PROJECT

SELF DRIVING CAR

ADVANCED LANE LINE FINDING

The purpose of this project is to process a video of a driving lane and interpolate a polygon over the drivable portion of lane. Along with this the radius of curvature and distance from the center also have to calculated and displayed.

Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

Images taken from pin hole camera usually have distortion. By distortion we mean the edges of the images are rounded due to which the shape of the objects changes. This leads to incorrect measurements. To correct distortion we need to to calibrate the camera so that it preprocesses those images before presenting them to us.

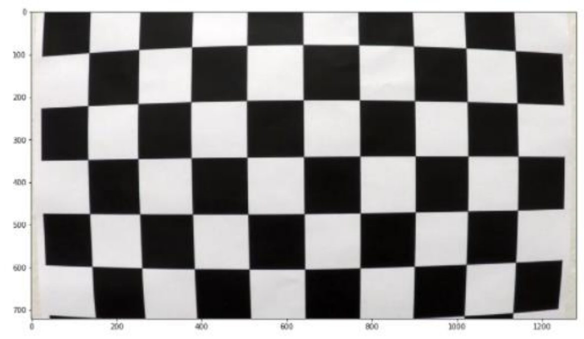
We correct distortion by taking pictures of known shapes and subsequently comparing those images with the original shape. Here we took a regular chessboard to undistort images. We read around 20 images of a chessboard. These images should generally be taken at different angles. We compare the coordinates of important features of the original object and the coordinates of those features in the image.

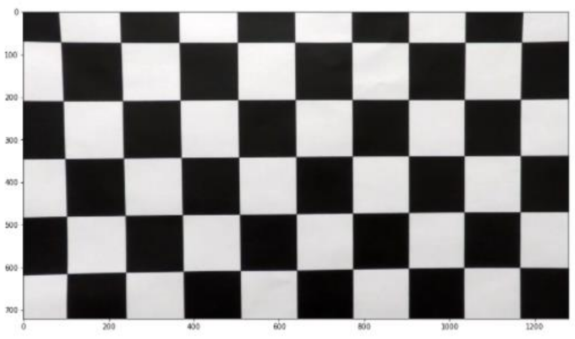
To find the coordinates of corners we use findChessboardCorners function of OpenCV.

We used drawChessboardCorners function to identify the various cornersof the chessboard image visually. We get the image points using these functions. We estimate the object points of the original chessboard manually

There is a calibrateCamera function using which we find the camera matrix and distortion coefficients.

Now we can undistort any image using the undistort function.





Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result

I tried a number of methods of feature extraction from the lane images.

Following are some of them:

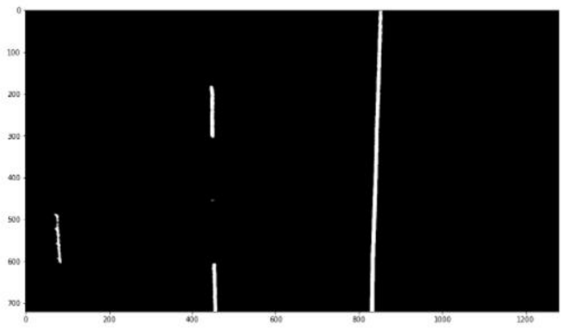
1) RGB & HLS
2) RGB & sobel
3) LAB & RGB
4) HLS and LAB

But the last methos i.e. HLS and LAB gave me the best result. I extracted the L channel of the HLS image and B channel of LAB image. I thresholeded these two different images with various experimental values. I used a purely experimental approach to arrive at the requisite methos.

Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

Perspective transform was used to get a bird's eye view of the lane. This means we could see something from the top of the lane. I identified the points of the lane in the image that I wanted to concentrate on. Exactly I identified four points. Now I needed to map these four points to another set of four points that will appear on the warped image.

I used the getPerspectiveTransform and warpPerspective functions of OpenCV to arrive at this. The getPerspective transform function returned a transform matrix that represents the source points to destination points mapping.



<u>Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?</u>

For identifying and drawing the lane lines first I applied the image processing pipeline to the image and got a warped image. Then I used a sliding window approach to identify the relevant pixels of lane edges. First I took a histogram of the bottom half of the image. Then I tried to find the peak of the left and right halves of the histogram. These will further act as the starting point of the left and the right lanes. Then divided the image vertically in 10 equal regions. Then I calculated all the pixels that were non zero.

Then in a loop that runs for as many times as there are windows, we calculated the pixels of the lane by taking a margin of 80 pixels. The most important calculation is that of the bottom most window. Rest of the windows can be calculated using the margin value.

We collect all the indices of the left and right lane edges. Here we get the x and y coordinates of all the pixels that belong to lane edges.

Subsequently we fit a second order polynomial using these pixel points. We get the coefficients of these polynomials for both the edges of the lane.

We used Np.polyfit function to calculate the polynomial fit.

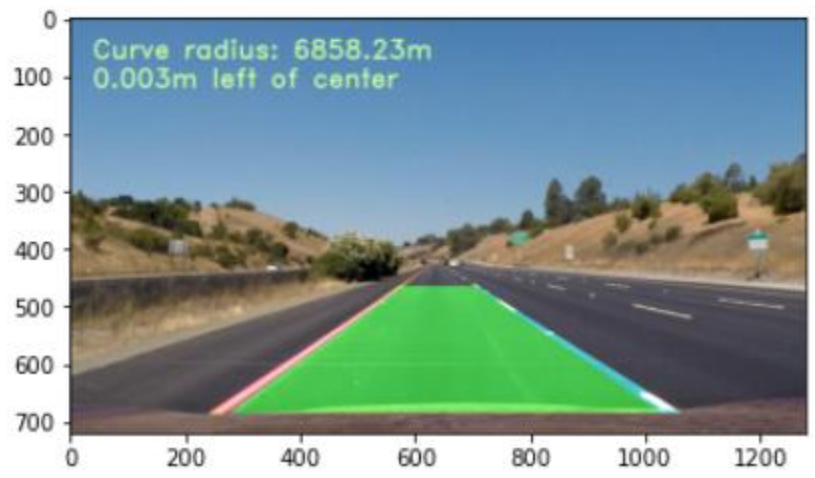
Then I created an empty image of the same dimensions as the original image. Then using linspace I recasted all the points to x fit. Finally I used a fillPoly functions to fill a polygon that surrounds these points.

<u>Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.</u>

First we defined conversions in x and y pixels space to meters. Then we defined the y-values where we want the radius of curvature. For this I chose the maximum y value.

Then I identified the x and y positions of all nonzero pixels in the image.Then I extracted the left and right lane pixel positions.

Then I fit a polynomial to these points using np.polyfit. Subsequently I converted these distances calculated using these polynomials into metre distance using the metre distance per pixel calculation. We get a left radius of curvature and right radius of curvature. We average these to arrive at the final radius of curvature value.



Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

I faced a lot of issue in generalizing the lane pixel finding logic. For some images it was able to find the binary pixels properly but for others it used to come totally blank. For this I had to do a lot of experimentation.

I suspect that my pipeline can fail on lane video that is taken with a low resolution video since my image pipeline heavily depends on HLS and LAB transforms . These will not give the desired result in that case.