Project 2 Advanced Land Finding Writeup

February 21, 2021

1 Project Goals

The goals / steps of this project are the following:

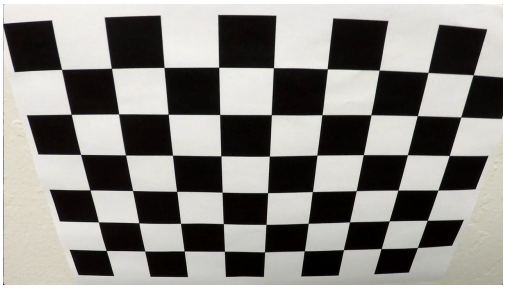
* Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
* Apply a distortion correction to raw images.
* Use color transforms, gradients, etc., to create a thresholded binary image.
* Apply a perspective transform to rectify binary image ("birds-eye view").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

2 Write up

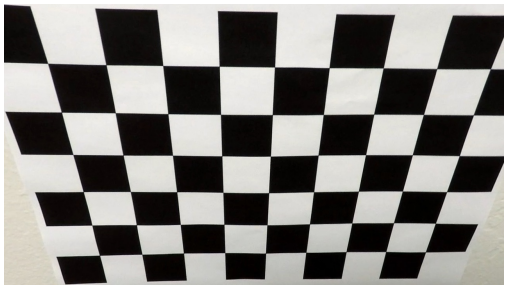
In this section I will go over the pipeline code in project2.py

2.1 Camera Calibration and Distortion Correction

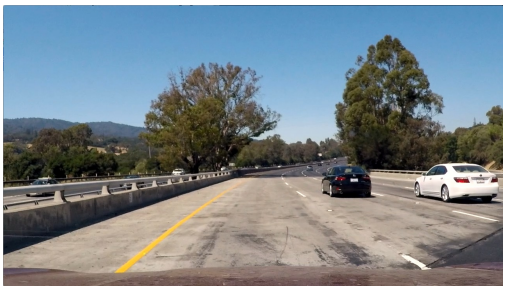
The first step is to calibrate the camera using the provided figures using the defined function *cameraCalibration* and *undistort*.
The first function defines object and find image points for camera calibration using $cv2.findChessboardCorners()$. The second function creates undistort images witg $cv2.calibrateCamera()$ and $cv2.undistort()$. An example of the figure before and after distortion correction is shown in Figure 1a , Figure 1b, Figure 2a and Figure 2b.



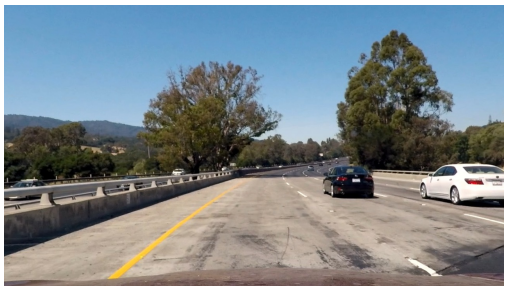
(a) Before Distortion Correction



(b) After Distortion Correction



(a) Before Distortion Correction



(b) After Distortion Correction

2.2 Create Color Thresholded Image

The function *thresholdImage* is used to create the binary threshold images. It use threshold in HLS S-Channel and Sobel X direction to create the thresholded binary image for line detection. Additionally a HLS L-Channel threshold is added to remove the dark tree shadow that also cross S-Channel threshold.

The final threshold values are $170 < HLS_S < 255$, $HLS_L >= 100$ and $40 < Sobel_x < 100$, an example of the output is shown in Figure 3.

Figure 3: Binary Thresholded Output of Test Image 5

2.3 Perspective Transform "Bird-Eye View"

The function *birdsEye* is used to warp the image to bird-eye view. The key parameters source and destination points are defined in the following Table 1. The values are verified by checking on a straight line as shown in Figure 4.

Table 1: Perspective Transform Parameters

Source Point	**Destination Point**
(592,450)	(240,0)
(688,450)	(1040,0)
(1120,720)	(1040,720)
(200,720)	(240,720)

Figure 4: Perspective Transform on Straight Road

2.4 Polynomial Fit

The function $search_around_poly$ is the main one to warp the image to bird-eye view. Before that, the function $fit_firstPoly$ uses sliding window method (as shown in Figure 5) to find the line to calculate the first polynomial and update the variable $self.left\_fit$ and $self.right\_fit$. After that, the line in the new frame is searched through with a smaller margin in an area based on the previous polynomial as Figure 6 shows. Then the calculated polynomial is shown in Figure 7.

2.5 Curvature and Vehicle Position Calculation

Curvature is calculated in the function *measure_curvature_real* through the following formula, where A and B are the coefficients in the polynomial and y is the location at the bottom of the image.

$$R_{curve} = \frac{(1 + (2Ay + B)^2)^{3/2}}{|2A|} \tag{1}$$

Then the vehicle position is calculate in function *measure_vehicle_position*, subtracted the average of the two polynomial points at the bottom of the image from the center point at image bottem.

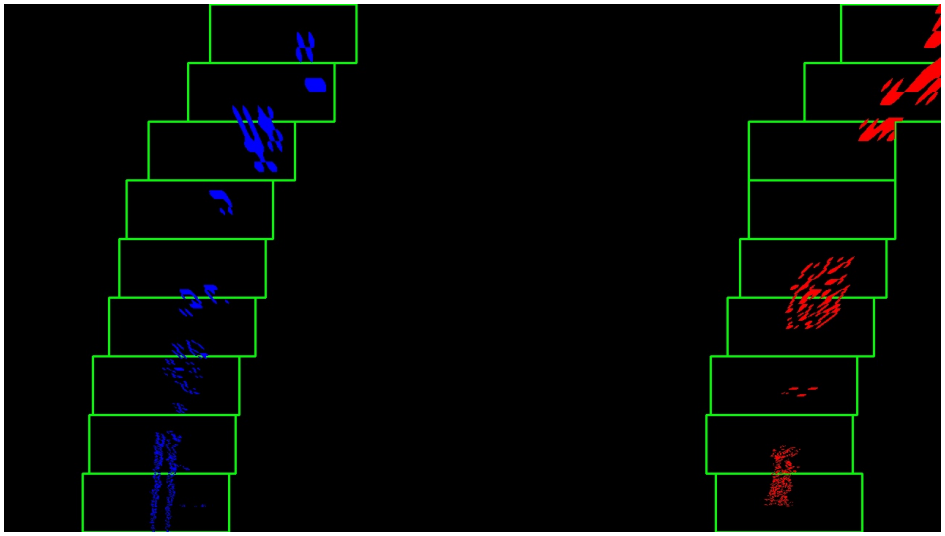


Figure 5: Sliding Windows

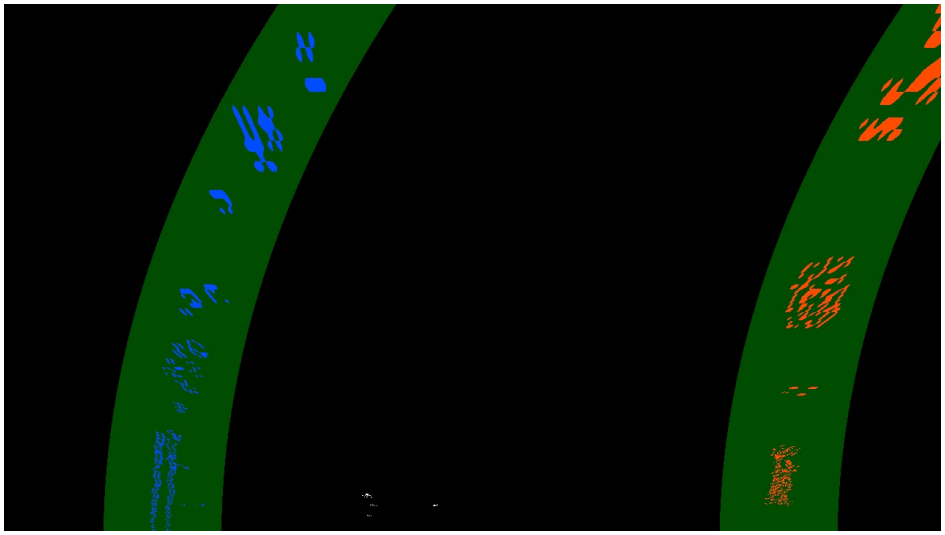


Figure 6: Polynomial Area

2.6 Inverse Perspective and Adding Comment

To finish the pipeline, A inverse perspective transformation is applied to the bird-eye view picture in function *inverse_perspective*. Then the calculated curvature and vehicle position is added the inverse perspective transformed image through function *add_word*. An example is shown in Figure 8. The output video is attached in submission

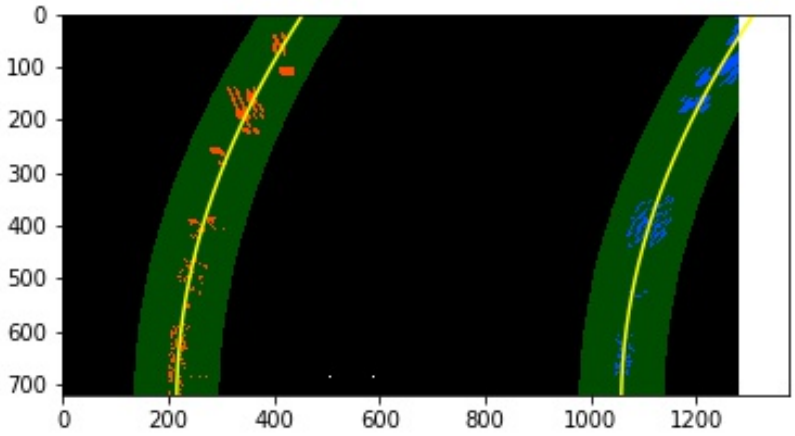


Figure 7: Polynomial Visualization

Figure 8: Example Output Frame

3 Discussion

The part that need most calibration effort is the to create the thresholded image. When the light condition changes or more noise on the road, the resulting thresholded picture can easily include lots of unwanted frame. To process the project video, the tree shadows create some difficulty so another restraint from the L channel in HLS space is added. But the same trick won't work on challenge video, a different set of calibration parameters in *thresholdImage* function or update the parameters dynamically based on the overall image

parameter such as lightness might work. But I could not spare more time on it. Also put all code in a overall object is another challenge for myself. I hope I have time to polish and update my code in future.