A

Adavanced lane finding Project

Faisal Waris

Udacity

Self-Driving Nano Degree

# Introduction

The goal of this project is to perform several image processing steps to identify lanes lines and measure the lane curvature.

The basic steps are as follows:

1. Calibrate camera to remove distortion
2. Threshold images to surface the salient features for lane detection
3. Apply perspective transform to obtain a bird’s-eye-view of the road for lane detection and curvature estimation
4. Identify lane lines and measure curvature
5. Project the detected lane boundaries onto the original image for validation and viewing

The steps outline above are explained in detail in the subsequent sections of this report. The final section is the discussion of the results.

The links to source code and other artifacts relevant to this report are given below:

|  |  |
| --- | --- |
| GitHub repository: | <https://github.com/fwaris/CarND-Advanced-Lane-Lines> |
| Processed video file: | <https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/output_videos/project_video_lanes.mp4> |
| Sample output images folder: | <https://github.com/fwaris/CarND-Advanced-Lane-Lines/tree/master/output_images> |

The pipeline processing code is organized into the following modules (files):

|  |  |  |
| --- | --- | --- |
| Module / file name | Function | Link to source in repo |
| BaseTypes.fs | Contains a structure to encapsulate hyperparameters such as thresholds window sizes and other constants for automated tuning of hyperparameters if need be. This structure is passed to most functions in the code and each function uses the hyperparameters from the structure as needed | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/AdvancedLaneMarking/BaseTypes.fs) |
| CalibrateCamera.fs | Calibrate camera | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/AdvancedLaneMarking/CalibrateCamera.fs) |
| ImageProc.fs | Processing single images for thresholding, perspective transform, etc. |  |
| LaneFind.fs | Find lanes by performing processing over pixels of the ‘thresholded’ and transformed images | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/AdvancedLaneMarking/ImageProc.fs) |
| LineFitting.fs | Fit hyperbola curves over the identified lane line pixels | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/AdvancedLaneMarking/LineFitting.fs) |
| VideoProcessing.fs | Contains the processing that puts everything together to process a video file | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/master/AdvancedLaneMarking/VideoProcessing.fs) |

# Camera calibration

The camera calibration was performed by using the give set of calibration images in the project repo (calibration 1-20). I followed the approach given in the [OpenCV tutorial](http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html). Essentially the provided set of calibration images were used to collect ‘object points’ and matching ‘image points’ from OpenCV FindChessboardCorners function. Figure 1 is an example of visualizing the found chessboard corners.



Figure : Chessboard corners visualized

The object points are a static collection of points roughly corresponding to the number of board squares ([l[ink to code](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/a24d7ae6751ed0b81f401c1746064358d6717d9c/AdvancedLaneMarking/CalibrateCamera.fs#L12)](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/361198a2e5f5827da367329e754c23f448ae93e0/AdvancedLaneMarking/CalibrateCamera.fs#L11) to generate static points). The code to generate the collection of image points from the calibration images is given [here](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/361198a2e5f5827da367329e754c23f448ae93e0/AdvancedLaneMarking/CalibrateCamera.fs#L25). The collection of points was used to [[[derive a 3x3 camera matrix to remove distortion, using the OpenCV CalibrateCamera function](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/361198a2e5f5827da367329e754c23f448ae93e0/AdvancedLaneMarking/CalibrateCamera.fs#L49)](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/a24d7ae6751ed0b81f401c1746064358d6717d9c/AdvancedLaneMarking/CalibrateCamera.fs#L50)](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/a24d7ae6751ed0b81f401c1746064358d6717d9c/AdvancedLaneMarking/CalibrateCamera.fs#L50).

The derived camera matrix was then used [to undistort the original camera image](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/a24d7ae6751ed0b81f401c1746064358d6717d9c/AdvancedLaneMarking/CalibrateCamera.fs#L67) as show in Figure 2.

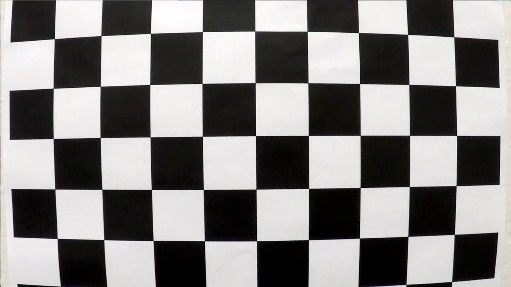
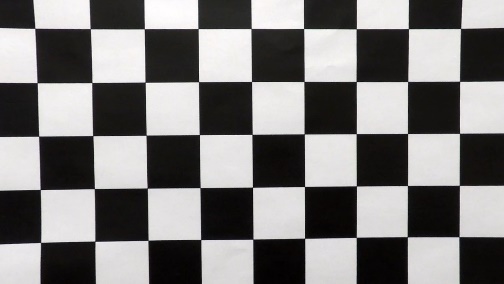
 

Figure : Calibration original and undistorted image

Initially I had an issue with the OpenCV API call CalibrateCamera due to incorrect length of the distortion coefficients array (I had passed in array of length 8 instead of the required 5). Once I corrected the API the camera calibration worked fine.

# Color and Gradient Transforms

The input image was processed with 3 types of thresholds:

|  |  |  |
| --- | --- | --- |
| Threshold | Comments | Code Link |
| Yellow color isolation | Used HSV color format to isolate yellow color in images such as those for solid lane lines  OpenCV methods CvtColor and InRange were utilized | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L18) |
| White color isolation | Same as above for white color. The yellow and white color masks were bitwise ORed using OpenCV BitwiseOr method | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L18) |
| Gradient threshold | The OpenCV Sobel function was used to find gradients in the x direction and then a mask was created that contained pixels where gradient is in a specified range | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L32) |
| Combination | The results of the yellow and white color masks was combined with the gradient mask using OpenCV BitwiseOr function | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L40) |

Figure : Original and ‘thresholded’ images

An example of the orginal and thresholded version of the same images are shown in Figure 3.

# Perspective Transform

To create a birds-eye-view perspective transform, 4 points where selected on the camera image and then their transformed points where determined using human judgment.

The source and destination transformation points are given below:

Source Points List = [691,450; 1150,720; 250,720; 597,450]

Destination Points List = [1150,0; 1150,720; 250,720; 250,0]

A [perspective transform matrix was created](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L11) using OpenCV GetPerspectiveTransform function.

The original and perspective-transformed images are show in Figure 4. The [transformed image was created with the OpenCV WrapPerspective function](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/ImageProc.fs#L55).

Figure : Original and perspective transformed image

For the purpose of performing lane detection however, the perspective tranform was applied to the thresholded image rather than the original image. Figure 5 is an example of such processing.



Figure : Perspective transform applied to thresholded image

# Lane Detection

The input for detecting lane pixels for line fitting is the thresholded and warped image as show in Figure 5. The lane detection steps and code links are described in the following table:

|  |  |  |
| --- | --- | --- |
| Step | Description | Link to source |
| Find left and right lane start points | Use a frequency count of pixels in each column along with a sliding window to search for left and right lanes in the two vertical halves of the image, respectively | [link1](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LaneFind.fs#L8) and [link2](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LaneFind.fs#L21) |
| Find left and right lane pixels | The sliding rectangle (window) search method was used to locate the pixels for the left and right lane pixels | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/VideoProcessing.fs#L32) |
| Line fitting | A x and y dimensions of the pixel points are swapped fit parabola equations that fit will to the lane curvature.  The [ILNumerics FitPolynomial](https://numerics.mathdotnet.com/api/MathNet.Numerics/Fit.htm#Polynomial) method was used to find the equation of the hyperbola | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LineFitting.fs#L18) |
| Lane marking | Using the two left and right curve equations, the lane section was drawn on an empty Mat using OpenCV polyfill function. This Mat was inverse perspective transformed and merged with the original image using several OpenCV functions.  Figure 6 shows the result of a combination of processes to illustrate the identification of lanes on a warped and thresholded image.  Figure 7 is an example of a fully processed frame. | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/VideoProcessing.fs#L84) |

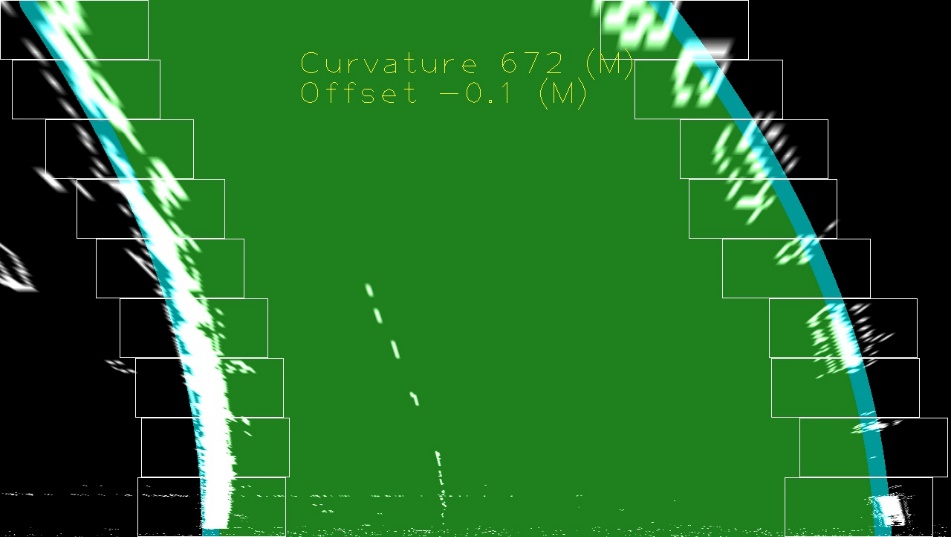


Figure : Lane curvature calculated and marked on warped image

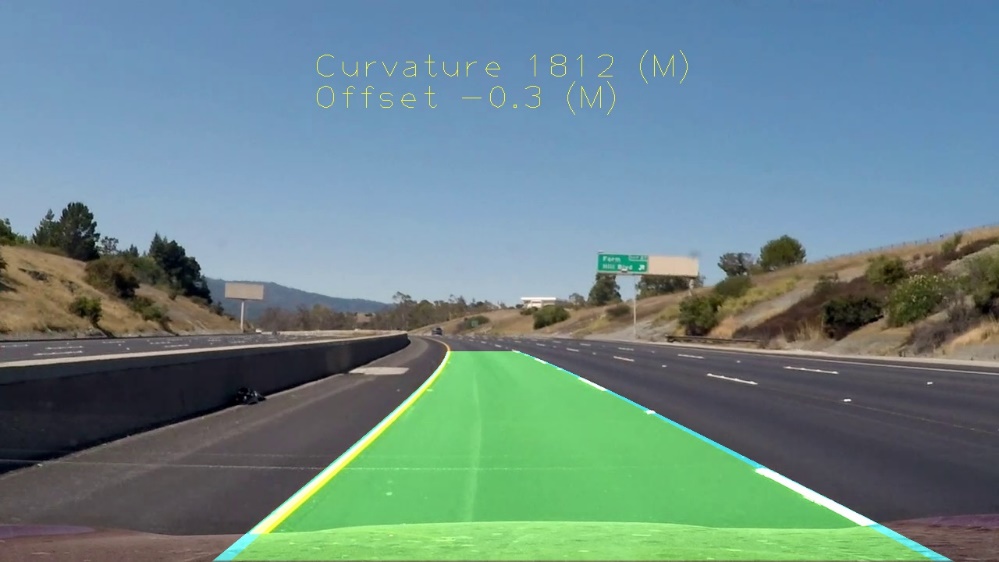


Figure : Example fully processed frame

# Curvature Measurement

To estimate the curvature of the road section in meters, the following processing was performed:

|  |  |  |
| --- | --- | --- |
| Step | Description | Link to source |
| Pixel to meters conversion | The estimated lane equations need to be re-estimated in world-space (in meters).  The pixel-to-meters conversion factors, given in the lecture were used for this processing | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LineFitting.fs#L25) |
| Re-estimate lanes in world space | The lane equations were used to first express line points in pixel space. These line points were transformed into world space using the conversion factors described earlier. Note the X dimension (which really is the swapped Y dimension from the camera frame) was flipped so that vehicle is located as X=0 (instead of X=720). The word-space points were used to re-estimate lane equations in meters. | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LineFitting.fs#L31) |
| Measure curvature | Once the lanes were estimated, the curvature at the bottom of the camera frame was measured using the formula given in the lecture | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/LineFitting.fs#L44) |
| Curvature smoothing | The left and right lane curvatures were averaged and smoothed over several frames before rendering as text on the output frame | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/VideoProcessing.fs#L61) |
| Position offset | The offset of the vehicle was calculated as the difference between the center of the frame and the midpoint of the two lanes from their respective start points. The start points where determined earlier during the start of frame processing. | [link](https://github.com/fwaris/CarND-Advanced-Lane-Lines/blob/39223f019fdafc50ae0a16b89e8feee38f3d2972/AdvancedLaneMarking/VideoProcessing.fs#L76) |

# Discussion

During the development of the code, several alternative image pre-processing steps where tried. For example, instead of ORing the color and gradient thresholds, bitwise AND was considered. It did not perform as well.

Not being a computer vision expert, this was a difficult project for me as it required a deeper exploration of the OpenCV API as compared to previous projects.

The code performs reasonably well on the project video but does not perform well on the challenge videos. I feel that greater mastery of the OpenCV API is required for me to effectively start tackling the harder videos.