Advanced Lane Finding Project

Camera calibration

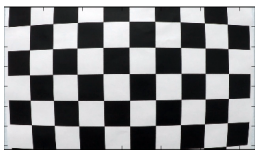
The camera calibration code can be found in camera_calibration.ipynb ipython notebook.

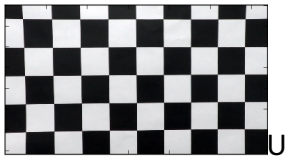
I went through the chessboard images to find 9,6 corners in the images and made a list of object points and image points. Object points are the real world x,y,z co-ordinates of the corners detected by the camera. Image points are the pixel positions of those object points.This can be found in cell[1] line 24 to 37 of the camara_calibration.ipynb notebook

These object and image points are used to calibrate the camera which returns a distortion matrix specific to the camera. Cell[5] line 8

Apply a distortion correction to raw images

I apply a undistort on one of the example chess board images and on the test1.jpg image to ensure things are okay. Cell [5] line 11 to 17. Next 2 images illustrate the distortion being removed.


Original

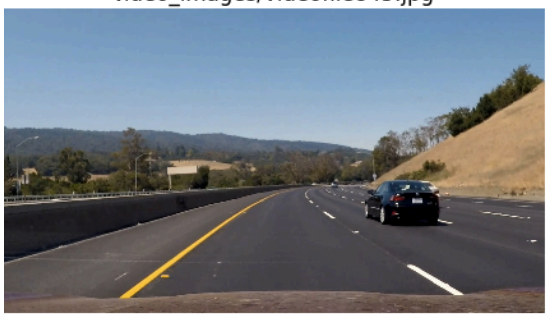


Undistorted


I save the distortion matrix into a pickle file for future use. camera_cal/cam_coefs_pickle.p Cell[6] line 5

Image thresholding

Purpose of this step is to gather only the information about the lane lines and remove all other information. This is accomplished by the apply_thresholding function in Untitles.ipynb I apply gradient threshold, magnitude threshold, directional threshold , S threshold (S part of HLS version of the image) and R threshold.

video_images/videofile845.jpg

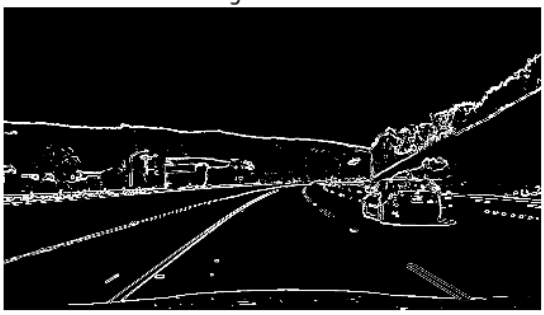


gradx threshold

grady threshold

Dir threshold

mag threshold



S color threshold

R color threshold

Fully thresholded

Perspective transformation

The fully thresholded image is then transformed to get a birds eye view of the region we are interested in ie the lanes. I use opencv's perspective transform to do this.
Untitled.ipynb, cell[2] corner_unwarp does the perspective transform

I use the below source and destination co-ordinates.
Source: (359,619),(532,495), (760,495), (960,619)
Destination: (398,633),(398,183),(933,183),(933,634)

The below image shows the result of the perspective transform

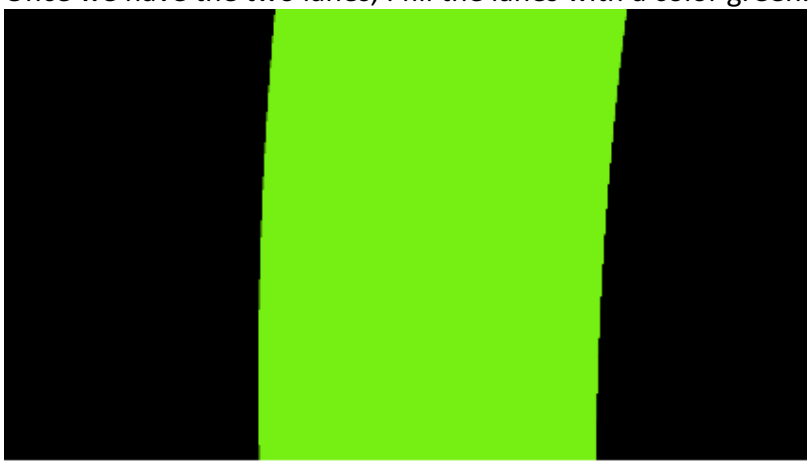
Detecting lane lines

Now, that we have the bird's eye view lane lines. We need to detect the lanes and fit a 2nd degree polynomial to it.

The lane detection is done by using a sliding window to search through the histogram of the image to find the 2 places where there are peaks. These peak windows contain the lane lines. Then we fit a polynomial through the center of these windows. One for left and another for right.

Code for this section is found in Untitled.ipynb, cell[2] : windowed_lane_finder function

Once we have the two lanes, I fill the lanes with a color green.



Curvature of the road and vehicle position

Curvature of road and vehicle position is calculated with the left and right lane information. A pixel to meters assumption is made based on information given in the class videos

ym_per_pix = 30/720 # meters per pixel in y dimension
xm_per_pix = 3.7/700 # meters per pixel in x dimension

Curvature of the road is calculated using the lane line equation, but by converting the pixel curvature to real world curvature.
Lines 153 to 163, cell[2] calculates the curvature

Vehicle position is approximated by assuming that the camera is at the center of the car and the deviation from the center of the lane gives the car's deviation from the center of the lane lines

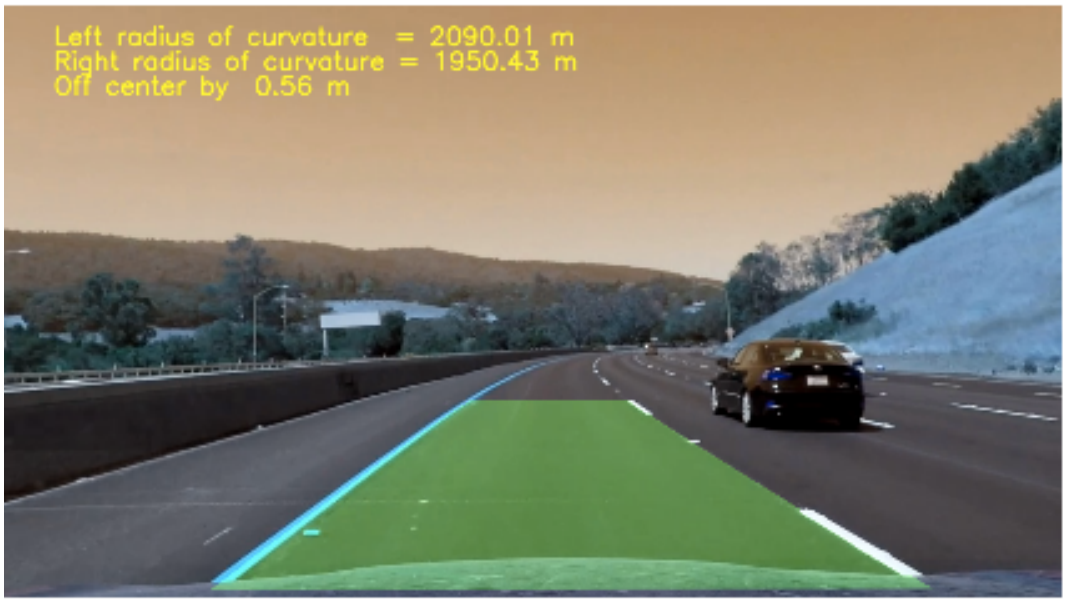
Lines 166 to 183 cell[2] does the vehicle position calculation

Display the lane in the original image with curvature and position information

The detected lane lines are warped back with inverse perspective transform and written back to the original image. Lines 346 – 352 , cell[2] does this.

The radius of curvature of the left and right lanes, as well as the positional information of the car are written onto the image as well. Lines 354-357, cell[2]

The project video can be found in advanced_lane_lines.mp4



Issues faced and steps taken to overcome them

Thresholding issues

When the images had a lot of shadow regions, without the S threshold, the yellow lines would not be detected. But, the with only the S threshold, sometimes the white lanes were not detected well, leading to crazy lines. Adding a R threshold improved the detection of white lines

Sometimes, the right lane of the lane next to the lane our car is driving on would get detected as a lane. So, I added a region of interest to eliminate such issues.
Cell[2], region_of_interest function (line 280)

Sanity checks

If also added a sanity check to ensure that a calculated line isn't very different from the most previously calculated line. I do this by calculating the Euclidean distance between the two line equations. If they are very far off, I re-use the most recent good lane line.
Cell[2]: similiarity function calculates the Euclidean distance.
Cell[2]: lines 138 to 148 does sanity checks

Further improvements

- Window search searches the image every frame instead of localizing the searches to regions closers to previous frame's lanes
- Lanes need to be averaged for previous 5 or 10 frames to make it less wobbly
- Perspective transform took in hard-coded cor-ordinates. Instead, choosing the src and dst points should be made programmatically, so that it works for any resolution