Udacity Project: Advanced Lane Finding

Project consists following components

Processing steps

The pipeline for this project comprises the following steps:

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images
2. Apply a distortion correction to raw images
3. Use color transforms, gradients, etc., to create a thresholded binary image
4. Apply a perspective transform to rectify binary image ("birds-eye view")
5. Detect lane pixels and fit to find the lane boundary
6. Determine the curvature of the lane and vehicle position with respect to center
7. Warp the detected lane boundaries back onto the original image
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Each step is discussed in more detail below

Step 1 & 2: Camera calibration and distortion coefficients

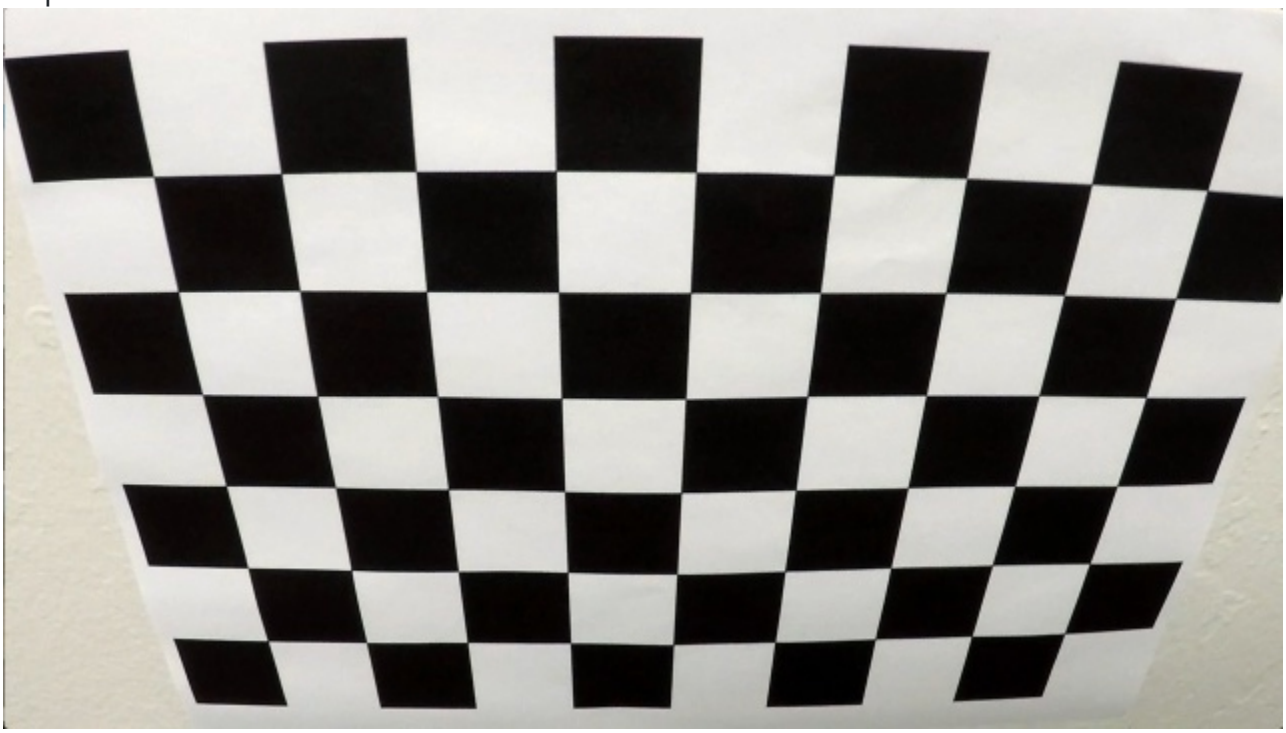
Using the checkerboard images provided with the project the calibration matrix and distortion coefficients are calculated. All required code is contained in the `CameraCalibrator` class. Function `__calibrate` performs the actual calibration. After calibration completes the calibration matrix and distortion coefficients are written to disc. Subsequent runs of the pipeline read the calibration matrix and distortion coefficients from disc, which is faster than performing the calibration again each time.

One complicating factor was that the checkerboard images provided with this project do not contain the same number of detectable corners. This was resolved by manually determining the number of detectable corners for each image.

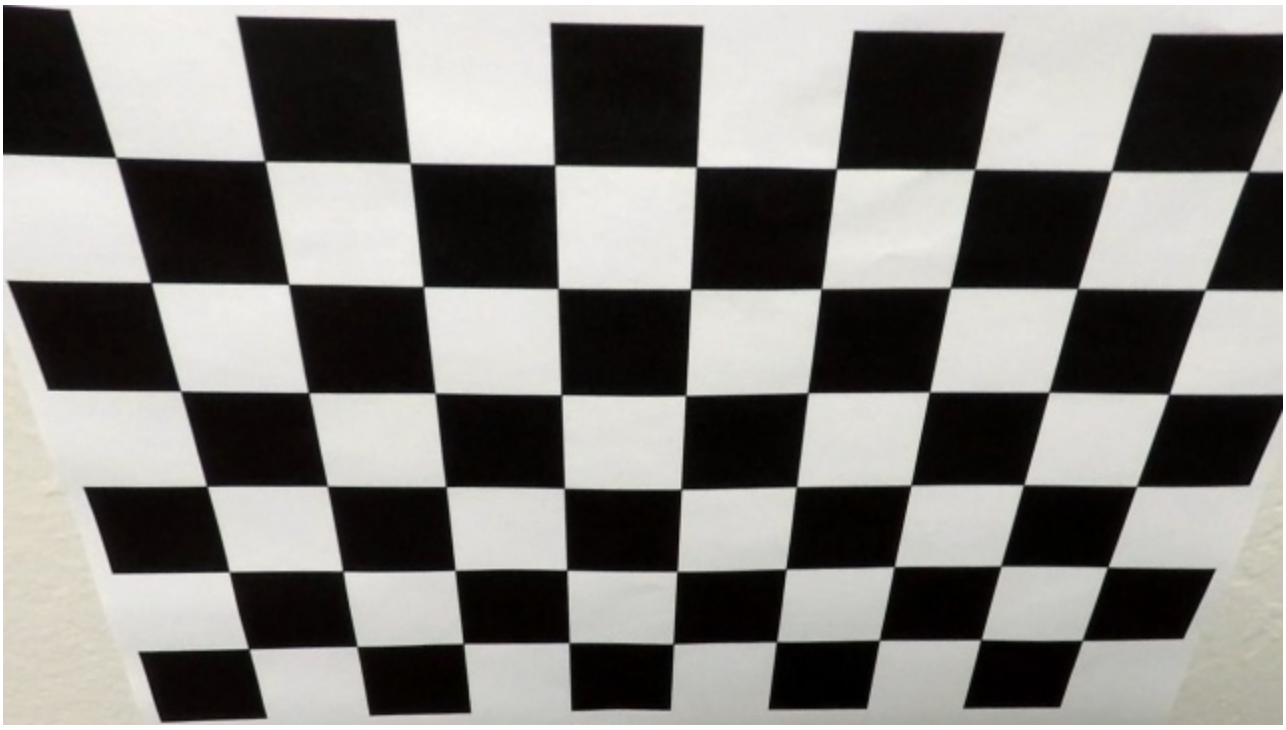
To show calibration works one of the checkerboard images is shown below. The first image is the original image, the second image is the image obtained after undistorting it using the calibration matrix and distortion coefficients.

Original checkerboard image:

, including references to various Python classes and functions implemented for each step.



Undistorted checkerboard image:



Next, one of the test images provided with this project is undistorted. The two images below show the original and undistorted version of test image `test5.jpg` respectively.
Original `test5.jpg` image:

Undistorted `test5.jpg` image:

Step 3: Threshold binary image

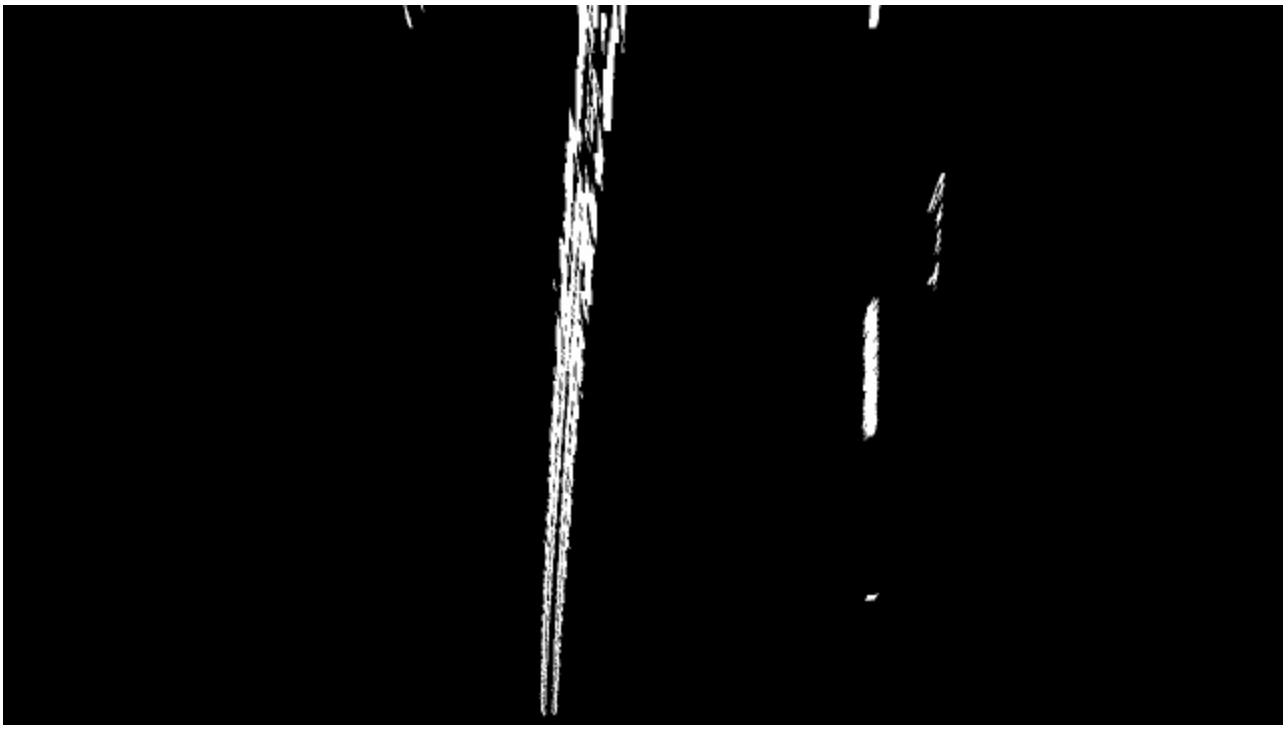
In order to detect lane lines in an image a binary threshold image is used. Any white image pixels are likely to be part of one of the two lane lines. All code required to create the binary threshold image is contained in function `threshold_image` in class `BinaryThresholder`. That class uses a combination of Sobel-x and Sobel-y operators, magnitude of the gradient, direction of the gradient, the S-channel of the image (after converting it from BGR to HLS), the U and V channels (after converting the image to YUV) and masks created to detect yellow pixles (for the outer left lane) and to detect lane lines covered by shadows, to create the binary image.
The various thresholds were determined using trial-and-error in combination, the test images provided with this project and the diagnostics view Once a proper binary threshold image is found, any detected pixels that are not inside a trapezoidal region of interest directly in front of the car are excluded. This helps avoid false positives.

The binary threshold image for test image `test5.jpg` is shown below.

Step 4: Perspective transform

In order to view the lane from above a bird's-eye view transform is applied to the binary threshold image. All code required to create the bird's-eye view transform is contained in the `__init__` function of class `PerspectiveTransformer`. The correct source and destination points required by OpenCV's `getPerspectiveTransform` function were determined using the `straight_lines1.jpg` image. The points were manually picked in such a way that the two lane lines appear as two straight and parallel lines in the bird's eye view image.
The perspective transform is only calculated once rather than for each image/video frame because for this project the assumption that the road is always flat is valid.

The image below shows the bird's-eye view of the binary threshold image shown above. As expected, it shows that the lane is curving slightly to the right.



Step 5: Detect lane lines

All lane detection code is contained in the `__detect_lane` and `__sliding_window` functions in the `LaneFinder` class. The lane lines are detected in two steps:

1. function `__detect_lane`: First a histogram is applied to the bottom half of the bird's-eye view of the binary threshold image. After splitting the image in half, the points in both halves with the highest number of pixels are assumed to be the starting points of the left and right lane line respectively
2. function `__sliding_window`: Using the two detected starting points for both lane lines, a sliding window approach is used to detect peaks in small areas around each starting point. To detect the peaks, `scipy.signal`'s `find_peaks_cwt` function is used.

After using the sliding window to detect lane line points starting at the bottom until the top of the image, second degree polynomials are fitted to both lane lines using numpy's `np.polyfit` function. The two polynomials approximate the left and right lane lines respectively. To avoid sudden jumps, any lane lines that are clearly not valid ones are rejected (e.g. if the left and right lanes are not parallel, which they should always be, or if the two lane lines are too close or too far apart). Also, the lane lines detected for the current frame are smoothed using previous' frames lane lines.

Step 6: Determine curvature and vehicle position

The curvature of a lane line is calculated in the `calc_curvature` function in the `Line` class. It uses the polynomial that approximates the lane line and converts detected lane pixels to meters first before calculating the curvature.
The position of the car is calculated by subtracting the center of the image/video frame, which is assumed to be equal to the center of the car, from the lane's center point. The lane's center point is calculated by averaging the left and right lane's fitted polynomial. Calculating the position of the car is done in `LaneFinder`'s `__plot_lane` function.

Step 7 and 8: Plot detected lane and additional information

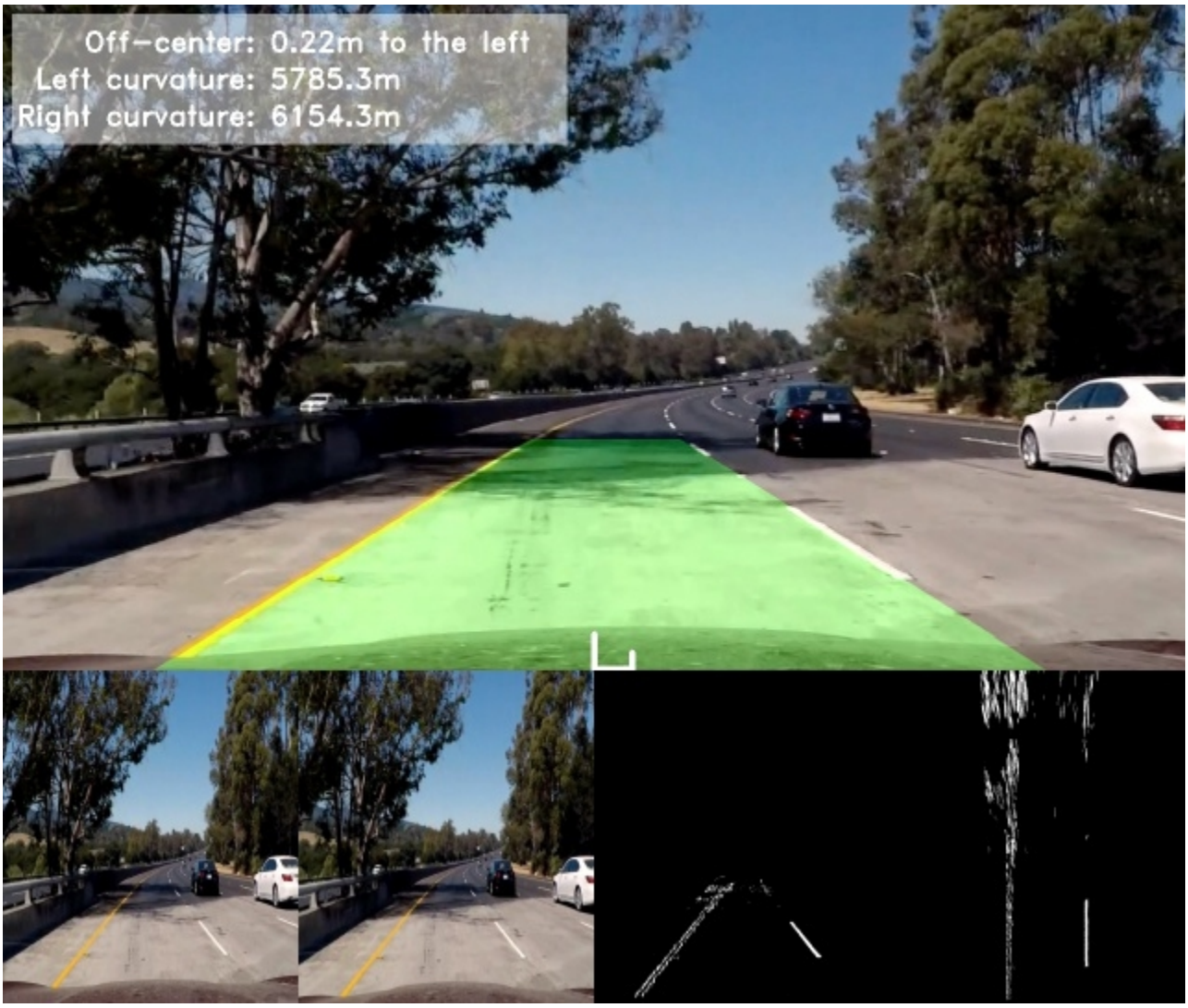
Function `__plot_lane` in the `LaneFinder` class is used to warp the detected lane lines back onto the original image and plot the detected lane using a filled polygon. It also plots additional information in the top left corner and at bottom of the image/video frame:

- Curvature and position of the car relative to the lane center is plotted in the top left corner
- The position of the car is indicated at the bottom with a long vertical white line. The detected lane center is indicated using a short white line. In the project video the car stays relatively close to the center of the lane, though it is mostly positioned somewhat to the left of the lane's center.

The image below shows the result for test image `test5.jpg`:

Diagnostic view

To simplify pipeline finetuning, a diagnostic view showing a number of images taken after various pipeline steps in a single image was used. This view is implemented in function `plot_diagnostics` in class `LaneFinder`.
An example is shown in the image below. It shows the detected lane and additional information in the top part of the image. Along the bottom it shows the orignal image, undistorted image, binary threshold image and bird's-eye view of the binary threshold image respectively.



Project video

Exactly the same pipeline is applied to videos. The pipeline is implemented in `LaneFinder`'s `process_video_frame` function. That function is identical to `process_image`, except that the latter includes additional code to create and save a number of images used throughout this file

Project code

All code for this project is implemented in the following `.py` files:

To run this project simply execute `Script_advanced-lane-finding-main.py`. Note that it is currently setup to produce all images included in the `output_images` folder and to process the project video `project_video.mp4`

- `Script_advanced_lane_finding_main.py`: this is the main script for this project that ties everything together
- `Script_CameraCalibrator.py`: this class calibrates the camera
- `Script_PerspectiveTransformer.py`: this class calculates the perspective transform
- `Script_LaneFinder.py`: all lane detection code is contained in this class
- `Script_BinaryThresholder.py`: this class encapsulates all code used to create the binary threshold image
- `Script_Line.py`: this class encapsulates a single lane line (i.e. either the left or right lane line)

.