Advanced Lane Lines

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.

Camera Calibration:

The code for this step is contained in the python script **Camera_Calibration.py**

Camera images are usually distorted due to the bending go the light on the edges of the lease. In our case to find out the lane lines, distortion correction is necessary to get a true gauge of how the lane is represented in the 2D image.

To do this, I started with a set of 16 chess board images provided. The basic idea was to find the corners in the distorted chess board images and map them to undistorted corners in the real world.

1-I start by preparing the obj points which are undistorted coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time.

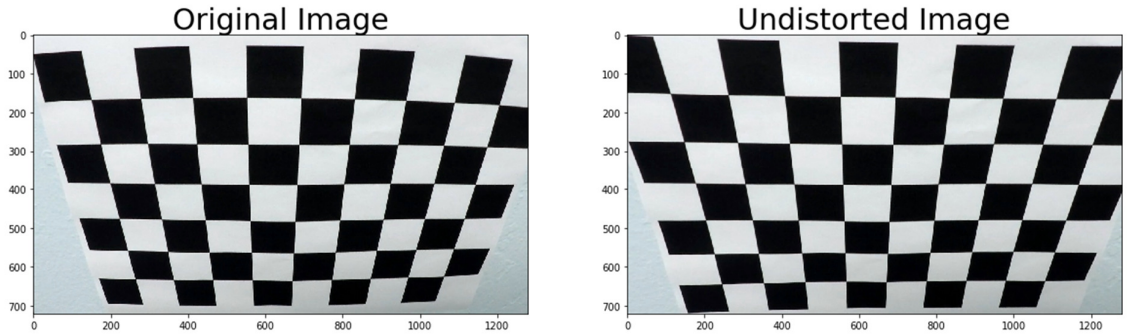
2-All chessboard corners in a test image are detected.

3- Imgpoints which are the corners of distorted image in 2D world are appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

4- Output objpoints and imgpoints are used to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function.

5- This distortion correction is applied to the test image using the cv2.undistort() function and obtained this result:

After that these distortion coefficients are written to wide_dist_pickle.p to be used later for distortion correction of camera



The main program is submitted as Jupyter Notebook main_project.ipynb

Distortion-correction

Once the distortion coefficients are calculated, distortion correction is applied to all the test images provided in the test image folder. The function undist() in the block 3 of notebook is used for this.

Block 1, 2 and 3 in the provided jupyter notebook reads in the test images and the distortion

coefficients from the wide_dist_pickle.p and apply distortion correction to all.

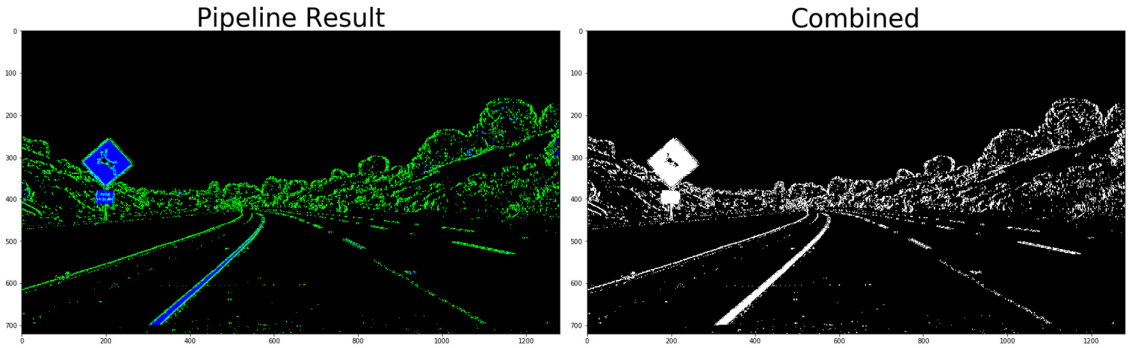
Color Transform and Gradients

The code for my thresholding includes a function called thresholding(), which appears in in the 4th code cell of the Jupyter notebook)

The main objective of this step was to identify lane pixels as accurately as possible. I used different colour transforms and gradient transforms and finally chose HLS colour transform for the theresholding along with the gradient threshold in x direction as it provided me with the best result.

I used a combination of HLS color threshold and gradient thresholds to generate a binary image

Here's an example of my output for this step. (note: this is not actually from one of the test images)

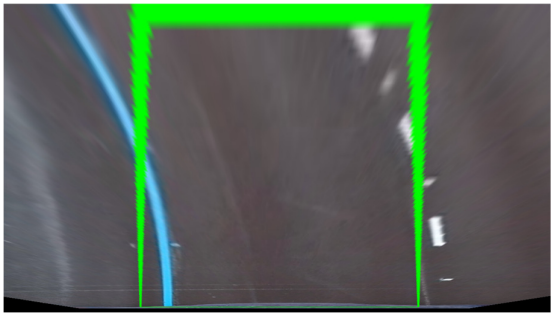


Perspective Transform

In 2D images, objects appear smaller the farther away they are from a viewpoint. So it is better to perform a perspective transform on the undistorted thresholded image to have a birdeye view of how the lane lines are so that later, the curve fitting through them can be done accurately.

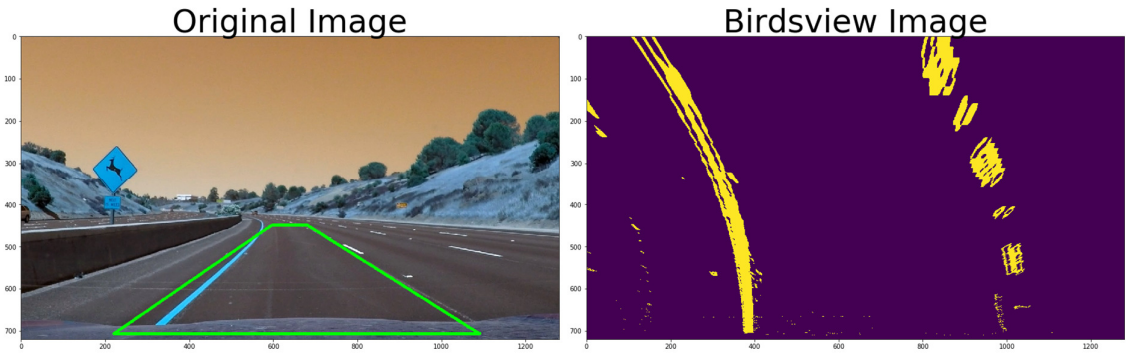
The code for my perspective transform includes a function called perspect(), which appears in in the 5th code cell of the Jupyter notebook). The function takes as inputs an image (img), as well as source (src) and destination (dst) points. I chose the following source and destination points:

src= np.float32(
[[220,707],
[597,448],
[680,448],
[1091,707]])



dst= np.float32(
[[320,707],
[320,10],
[970,10] ,
[970,707]])

I verified that my perspective transform was working as expected by drawing the src and dst points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



The perspective transform is applied to all the pictures and are stored in the output images folder.

The colours look different in the picture as the matplotlib and opencv reads in images differently(RGB vs BGR).

A functions imagepro is defined which applies all the image processing - the undistortion, thresholding and perspective transform.

Lane Equation

This is done in the code cell 7,8 of the Jupiter notebook. The function findlanefirst() takes in the binary image after processing and returns the fitted curve coefficients.

plot_full_search() shows the results of the findlanefirst()

After applying undistortion, thresholds and perspective transform, the lane lines are clearly visible (as shown in figure 9).

Next step is to fit a polynomial equation through it. A second degree polynomial equation is fitted through the lane pixels using following algorithm:

1- Detect the location of lane at the base of the picture by taking histogram of the lower half of the binary image. The peaks in the histogram are a measure of the x position of lane.

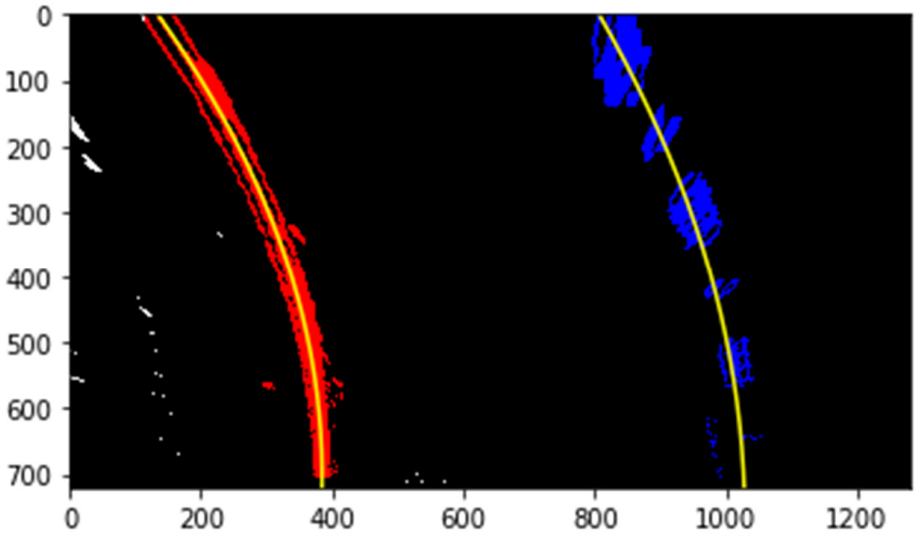
2- All the non-zero pixels are identified in the image

3-Next a sliding window is defined at the x positions of lane and all the non-zero pixels appearing inside the window are identified.

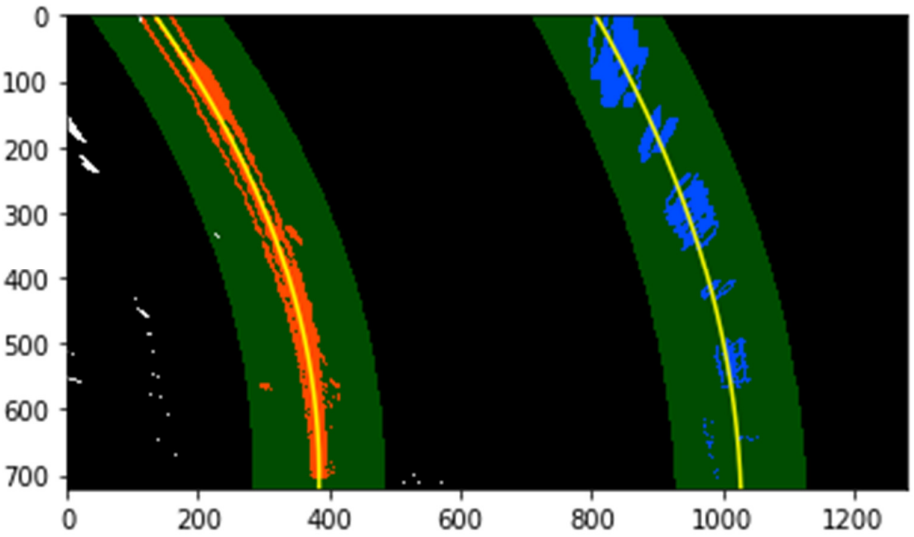
4- The sliding window is moved in Y direction to find more non zero pixels and offsetted in X to their mean in case we find more than a set number.

5- Once we have all the good pixel candidates for lanes in the entire image, a second degree polynomial is fitted through them $f(y)=Ay2+By+C$

6- The steps are repeated for left and right lane line separately.



Once you know where the lines are, you have a fit! In the next frame of video, you don't need to do a blind search again, but instead you can just search in a margin around the previous line position. findlane() function in code cell 9 is used to find the lane lines once you already have the fit.



Radius of curvature and Distance from Centre

once the polynomial is fitted through the lane lines, its radius of curvature is calculated using Curvdist() function appearing in the 12th code cell of the Jupyter notebook. The formula used for the curvature:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

In order to cover the pixel values into the road units the following conversion is used

ym_per_pix = 30/720

xm_per_pix = 3.7/700

where they are in meters per pixel units

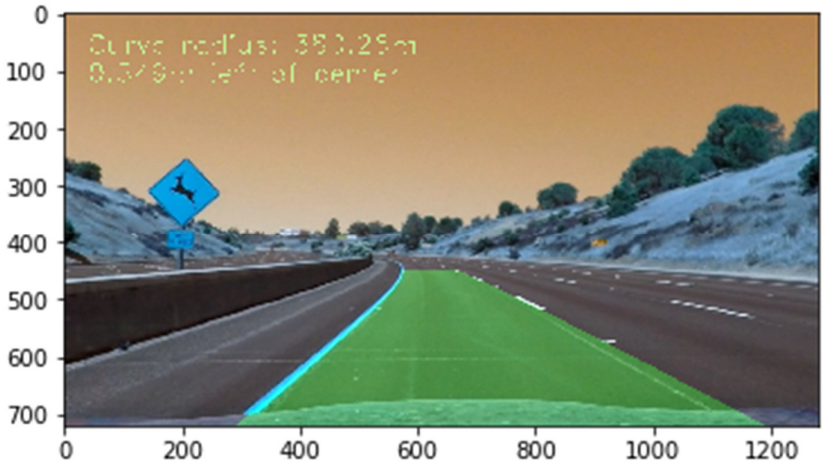
In order to calculate the distance from the centre, assumption is made that the camera is mounted on the centre of the car. The average of the left and right lane is taken at the bottom of the image and then subtracted from the centre of the image.

The distance is then mutiplied by the xm_per_pix to convert it into metres.

Warping back

Once the lane lines are identified, plotback() function is used (13th code cell of Jupyter Notebook) to plot the full lane back onto the original image. The inverse Minv matrix obtained from the prospect function is used to plot the identified lane back onto the actual image.

draw_data() function in code cell 14 is used to write the results of curvdist function onto the image. The final image is shown below:



Class Lane

A class lane is defined in the 15th cell block. to keep track of all the interesting parameters we measure from frame to frame. The main ones used are the current fit and best fit. In order to smoothen the results, the average of 5 frames or 5 fit coefficients is taken and passed as the best fit.

Main pipeline

process_image() is the main image function that takes in the video frame one by one and find the lane lines by using following steps:

1- Get the binary image using aforementioned imagepro()

2- Find the lane fits depending on if the current fits are available or not

3- Calculate curvature and lane width and check if the values make sense

4- Use the average of 5 previous fits for smoothness or if the sanity checks in step 3 reveal that the lane lines you've detected are problematic, skip these fits.

Video :

VideoFileClip() and fl_image(process_image) functions are used to create a clip and applying the process_image function to every frame of the video. The code performed good on the test video provided and it is able to detect the lane without significant departures or failures. It did not perform well on the challenge video.

Problems/ Issues

- The pipeline failed in the challenge video where the lane has various colour within it. So we need to improve on the lane detection part.
- A better parameter optimization or a better technichque to get all the edges in picture.
- Also it would be a good idea moving further to store all the coefficients of the fits as a history from frame to frame and look for any significant departures.