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 ("bird-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.

Files included in this submission

- **writeup.pdf** - The present documents, detailing how the lane detection pipeline was implemented.
- **code.py** and **function.py**. These two python files contain the code that implements the lane detection pipeline.
- **project_video_output.mp4** - Video highlighting the lane as a result of the calculations made in the python code.
- **output_images** - this is a folder containing the images produced during the setting up and test of the pipeline and included in this document.

Camera Calibration

The images captured by a camera are always distorted to some degree, due to the use of lenses. To remove this distortion one has first to compute the camera matrix and distortion coefficients, and then use them to correct the images.

In the code provided in this submission, the matrix and distortion coefficients are calculated in the function `find_calibration_parameters` (file "functions.py", lines 15-45) This function:

1. reads in a set of chessboard pictures taken at different distances and angles,
2. identifies the intersection points of the chessboard in each image (using the cv2 function `findChessboardCorners`),
3. maps the intersection points into a regular grid of points evenly separated on the x and y axes (using the cv2 function `calibrateCamera`). Through this mapping, the matrix and distortion coefficients are calculated.

Once the matrix and distortion coefficients are known, they can be used to correct any image captured through that camera. In the code submitted this was done using the cv2 function `undistort` (file "code.py", line 47).

Fig.1 show how one of the chessboard images before and after correction. Because of the regularity of a chessboard pattern, this before/after comparison is useful to assess whether the image correction works properly. In our case it seems to be the case.

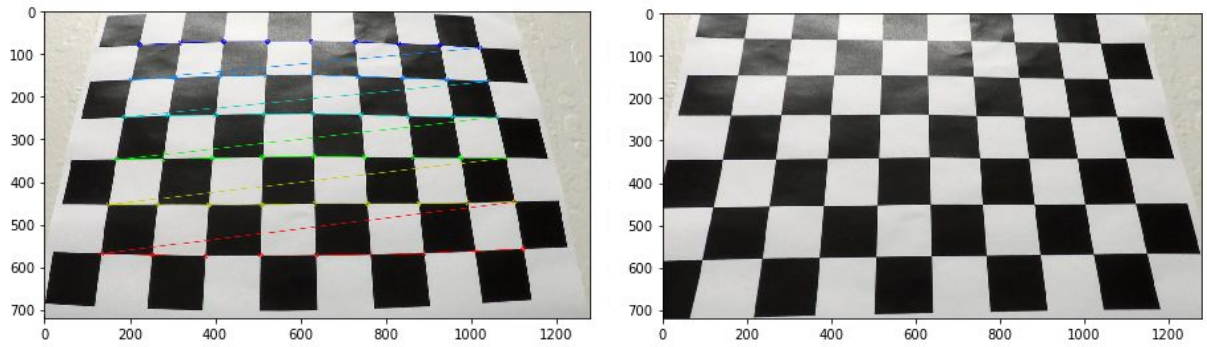

Figure 1 - Example of chessboard image before (left) and after (right) camera correction.

Fig.2 shows the same before/after comparison for an image of the road lane. For this kind of image it is more difficult to appreciate the camera correction. However it is still possible to see the difference between the corrected and uncorrected images when focusing on their edges (for example on the bottom edge where the car hood is visible).

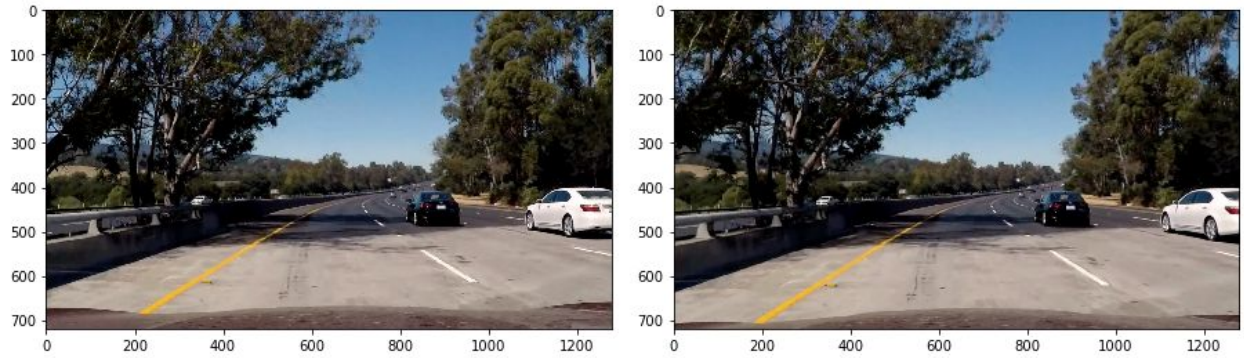

Figure 2 - Road image before (left) and after (right) camera correction.

2. Thresholded binary image

The creation of a binary image from the original road image represents here the first step towards the identification of the lane-lines.

For each road image, the binary image was created by computing and highlighting different aspects of the picture. In particular a Sobel transform along the x axes was applied to the blue and red channels. The original image was also converted into the HLS colormap, and subsequently filtered on the L (lightness) and S (saturation) components.

For each of these transforms and filters a binary image was created where the transformed value of each pixel was mapped to 1 if comprised between specific thresholds, or to 0.

These binary images were finally combined in one single binary image by means of overposition. The code used to obtained the binary thresholded image is contained in the file “code.py”, lines 50-68.

Fig.3 shows the binary images obtained by applying different transforms and filters to the same original road image and the final final result obtained by combining them.

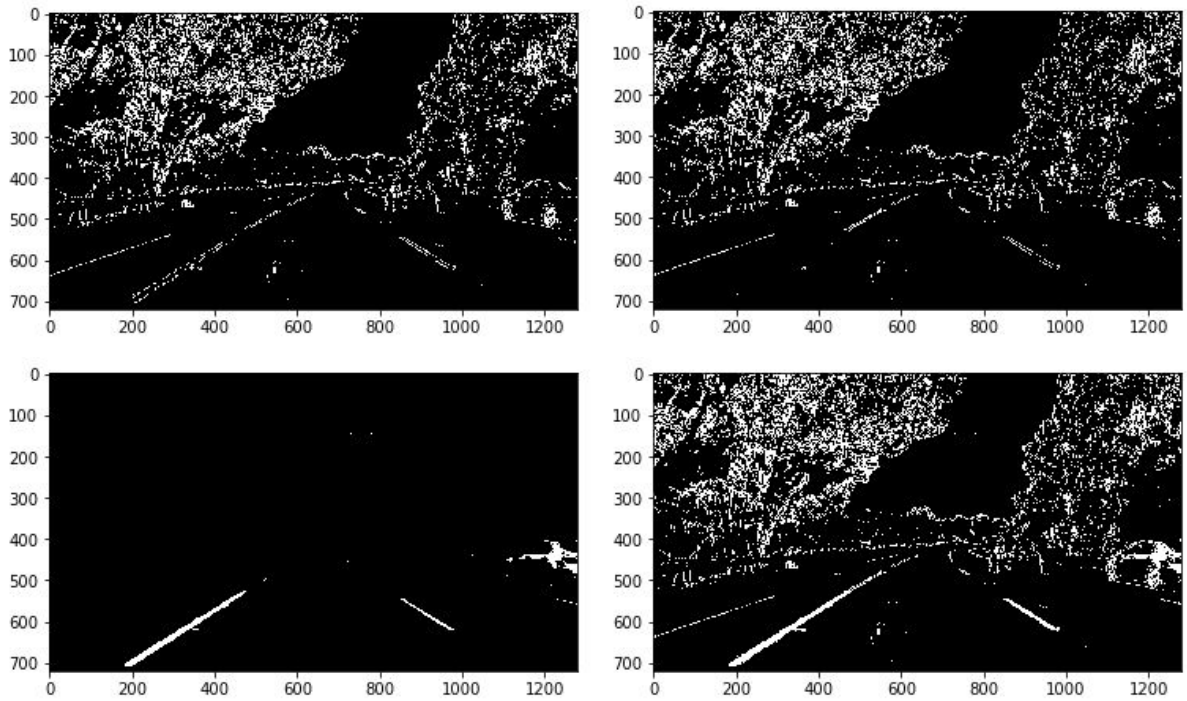

Figure 3 - The binary images obtained by applying the Sobel transform along the x axes on the red and blue channels of the original road image are shown at the top-left and top-right panel respectively. The bottom-left panel shows the binary image obtained by filtering the original image on the L (lightness) and S (saturation) channels of the HLS colour-map. The bottom-right panel shows the final binary image, obtained by overimposing the other three.

3. Perspective transform

The code for my perspective transform includes the function `find_perspective_tranform_parameters` (file "functions.py", lines 150-160). This function maps four source points selected from the lane-lines of a road image to four destination points that define a regular quadrilateral shape with parallel opposite edges. Both the source points and the destination points were hard-coded in the function.

Through this mapping the function can calculate the transform matrix M that is subsequently used to transform the original road images into bird-eye views of the road.

Fig.4 shows the original picture that was used to select the 4 source points, and its corresponding bird-eye view. The bird-eye view is obtained by applying the perspective transform matrix M to the original picture through the function `warpPerspective` (file "code.py", line 74). It should be noticed that the image chosen to select the source points captures a straight road tract.

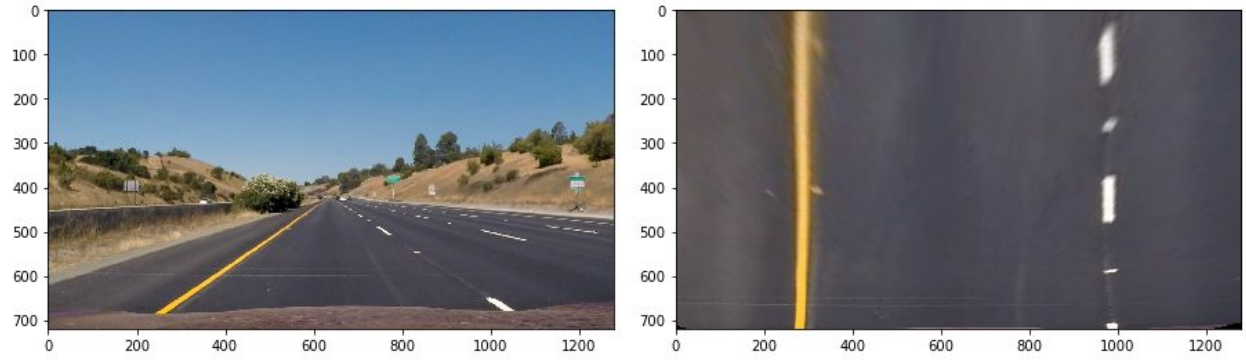

Figure 4 - Road image before (left) and after (right) the perspective transform. The original undistorted image (left) was used to find the transform matrix. The image is particularly suitable for this purpose as the lane-lines are straight.

In the pipeline followed in this project it is actually the previously constructed binary thresholded image to be transformed into a bird eye binary picture. This is shown in Fig.5, where a binary bird-eye image is generated by transforming a binary image of a slightly bending road.

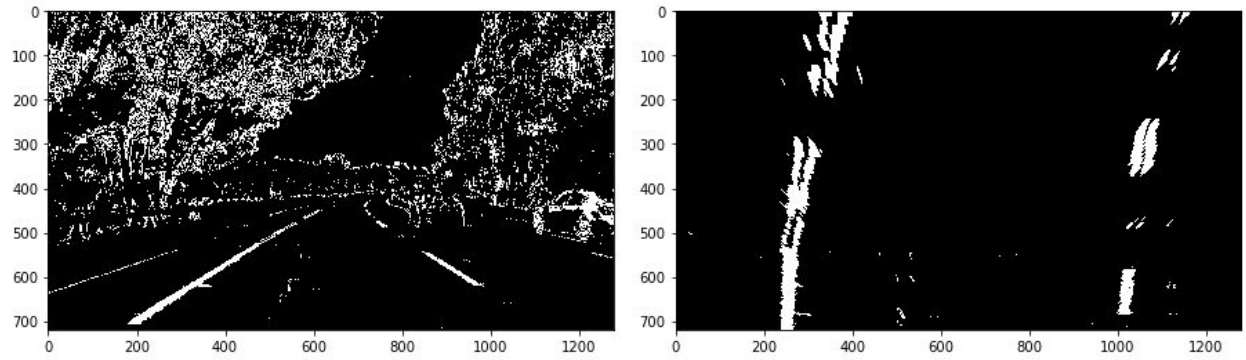

Figure 5 - Example of perspective transform on a binary thresholded image (left) resulting in a binary bird-eye view or the road (right)..

4. Lane-lines detection

Once obtained a bird-eye binary image of the road, the next step consists of identifying those pixels that belong to the lane-lines.

In the code this is done by the function `find_lane_lines_pixel` (file "functions.py", lines 169-233). Roughly speaking the process consists of dividing the image in horizontal stripes and, in each stripe, identifying the two regions (along the x axes) of maximal pixel density. These regions will contain the pixel belonging to the lane-lines.

The x and y coordinates of the pixels belonging to the left and right lane-lines are subsequently fit with two second order polynomials (one polynomial for each lane-line). The fitting is performed in "code.py", lines 81-82, and the result shown in Fig.6.

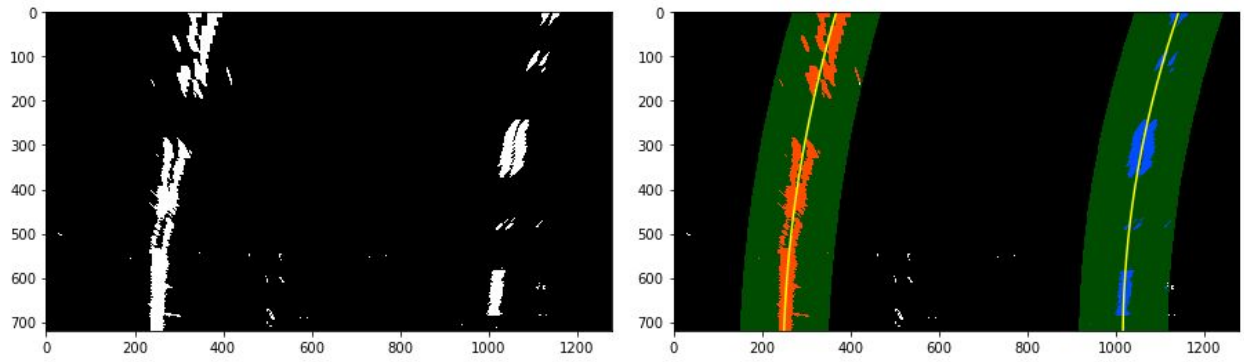

Figure 6 - Bird-eye image of the road before (left) and after (right) lane-lines detection. The pixel belonging to the lane-lines have been highlighted in different colours (red for the left line and blue for the right line). The two yellow lines represent the polynomial fittings.

5. Radius of curvature of the lane and position of the vehicle with respect to center.

To calculate the radii of curvature of the lane-lines I performed a second fit on the relevant pixels, where this time the x and y coordinates were expressed in meters.

The coefficients of each polynomial were then used to calculate the curvature of the corresponding lane-line through the following formula:

$$R = \frac{\left(1+(2Ay-B)^{2}\right)^{3/2}}{|2A|}$$

where A and B are the coefficients of the polynomial $f(y) = Ay^2 + By + C$.

The offset of the vehicle with respect of the center of the lane was calculated through the following formula:

$$\frac{\left(x_{left}+x_{right}\right)}{2} - \frac{Img_x}{2}$$

where x_{left} and x_{right} are the x coordinates of the left and right lane-lines respectively along the bottom edge of the picture, and Img_x is the x coordinate of the middle point of the image.

The code for the calculation of the curvature and the offset is included in the file "code.py", lines 86-97.

6. Example image of your result plotted back down onto the road such that the lane area is identified clearly.

Once identified the lane-lines the results were projected back to the original undistorted image. Fig.7 shows an example of a road image with the lane highlighted in green as a result of the previous calculations.

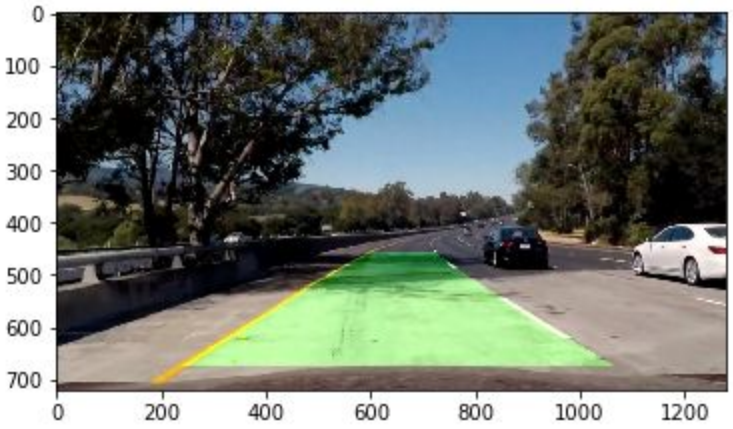

Figure 7 - In this road image the lane has been highlighted to visually show the result of the calculations performed in the pipeline.

The relevant code for this task is in the file "code.py", lines 102-108. The function `project_back` (line 108) is defined in the file "functions.py", lines 278-296.

7. Video

The pipeline described in this document was applied to all the frames of a video capturing the road from a moving vehicle. As a result an output video was generated where the lane of the road was coloured in green. The video is submitted as part of this project, and its file-name is "project_video_output.mp4".

8. Discussion

When applying my pipeline to the "challenge_video.mp4" the result is not as good as for the "project_video.mp4". The reason is that the algorithm does not distinguish particularly well between pixel that belong to the lane-lines and pixels that don't. The reason probably lies in the choice of filters and/or their thresholds used to generate the binary image.

This is where I would mostly focus for further improve my pipeline.