Advanced Lane Finding Project

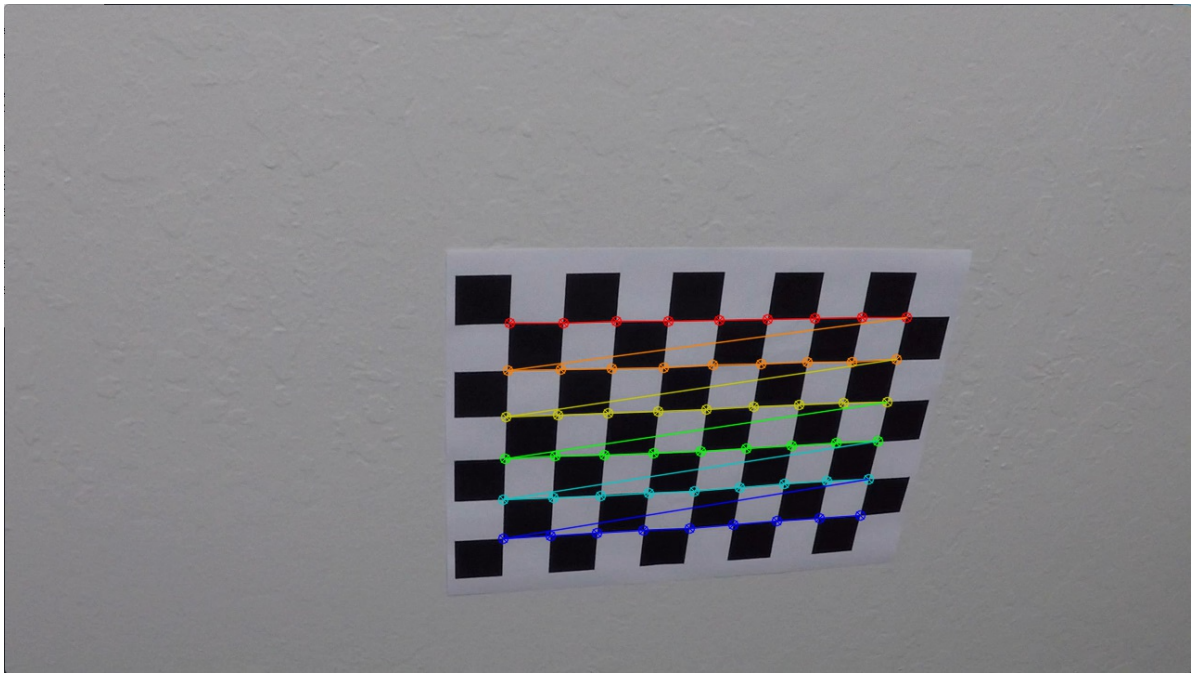
The goals / steps of this project are the following:

* 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.

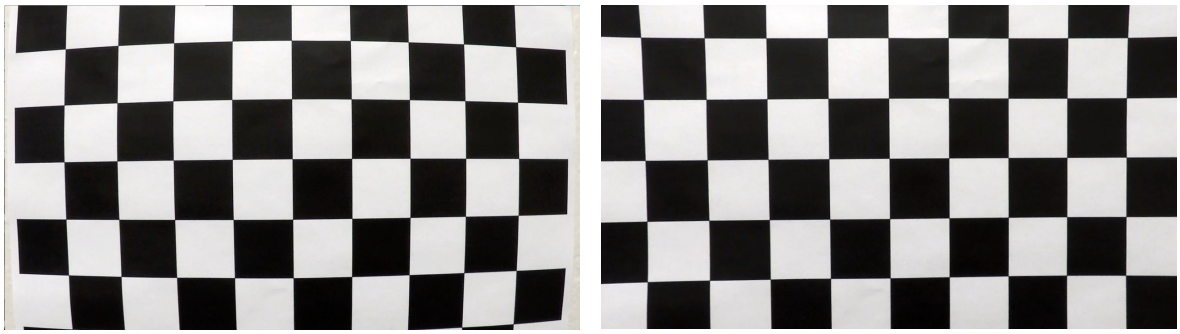
Camera Calibration

The images provided in director “**camera_cal**” directory used for calibration.

In the second cell of **lane_finding.ipynb** calibration images used to generate "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. 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. Object points are stored in **objpoints** array. `**imgpoints**` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection. Corners found are marked in calibration images and stored in the **output_images** directory with images “cornersX.jpg” sample image is shown below.



In the third cell `objpoints` and `imgpoints` used to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. This correction is used on the test image using the `cv2.undistort()` function. Following figure shows original image and undistorted image. Calibration data stored is stored in a file.



Pipeline (single images)

Example of a distortion-corrected image.

In the fourth cell of the notebook distortion correction to test images applied. First calibration data is loaded from saved pickle from the previous step. By use of loaded calibration data test images are corrected and stored in 'output_images/undistorted_test/' directory. An example of correction is shown below. First image is original image second image is undistorted image.

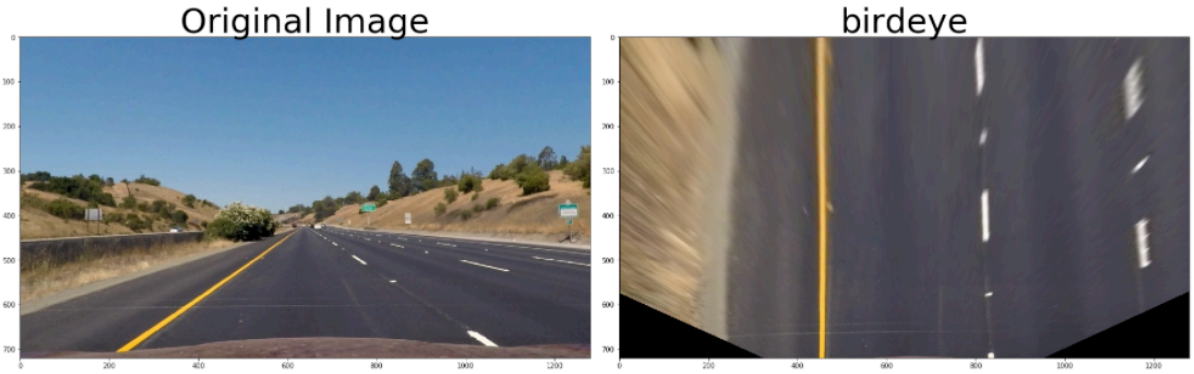
Perspective Transformation

In the fifth cell of the notebook conv_birdeye() function is defined. This function accepts undistorted image (img) as an input. Source and destination points are hardcoded as follows. h,w and parametes are height and width of the image.

```
src = np.float32([(580,460),(710,460), (250,680), (1040,680)])
dst = np.float32([(447,0),(w-447,0),(447,h),(w-447,h)])
```

With use these points perspective transform matrix M and inverse transform matrix Minv is calculated. With use of M perspective transformation is performed on input image. Transformed image in matrices are returned.

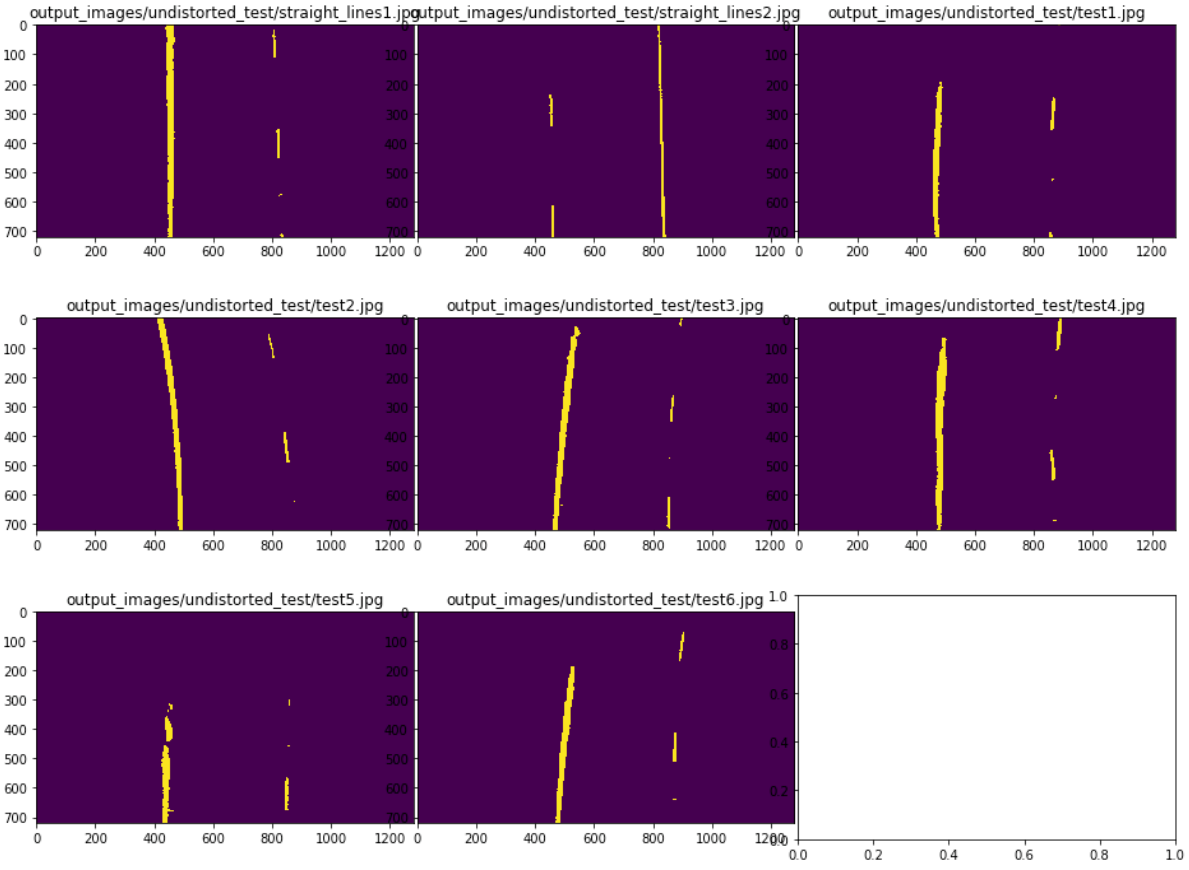
In the sixth cell perspective transformation is tested with one of the undistorted test images. Following shows input image and output image. Straight lines are parallel in the birdeye image which shows transformation is correct.



Color Transformation

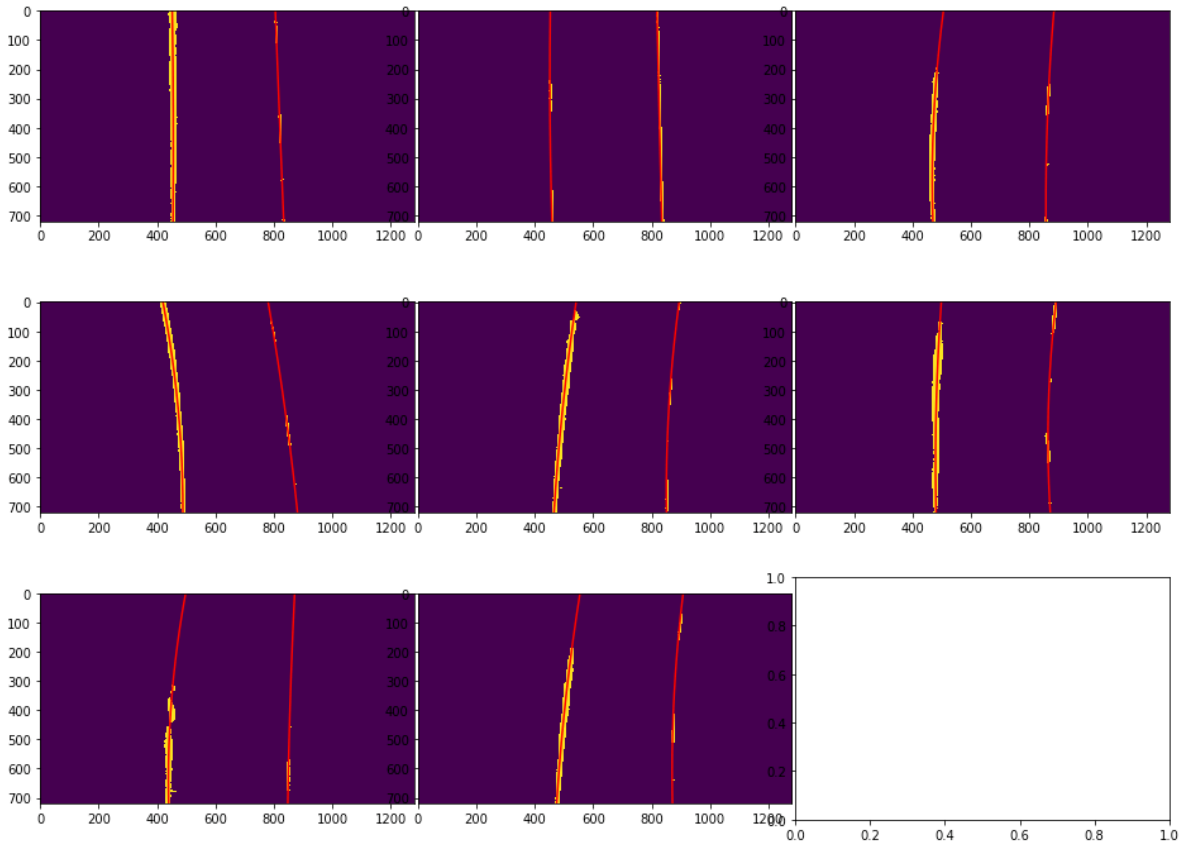
After several transformation functions are implemented between 8th and 12th cells. HSL and LAB color schemes are selected. L channels and B channels selected and merged for better identifying the lanes. Also a region of interest (ROI) function is used to eliminate other areas.

This functions are tested on test images in the 13th cell following is the output of test.



Polynomial Fit

The code implemented in the lesson is used to fit polynomial. This function is implemented in the 14th cell. In the 15th cell a function for ploting determined polynomials is implemented. Polynomial fit is also tested with the test images. Following is the output generated by this function.



Curvature and Car Location Determination

Get_curvature function is implemented to understand the road curvature. First the birdeye image lane width and size of a lane line is measured. Since the width of the lane(3.7m) and size of lane line(3m) is known number measured pixels are used to create pixel to meter conversion. Following examples shows the measurement.

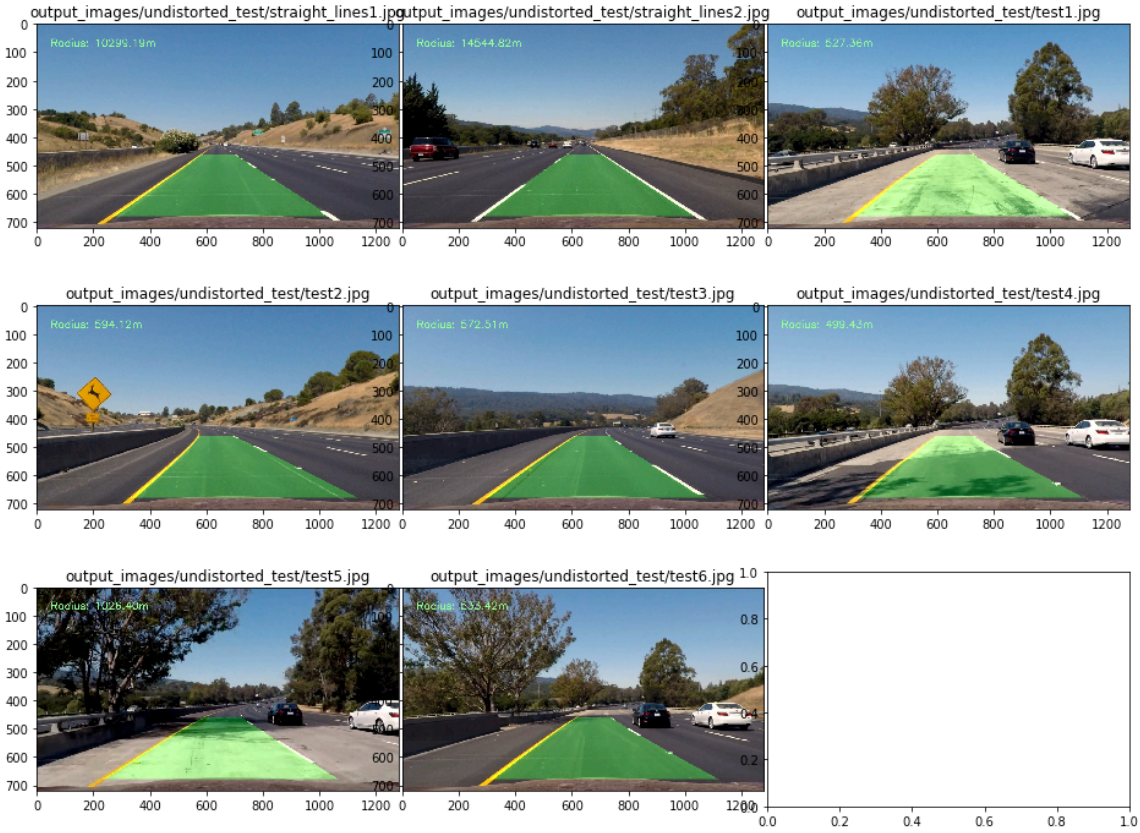
Code explained in the lessons used to calculate the curvature of both lanes.

Get_distance function is implemented in the next cell to calculate the car position within the lane. Basically distance from the midpoint of the lane and the midpoint of the image is subtracted to understand displacement. Curvature and car position calculation is tested together with line fitting. Following output is produced for radius information.

```
2668.38994891 17929.9833201
1422.10973325 27667.5305051
378.355063503 676.355827078
321.266831009 866.976303358
732.93994358 412.079968745
654.790140459 344.07916082
296.649794566 1756.14506239
643.014657996 423.82632397
```

Plotting the lane back to the original image

Draw_lane_poly function is implemented to be able to draw the polygon back to the image. The invare conversion matrix calculated by the bird_eye() function is used to convert calculated polygon to camera view. Draw_poly() function, get_curvature() and get_distance() functions are tested on test images and following output is produced. This picture shows function is working correctly. Printing curvature on the image is also tested with this pipeline.



Pipeline (video)

Implemented functions are tested with the video provided. Output video is stored in :

./ project_out_video.mp4

Discussion

Mostly the techniques and sample codes provided during the class were enough to create acceptable result. Most difficult part is again to play with thresholds and select a color space and layer which is less prone the changes in the road texture and lighting conditons.

Especially perspective transformation part was very informative. Birdeye images can be fused together with satellite images to create better detection.

Current project has a lot of constants. These numbers possibly won't work if camera is tilted due to changes in road slope. A sensor (gyro) can be used to understand this angle in real world. (or slope from a digital map).