Advanced Lane Finding Project

Introduction

The goal of this project is to develop a software pipeline that identifies the lane boundaries of the video for autonomous driving from the car's front camera Camera calibration.

Environment

- Linux 18.04, Python3.6, OpneCV 3.1

File

- main.ipynb : main code (create result viedo)
- utils.ipynb(utils.py) : sub code (create result image)

Step of this project are the following

- *step 1 : Use OpenCV to apply camera calibration through the chessboard image. You can use this method to obtain a matrix for the camera. As a result, you can modify the distorted image to avoid distortion.*
- *step 2 : Using birds eye view a perspective of lane lines for transformation to warp image*
- *step 3 : Using color thresholds to create a binary image which isolates the pixels representing lane lines.*
- *step 4 : Detects lane line pixels and calculates lane polynomials.*
- *step 5 : Calculates the curvature of the lane from the point of view of the vehicle.*
- *step 6 : Convert the image back to the original image form.*
- *step 7 : Visualize the chation curvature and position in real time.*

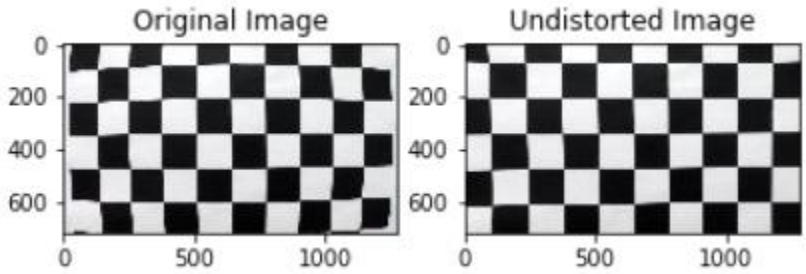
step 1 : Camera Calibration

Camera calibration is calculated using various angles of chessboard prepared in advance.

Using a chessboard with x-point=9, y-point=6, import corners, or image points, through the OpenCV function cv2.findChessboardCorners.

Then, get the values ret, mtx, dist, revecs, and tvecs through the

cv2.calibrateCamera. In fact, only ‘mtx’ and ‘dist’ are required.



[image 1. Undistorted]

step 2 : Perspective Transform

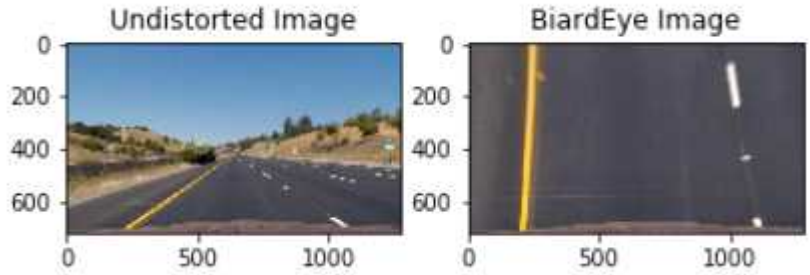
Based on the undistorted image.

Fixed points src = np.float32 ([[490, 482],[810, 482],[1250, 720],[40, 720]]),

dst = np.float32([[0, 0], [1280, 0],[1250, 720],[40, 720]])

The two values are returned to Matrix modified by the cv2.getPerspectiveTransform function.

After obtaining the matrix M, it is possible to obtain the warped image by passing the undistorted image and the size of the M, image to the cv2.warpPerspective function.



[image 2. Road Transformed]

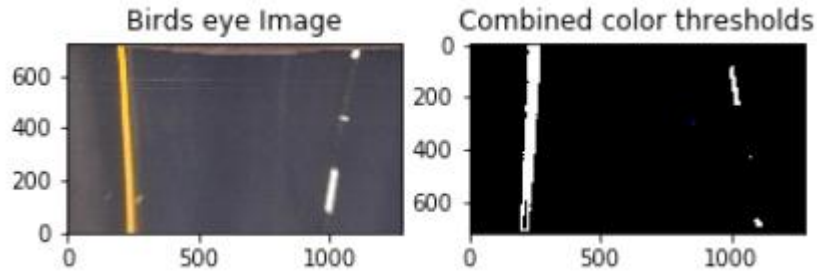
step 3 : Binary Thresholds

This part uses the previously used new eye image and is the result.

Since the raw channel must be disconnected first, convert s_channel from BGR to HLS_channel, and convert l_channel from BGR to LUV, b_channel from BGR to Labd via cv2.cvtColor to LUV.

Then adjust the threshold_min and max values for the s_channel. Similarly, b_channel and l_channel do the same. The channel is then

combined into a combined_binary image.



[image 3. Binary Example]

step 4 : C*alculates lane polynomials.*

At this time, I could only separate pixels belonging to the lane line using a combined binary image. The next step was to fit a polynomial to each lane line as follows.

Identifies the peak in the image histogram that determines the lane position.

Use the numpy function numpy.nonzero to identify all nonzero pixels around the histogram peak.

Use the numeric function numberpy to fit a polynomial to each lane.Polyfit

After the polynomial has been fitted, the following calculations allow the vehicle to be placed relative to the centre:

step 5 : Calculates lane curvature

Calculate the average of the x intercepts at = (right x_int+left x_int)/2 respectively at the two polynomial positions.

Calculate the distance from the centre using the absolute value of the vehicle position minus the midpoint along the horizontal axis distance_from_center = abs (image_width/2 - position).

If the horizontal position of the vehicle is greater than image_width/2, the vehicle is considered to move from center to left and otherwise from center to right.

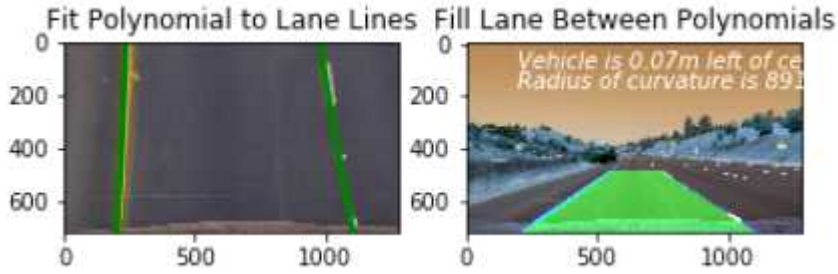
Finally, the distance from the center was converted from pixel to meter

by multiplying the number of pixels by 3.7/700.

The following codes were then used to calculate the curvature radius of each lane in meters:

step 6 : Convert the image.

Finally, the polynomial was plotted on twisted images, highlighted lanes, and twisted the image from the bird's eye to its original visual using another perspective variant, and printed from the center to the radius of curvature.



[image 4. Output]

step 7 : Video Processing

The same method of image processing, but the final step was to extend the pipeline to process images frame-by-frame, enabling real-time processing of image streams in real-world vehicles.

For each lane on the left and right sides of each lane, we created an integrated class so that each lane's stored features can be averaged between frames.

Determines whether a lane was detected in the previous frame, and in this case, only lane pixels close to the polynomial calculated in the previous frame are checked.

The output averages the coefficients of polynomial for each lane line over a range of 10 frames.

[video1 5. Video]