul Tarkunde KPIT Scholar Batch**

The goals / steps of this project are the following:

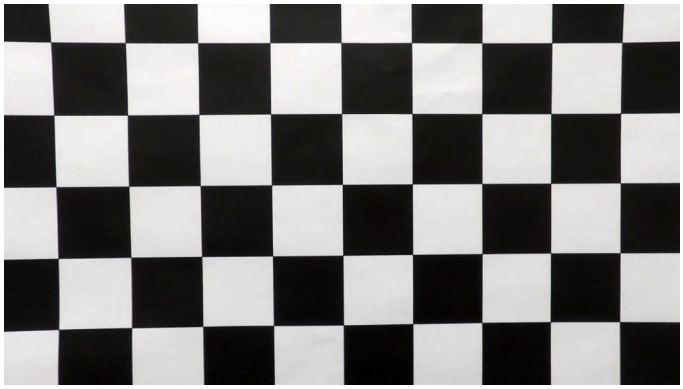
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").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

Camera Calibration

The code for this step is contained in the ***caliberate_camera*** file Project.py (in lines 91 to 128).
The code calibrates the 9 x 6 corner chessboard images to get the distortation coeeficients.

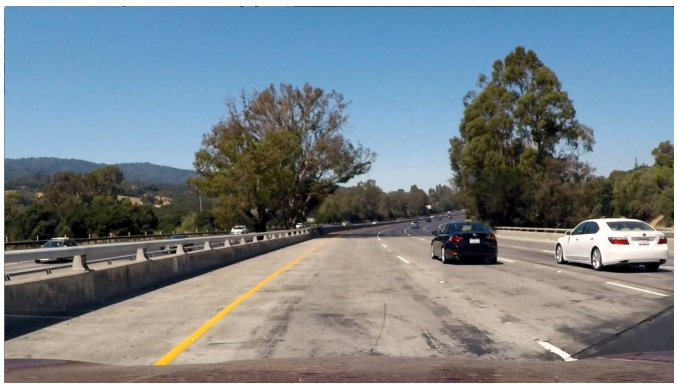


Uncaliberated image
(./camera_cal/calibration1.jpg)

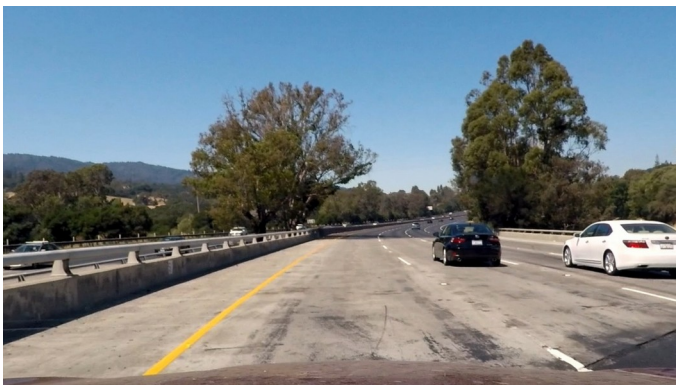


Caliberated image
(./output_images/camera_cal/calibration1.jpg)

These coeffiecents are then applied to the sample images using the function ***undistort_image*** (Line 130) Here is the output on actual images



./test_images/test1.jpg

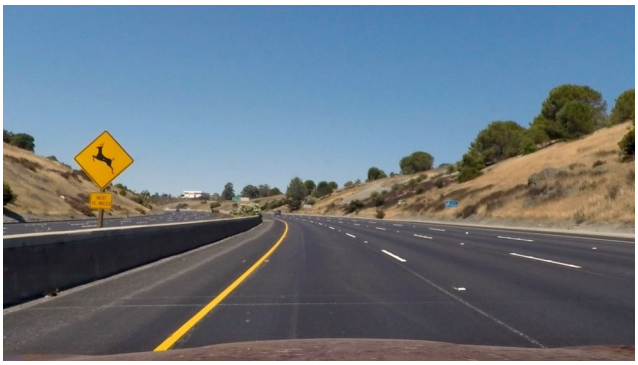


./output_images/test_images/test1.jpg

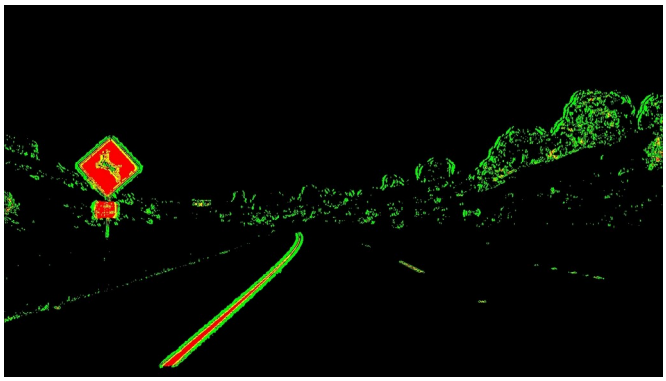
Binary Images:

I used some calibration techniques to produce binary images
The code is located at ***create_thresholded_image*** at line 136. The code

- converts image to HLS color transform
- uses sobel on the saturation channel to apply derivative on x direction
- and converts it to binary and stacks to create a three channel image.



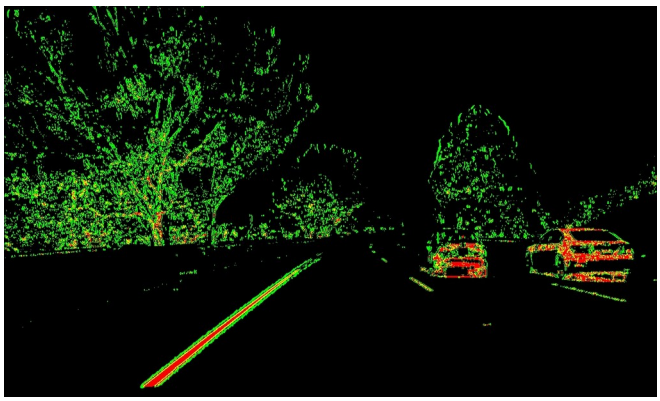
./test_images/test2.jpg



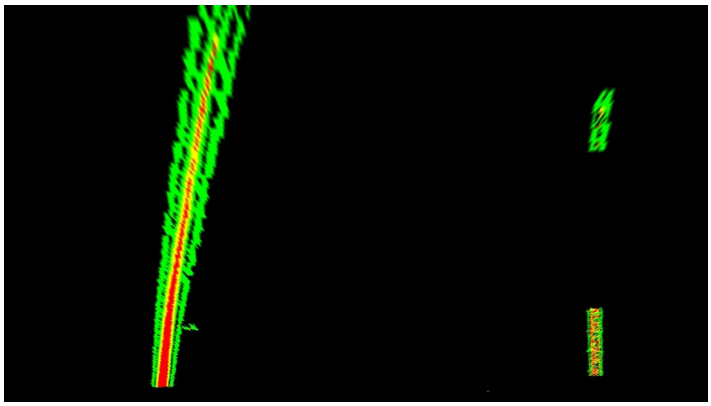
./output_images/test_images/threshold.jpg

Perspective Transform:

The perspective transform is acheived using the function ***apply_perspective*** atline 160. The perspective points are chosen by trial and error. Taking a straight line image, opening it in inkscape and checking the line end coordinates. The top level coordinates are then chosen by using the x coordinates of the bottom coordinates and y is zero or the top of the image. Minor adjustements are done using x_correct and y_offset. The function also returns a map matrix which is used to upwrap the line image later on.



Before perspective transform
./output_images/before_un_wrapped_perspective.jpg



After perspective transform top
./output_images/after_perspective.jpg

Lane line pixels and fitting of polynomial:

The laneline pixels are identified using function ***lane_line_pixels*** line 292. It uses the line class which has stored values to do sanity check. For the first image the function uses histogram to identify the lane pixels while for the subsequent images it uses search around the earlier detected polynomial values with margin. The function also does sanity checks and then either updates the class values or set the current values from the earlier image values.

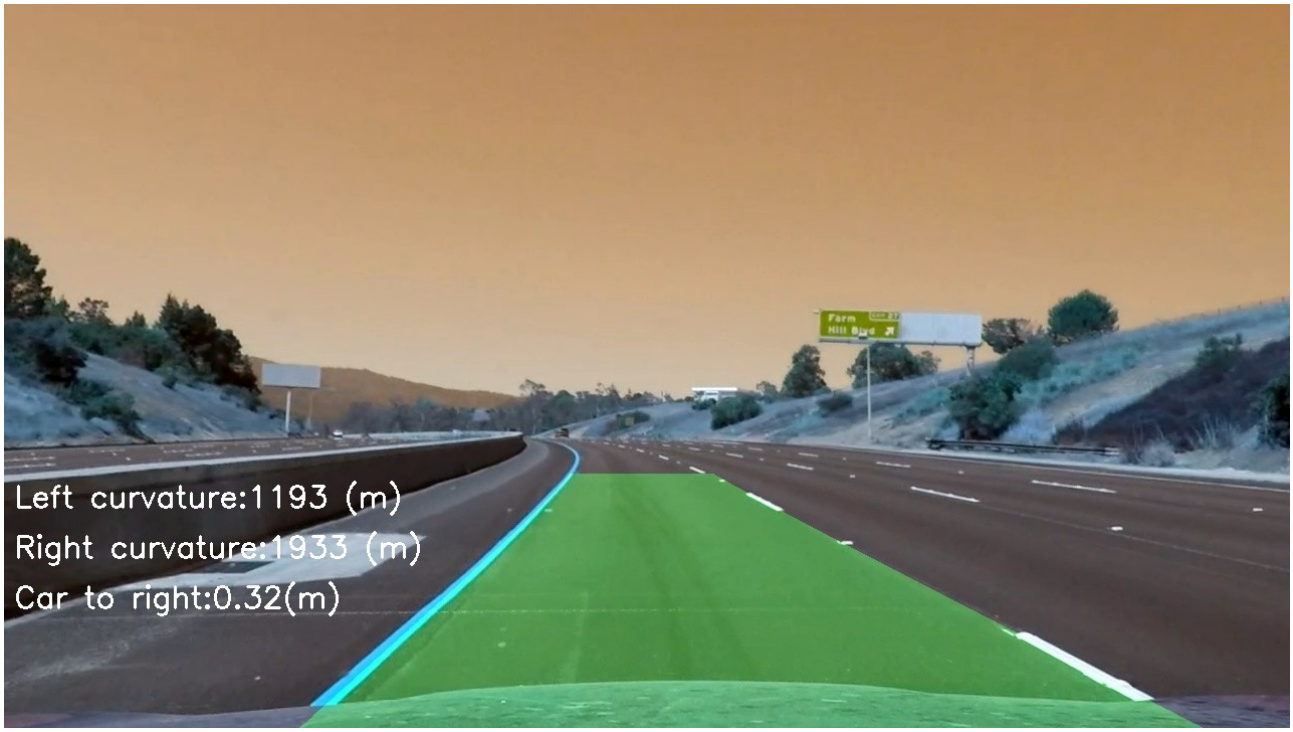
Curvature calculation:

Curvature caluclautions are done using function ***measure_curvature_real*** at line 385 which also calls ***polyfit_data*** line 372. These work in meters instead of pixels. They are calculated based on second order equation whose coefficents are received from numpy polyfit function. The values are stored in the line class instance which are used for sanity check for next frame and printing them on the image.

Perspecitve unwrapping:

Final step in image pipeline is processed by function ***unWrapPerspective*** at line 360. The function takes the binary warped image and the origianal image. The warped image is reverse warped to original perspective and then weight added to the original image. The image is then ready for writing text on and putting in video stream.

After all the processing the image is written



./output_images/final_image_20.jpg
(The image coloration due opening of image with COLOR_RGB2GRAY. The final video output is rendered correctly)

Pipeline (video)

The video is located at **./project_video_out.mp4**. The pipeline functions are called from main function ***process_image*** line 440 which is called for processing of each video frame. The pipeline consists of seven steps as described above. Some additional functions are used line cv2.invert to reverse calculate the matrix values and a ***text_on_image*** *is used to write the curvature and lane center values*

Discussion

This was very challenging project. There was issue in maintaining the historic value with the help of classes and did not knew where would a good place be to store and read the value. However testing the code after every few pipeline elements solved it. During final issue there were challenges with the curvature and the lane drawing getting jumped. This was solved by observing reasonable curvature and lane width values and taking a action when it was detected by copying the previous value.

The program will have issues at road crossings and pedastrian crossings. Challenge videos too.