Self Driving Car Nanodegree Project 4

Michael Person

April 19, 2017

1 Camera Calibration

The camera calibration was done in the Calibration.ipynb file included in the submission. The chessboard that was used in order to perform the calibration was a 9x6 instead of the 8x6 one that was done in the course module and that was mainly the only difference in the code that we did in the module versus what I used. The same pipeline was used to form the meshgrid that the corners actually lie on, convert the image to grayscale and then use the opencv function findChessboardCorners and then check if the function did indeed find corners. Then from the found corners and known locations the correction matrices were returned from the opencv function calibrateCamera.

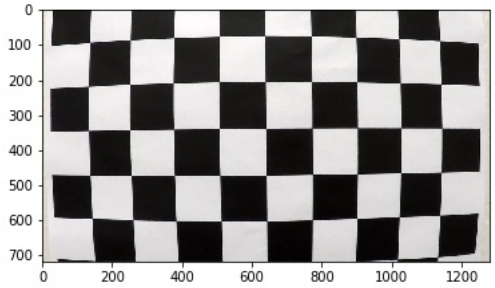

Figure 1: Original Image

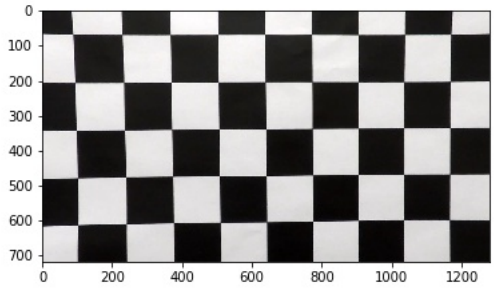

Figure 2: Undistorted Image

2 Pipeline

2.1 Distortion Correction

First the distortion matrices from 1 are used to undistort the image using the opencv function undistort. The below image is the output of this and it is difficult to tell what is different from the original image but it is most readily available from the green sign on the right side of the image. The sign appears slightly larger in the undistorted image than it does in the original.

Figure 3: Undistorted Road Scene

2.2 Combined Binary Image

I took the advice of one of the forum mentors and used a combination of the HLS colorspace and the standard image RGB colorspace. I threshold for yellow lines requiring higher than a certain value in the R and G channels and lower than that threshold in the B channel. Next I take the gradient of the S channel and then also a threshold of the L channel. Next I combine these 3 binary images by saying if the pixel is either a 1 in the L threshold or yellow threshold and in the gradient threshold then it is a lane. This method works well.

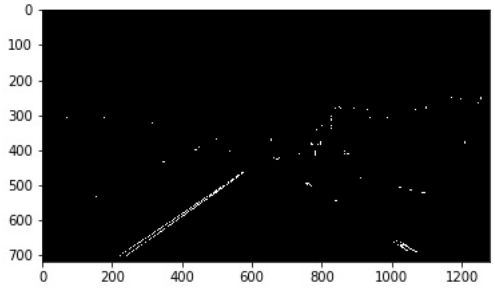

Figure 4: Final Binary Image

2.3 Perspective Transform

Next a perspective transform was performed on the image in order give us a birds eye view of the lanes in front of us. This was done using the opencv function warpPerspective and after hard coding the vertices in which the image was supposed to be transformed around. This both acted as a masking and also as a way in order to fit the proposed lanes much easier.

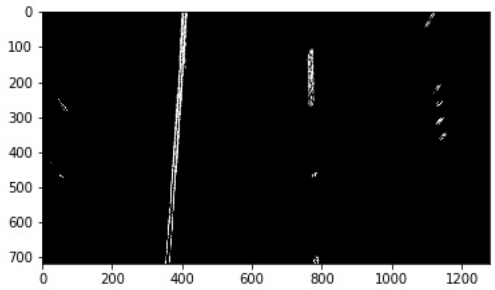

Figure 5: Warped Lane

2.4 Lane Pixel Identification

The start of the lane pixel identification, a horizontally sliding histogram search occurs binning all the found pixels in order to tally where the most common found pixel values are. At many of the vertical locations in the image this histogram binning takes place in order to find the most probable horizontal position that the lane is because the lane will most likely have the most found pixels.

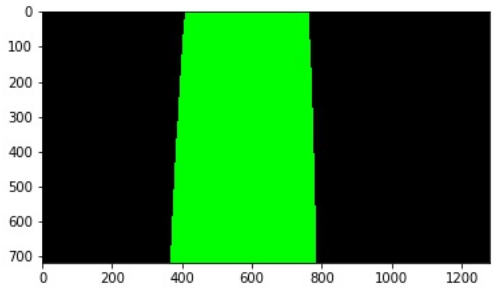

Figure 6: Detected Lanes

2.5 Lane Fitting

After the lane positions have been found, they are passed into a second order polynomial fitter in order to approximate the lanes that have been found. I then filter the coefficients over the past 5 iterations and use a median filter of each coefficient which provides very robust defense against noise. The only sanity check I perform is to check whether the euclidean norm of the distance of the past filtered fit and the current found fit is small and if so then I use the current fit to create a new filtered fit if not then I only use the filtered fit.

These lanes are annotated onto a blank image in order to be warped back out.

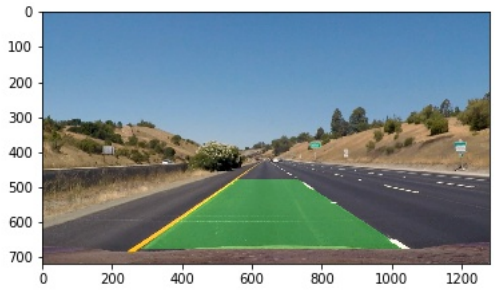

Figure 7: Found Lane

2.6 Calculated of Useful Values

The radius of curvature and position of the vehicle inside the lane were also calculated and printed at the top of the returned image. These values are useful because they could be used in a control loop in order to determine what PWM to send to motor drivers in order to steer the vehicle since annotating the lanes is only useful for human visualization and not actually controlling the vehicle.

2.7 Second Perspective Transform

The found lanes are warped back out into the original perspective. The rectangle of the found lanes turns into a trapezoid and the opencv function fillPoly and addWeighted are used in order to annotate the original image of where the found lanes are.

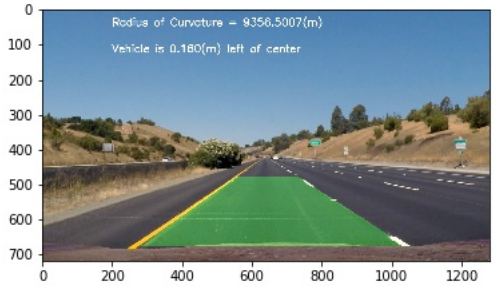

Figure 8: Annotated Lane

3 Problems

The biggest problem that was faced with this algorithm was that the lanes needed to be in roughly the same place and also needed to be very pronounced. If either of these criteria failed then the lanes would not be updated and whatever the first lane that was found would be stuck on the screen the whole video. This is obviously not good and could potentially be fixed by implementing a dynamic threshold based upon the overl pixel intensity of the current iteration rather than a constant value.