CarND-Advanced-Lane-Lines

1. Pipeline

The pipeline of advanced lane line detection process consists of the following steps:

1. Camera calibration and estimation of the perspective transform (performed only once)
2. Camera distortion correction
3. Binary mask estimation
4. Perspective distortion correction
5. Line detection (with prediction filters)
6. Post-processing

Each step will be treated in detail in the following sections.

2. Distortion correction

In this project, there are two sources of distortion that should be taken care of. First, we have to correct the distortion introduced by the camera. Second, in order to measure the curvature of the lines, we have to correct the perspective distortion the effect of which is that further object appear smaller in the scene.

2.1 Camera calibration

In order to correct the distortion introduced by the camera we have to find its parameters. This can be conveniently done by taking picture of a chessboard pattern and then matching it to the ground truth. In this project, the calibration process is performed by the *calibrate_camera()* function from *distortion.py* file. An example of camera distortion correction is shown in Fig. 1.

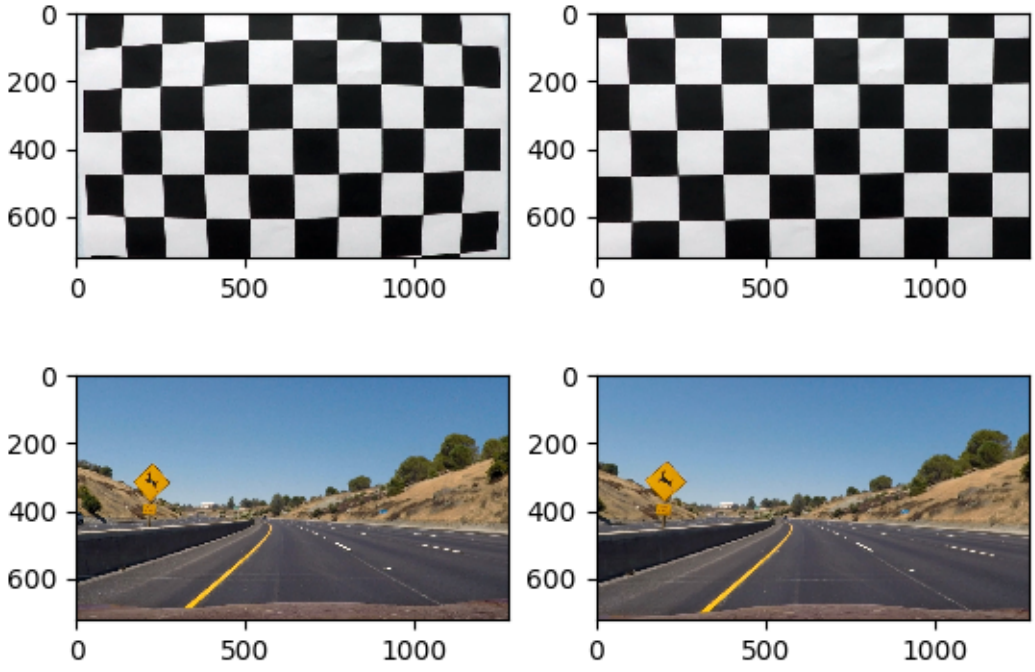

Figure 1: Original image (left) and with camera distortion correction applied (right).

2.2 Perspective distortion

This type of distortion is caused by the fact that objects that are closer to the camera appear bigger in the captured scene. In order to measure the attributes of the detected lines (e.g. curvature), we should have an aerial view of the road ahead of us (without perspective distortion). This is can be done by *ad-hoc* mapping of a source image to a desired shape. Figure 2 shows the pair of images that has been used to estimate the perspective distortion matrix.

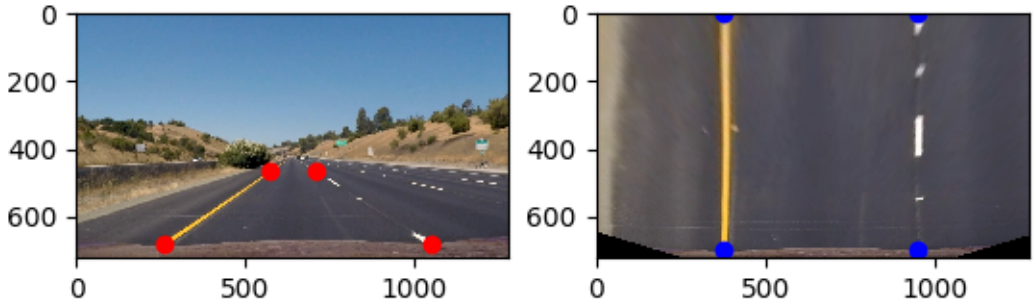

Figure 2: Perspective distortion correction. Original scene points (left, red dots) are transformed to blue points (right).

3. Binary mask

In order to estimate line parameters, first we have identify the pixels that are likely to belong to a lane line. As seen in this course, there are several ways to detect such pixels. In this project, we have opted for a combined approach consisting of the following two detectors:

- White lines detector which detects regions that are whiter than the surrounding area. The whiteness metric is defined as the average value of the three RGB channels. This operation is performed by *adaptive_white_thresh()* method from *thresholding.py*.
- Yellow lines detector which detects regions that are yellower than the surrounding are. The yellowness metric is defined as the ratio of the R and G channels (whose combination produces the metameric yellow colour) with respect to the B channel. The higher the ratio, the yellower the colour. This operation is performed by *adaptive_yellow_thresh()* method from *thresholding.py*.

An example of binary mask generation is shown in Fig. 3.

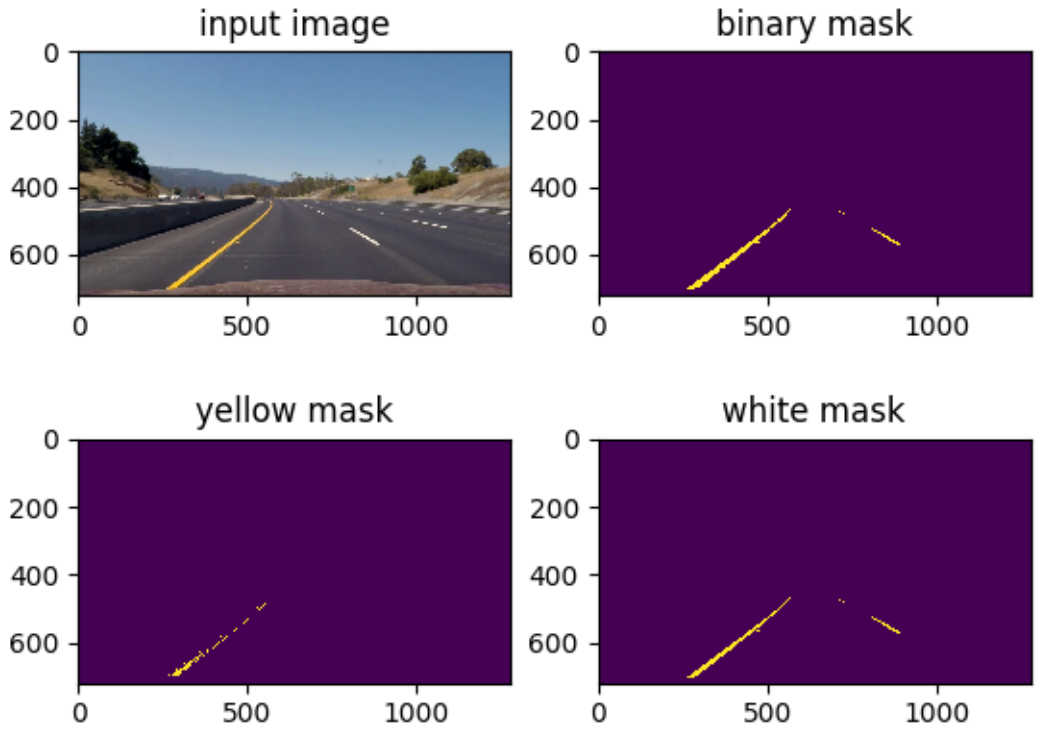

Figure 3: Example of binary mask generation. The resulting mask is the combination of yellow and white masks.

Both detectors are adaptive, i.e., the yellowness and whiteness threshold are measured according to the current scene. In particular, the current road surface (i.e., the image region that we have considered when computing the perspective distortion correction matrix) is considered so the whiteness and yellowness of each pixel is compared to the mean whiteness and yellowness values of the road. This approach is illustrated in Fig. 4. The objective is that the thresholds are invariant to scene illumination (sun vs. shadow).

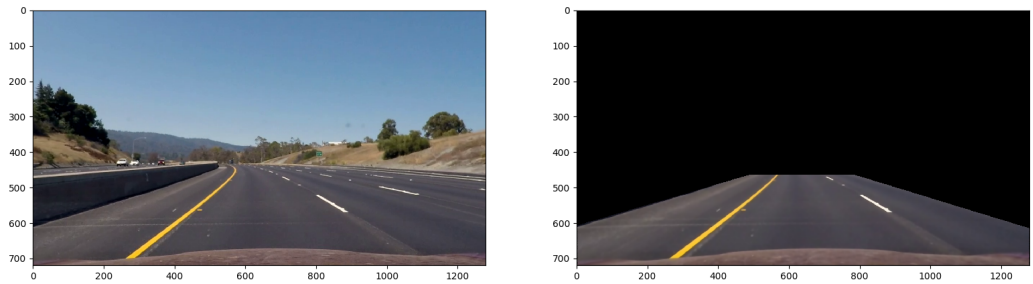

Figure 4: Original image (left) and part of the image considered when setting the region for perspective distortion correction estimation.

In order to reduce the effect of noise, the source image (after camera distortion correction) is filtered by a (relatively) strong bilateral filter so the visual noise is reduced without blurring the edges. The example of using such filter is shown in Fig. 5.

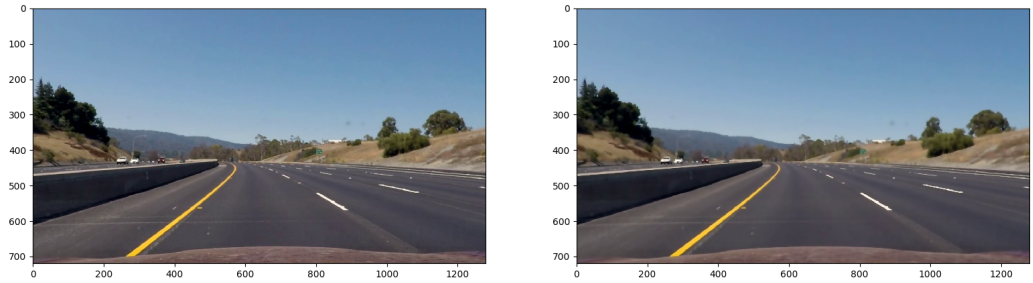

Figure 5: Example of image preprocessing. Original image (left) and the same image after applying a bilateral filter (right).

4. Line detection

Given the binary mask (after perspective distortion correction), the lines are detected "slice-wise" (using horizontal slices). In each slice, a sliding window (kernel) is used to detect the region with the higher amount of edge pixels. This search is based on 3 main assumptions (see *get_center()* from *detection.py* for more detail):

- Lane lines should produce a very high kernel response.
- Lane lines are vertically shaped objects.
- Lane lines are (relatively) narrow so they are not likely to fill the whole kernel.

For each slice, we find the corresponding centre of the left and right lines. This centre is computed as the centre of mass of the "hot" pixels (from the binary map) within the kernel. These centres are then used to estimate the polynomial fit that represents the two lane lines. The centers are computed using the *get_centers()* function from *detection.py*. The slicing procedure is illustrated in Fig 6.

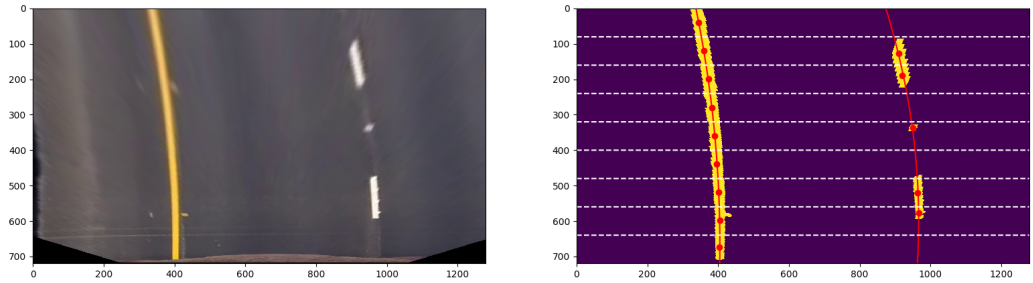

Figure 6: Original undistorted road image and the corresponding binary mask divided into (horizontal) slices. The red dost correspond to the detected line position in each slice. The polynomial (second order) fit is represented by red line.

Note: The base line search based on the edge histogram (as shown in the course) is just a particular case of our approach (with kernel width equal to one) so we do not implement a special method for such purpose.

5. Prediction

During the line search, we have implemented two prediction (or look-ahead) filters whose objective is twofold:

- To speed up the line search.
- To make the output more robust in case the detection fails.

The two filters are detailed in the following subsections.

5.1. Intra-frame filter

When searching for the lines slice-wise, the line position variation within the same frame should be smooth. This is implemented in *get_centers()* function from *detection.py*. If a line is detected within the first mask slice, the next slice is searched only around the position of the previously detected line (*reference_left* and *reference_right* variables in *get_centers()*). Here we implicitly assume that bottom line estimations are more reliable than top ones. We consider this a reasonable assumption since bottom objects appear bigger and are less blurred by perspective distortion correction.

5.2. Inter-frame filter

The detected lines (or line centres) between two consecutive frames should be similar. Therefore, the line search in the current frame is carried out in the vicinity of the line detected in the previous frame (if available). This filtering is performed by passing the *inter_reference* parameter to *get_centers()*. Note that if i*nter_reference* is provided it overrides the intra-frame filter. This is based on the assumption that temporal correlation between two consecutive frames is much higher than spatial pixel correlation within the same frame. So, in other words, the prediction step uses spatial information (intra-frame filtering) only it the temporal information is not available.

6. Post-processing

Both line curvature and vehicle offset are computed using the same procedure as taught in the course. To stabilise the output, lines position and line attributes are averaged using the last *3* frames (the best line fit actually uses a moving average). To keep track of the detected parameters, an instance of the Line() class (see lines.py) is used.

7. Discussion

A good and robust lane line detector should be invariant to scene illumination (sun vs. shadow) and to the road surface. I have tried to take that into account by applying an adaptive thresholding and whiteness and yellowness detectors but, obviously, this solution is far from being ideal. This is actually (in my opinion) the most critical step and it would require the combination of various detectors (in an adaptive way) in order to get a robust estimation of line pixels.

Personally, I am not a fan of hard thresholding and I would investigate the possibility of applying weights to the pixels that are believed to belong to a line (soft data). In fact, in the current implementation, all pixels that are considered to compose a line are treated equally (when, e.g., estimating the best polynomial fit). We could actually design a (relatively simple) metric as a confidence of a pixel belonging to a line.

There is *a priori* information that, in the current implementation, we are not taking into account at all. In fact, the two (detected) lines must be parallel which implies constant separation and identical curvature.

Noise in the binary mask (in its upper part) can be hugely amplified by the perspective distortion correction. This unequal treatment of noisy pixels should be taken into account during the detection process.

I believe image pre-processing is very important for a robust result. I have used a (relatively) strong bilateral filter to get rid of noise while preserving the shape of the lines without adding blurring. Perhaps an even more aggressive bilateral filter would be enhancing but this is a subject to be studied.