Self-driving car Nanodegree (Term 1). Project 4: Advanced Lane Finding

Ruggero Vasile

June 22, 2017

Abstract

Report for the fourth project in the self-driving car nanodegree program, term 1. The project consists in building a pipeline for line detection that works in a variety of visual conditions, for single frames and video streams. To assess accuracy and performance, the pipeline is tested on six example images and on a project video.

1 Introduction

In this report I collect all the relevant steps of the process towards the pipeline for the advanced line finding project. The pipeline consists of the following steps:

- *Camera calibration.* Assuming that the images and the videos are taken by the same camera, this step is required to correct the visual input from possible radial or tangential distortion of the camera. Each image, or video frame, before being processed, will have to pass through a function that removes the distortion.
- *Perspective transformation.* In order to find the correct shape of the lines on the road, we need to correct the image for perspective. This allows to project the street plane from a bird view and therefore obtain the correct equations for the road lines. This step, as it is here performed, assumes that the road is flat. When this is not the case, the algorithm fails and it would be required to approach the problem using more sophisticated pipelines.
- *Creation of thresholded images.* To isolate the lines from other objects in the image we apply different thresholding techniques. Among them we use here gradient thresholding, and color thresholding. The main challenge for this step is to choose the correct thresholding techniques or combination of techniques that, in the current visual conditions, will provide a good enough thresholded image containing the desired lines.
- *Line identification.* Once the thresholded image is obtained we need to identify the lines in it. We assume that two lines are to be identified for each frame. The outputs of this step are the equations of the two road lines in the perspective transformed images.
- *Back to street perspective.* In this step we undo the perspective transformation and obtain the lines as seen by the driver. These lines will be then superimposed on the original images to assess the quality of the pipeline.

The full submission consists of this guiding report, python scripts files (*classes.py*, *utils.py* and *preprocess.py*) which contain the relevant classes and functions used to produce the results, and an ipython notebook which generates the images for the report. Together with these documents there can be found the test file images and the video for the project challenge requested.

2 Camera Calibration

In this section I describe the camera calibration step. I use a standard tools based on chessboard images. Udacity provides 20 chessboard calibration images taken with the same camera mounted on the car. Using the openCV functions *cv2.findchessboardpoints* and *cv2.calibratecamera* on the calibration images, I extract the distortion coefficients. These coefficients are then used to undistort any upcoming image through the openCV *cv2.undistort* function. In Fig. 1 I show the effect of distortion removal on distorted images.

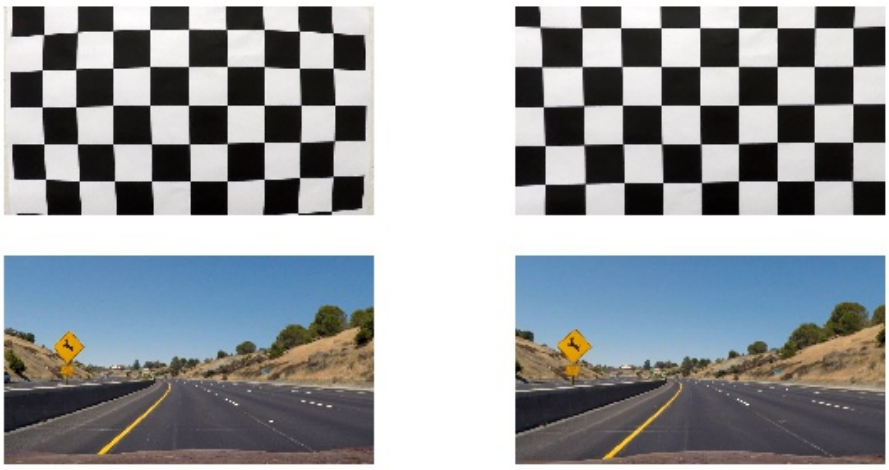

Figure 1. Effect of camera calibration distortion removal. (Above) On the left the original image affected by evident radial distortion and, on the right, the undistorted result. (Below) Same comparison for a image taken by the same camera on the road.

3 Perspective Transformation

The next step consists in extracting the correct perspective transformation matrix from one of the two sample images provided by Udacity. The perspective transformation can be calculated by choosing 4 points in the original image that are supposed to be vertices of a rectangle when viewed from a frontal perspective. The transformation, using the openCV functions *cv2.getPerspectiveTransform* to evaluate the transformation matrix and *cv2.warpPerspective* to apply the transformation, warps the points in the new perspective. In order to choose correctly the initial points we need to choose a picture from which points in a rectangle are easily identifiable. A road picture with straight lines is exactly what we need. The effect of transformation can be seen in Fig. 2 where straight lines appearing non straight in the original picture are warped correctly to straight lines in the transformed picture. Once we apply the transformation to another image, for instance, where a car is undertaking a curve, we can see how, from the bird view point, we are able to identify the real curvature of the road lines (see Fig. 3).

Figure 2. (Left) The original image with, superimposed, a trapezoidal region (red line) which is supposed to be rectangular in 2D. (Right) The transformed image from a bird view scenario where the trapezoidal region is correctly warped into a rectangular region. The road lines appear correctly to be straight.

Figure 3. (Left) The original image with a car undertaking a curve. (Right) The transformed image from a bird view scenario where we can clearly see that the lines are not straight.

4 Thresholded Images

Together with the previously described perspective transformation, the step described in this section, constitutes the essence of correct road lines identification. The protocol here described is one of the possible solutions to the problem, and it has been proved to be of sufficient rigour for the present challenge.

I use here the concept of gradient and color thresholding, a set of techniques which aims at detecting color or intensity changes in images.

4.1 Gradient Thresholding

Such techniques is applied on RGB images by first performing a greyscale transformation. Once the images have one single intensity channel, the Sobel operators for gradient evaluation can be applied and the intensity gradient in one direction computed. Different intensity gradients are obtained by changing the kernel size of the Sobel operators. I denote with S_x and S_y the gradients obtained using the sobel-x and the sobel-y operators respectively. Given S_x and S_y we can evaluate other quantities like:

- Absolute gradient in x ($|S_x|$) and y ($|S_y|$) directions.
- Magnitude gradient obtained by squaring the $M = \sqrt{S_x^2 + S_y^2}$
- Direction gradient: $D = \arctan S_y/S_x$

Once the gradients are evaluated we can create a masked image by selecting only those pixels for which the correspondent gradient lies within a certain thresholding interval. Depending on the values of the thresholds, different results can be obtained. In Fig. 4 I show the effect masked pictures obtained by evaluating four different intensity gradients and thresholding them. The values of the thresholding intervals are reported in the figure captions and have been manually optimized trying to reach the best possible results. We can see that the absolute gradient x-component achieves the best results since it concentrate on finding mostly vertical edges. Despite so it fails in recognising the yellow line in high lightness because the line pixels do not differ much with those of the adjacent street in grayscale. The other gradients instead are even more noisy and therefore will be in the following discarded.

4.2 Color thresholding

While gradient thresholding is used to find discontinuities using pixel intensities in gray scale images, color thresholding attempts to use information on the color of the objects to recognise. This is particularly well suited in our case since we are interested in detecting lines which, in the work cases, are mostly yellow and white. It could be therefore of use to threshold the images based on different channel attributes. For doing this I first consider the HLS color channel representation and threshold on the values of the different channels to select specific interesting colors. Specifically:

- The yellow color in HLS color space can be identified with channels values between $[15, 50, 100]$ and $[25, 200, 255]$
- The white color instead with channels intensity values between $[0, 200, 0]$ and $[255, 255, 255]$.

Given the above rules it is possible to threshold the images and generate a binary mask accordingly. In Fig. 5 we can see the two results that achieve good detection, respectively, of the yellow line and of part of the white line.

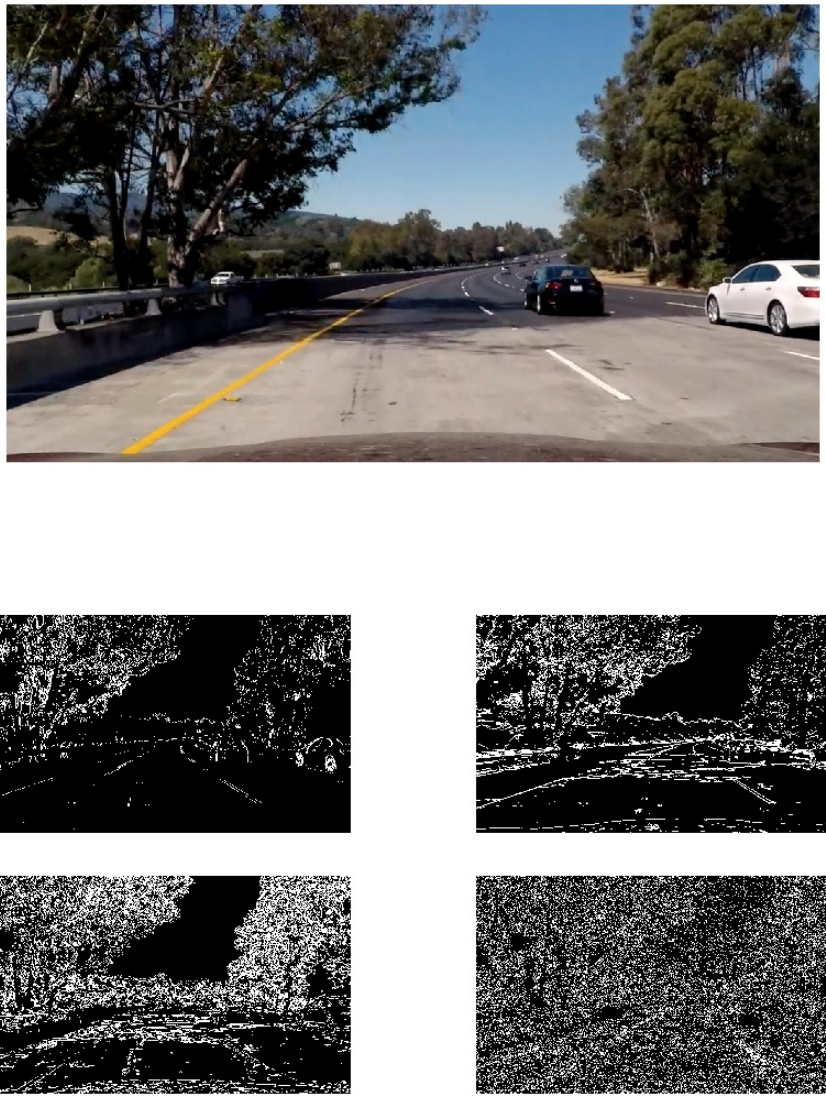

Figure 4. Above: The original image. Panel Below: (upper left) the absolute gradient x-component with threshold interval $(25, 255)$. (upper right) The absolute gradient y-component with threshold interval $(20, 150)$. (lower left) The magnitude gradient with threshold interval $(30, 100)$. (lower right) The direction gradient with threshold interval $(0.8, 1.3)$.

Figure 5. (Above) Original image. (Below left) Yellow color threshold. (Below right) White color threshold

4.3 Combining Masks

We have seen that a different masks are able to recognise different part of the lines in a image, but alone they would work imperfectly. It makes therefore sense to combine some of them in order to obtain a more complete mask that would be able to identify the whole road lines. Due to the previous results I opt for a combination of absolute gradient in x direction, white color mask and yellow color mask. In Fig. 6 we see the result of such action, where the combination is able to identify the yellow parts of the line in both shadow and not shadow conditions as well as the white right discontinuous line.

5 Line identification

Line identification is performed by combining the two techniques described in the previous section, thresholding for determining the line segments positions, and perspective transformation which allows to map the line in a bird view point (see Fig. 7). Given the warped figures we can work towards the line identification. As mentioned before, the single image line identification and the line tracking in the video are subjected to different technical constraints. I will describe them separately in the following sections. For both cases the code implemented makes uses of three fundamental object classes: a line class which contains info about a single line of the road, e.g. its curvature, a lane classes, which collects the two lines of the road, and a pipeline class. The pipeline class has a *run* method which takes a frame and returns the same frame with superimposed lane, lines and other information required by the project. The technical descriptions of the classes member variables and methods can be found directly in the code files. In this report I will just outline the functionality of the pipeline and the results.

5.1 Identification of road lines in a single frame

Here are the steps followed towards line identification in a single frame.

- Given an input frame, I first remove the distortion, apply the combined mask and perspective transform.
- Given the warped image I assume there are two lines, a left line and a right line. The first step is to identify the initial line points, i.e. for a given line a pair of coordinates through the line passes. As suggested in the course I build an histogram that counts the number of pixel in the warped image with any value of the y coordinate and a given x (see Fig. 8). The positions of the two main peaks in the histogram x_l and x_r approximate the x-coordinate of the two lines of the road.
- Once we have the approximate location of the lines I build two sliding windows, one for each line, to track the lines along the y-direction. I fix to 100×100 the dimension of windows, centered initially, respectively in positions $(x_l, 670)$ and $(x_r, 670)$ (670 equals the pixel image size along y minus half of the window size).
- Each window selects part of the image where line pixels are present. Therefore for each line I extract the coordinates of these pixels, save them and move up the sliding window. The new center, for the left line case, will be $(x_l - m, 570)$, where m is the mean line x-coordinate for the pixels encompassed in the first sliding window.
- I iterate the previous operation until I reach the upper part of the line. All the pixels encompassed in the sliding windows will be identified as line pixels (see lower panels of Fig. 8).
- Once having all the pixels belonging to the left and right lines, I perform two independent quadratic fits and extract the coefficients of the lines (the equation is of the form $x = Ay^2 + By + C$).

Figure 6. (Above) Original image. (Below left). Full mask after application of absolute gradient in x-direction mask, and yellow and white masks.

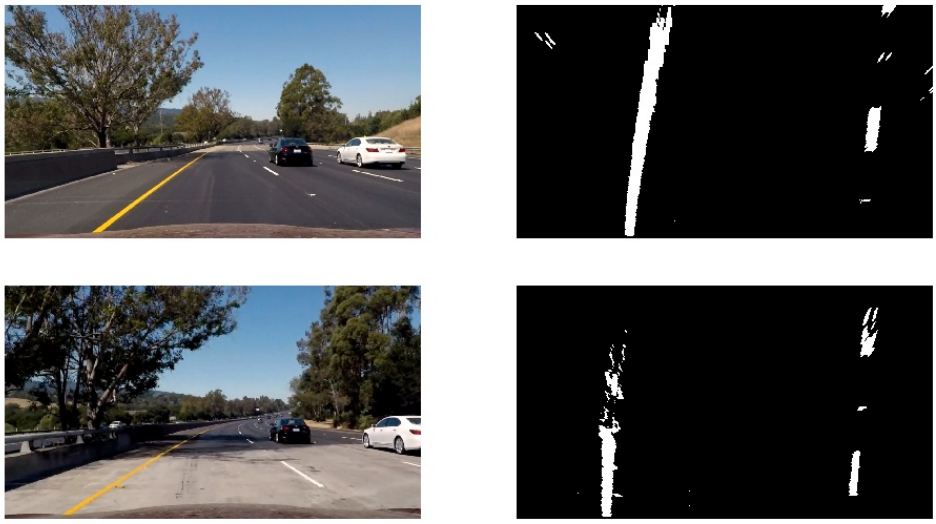

Figure 7. Pair of original images (left) and correspondent image after combined thresholding and perspective transformations (right).

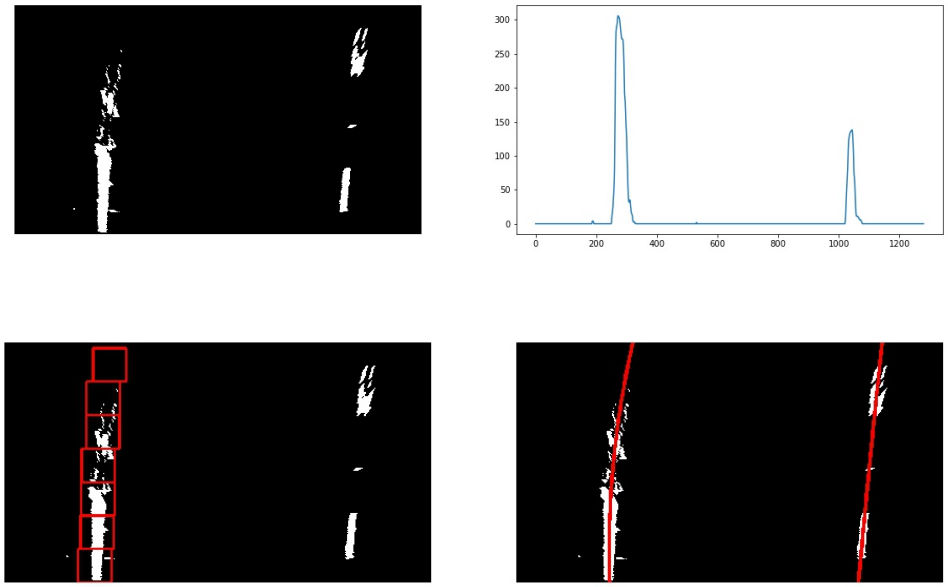

Figure 8. (Above). On the left the masked image. On the right the left and right peaks show the approximate position of the left and right road lines. (Below) On the left the sliding windows that follow the left line for all the frame. On the right the final identifications of the two lane lines.

5.2 Identification of road lines in video frames: Line Tracking

From two subsequent video frames we can expect the lines not to change appreciably. For this reason once we identify the lines in one video frame, we can use this information for the identification of the lines in the subsequent frame. For each line of the previous frame, in fact, one can select a bounding area which represent the part of the subsequent frame to search in for the new lines. This reduces the search time and allows for faster identification of the lines in the next frames.

6 Back to original perspective: Results

I have built a pipeline that starting from a single frame, applies various transformations that bring us to identify road lane lines in a variety of conditions. We are now ready to go back to the original perspective and plot the lane, the lines and other information on the test images and on the video frames.

6.1 Lane curvature and car position

Among the informations requested by the project rubric there are:

- The lane curvature: this is defined as the radius of an imaginary circle tangent to lines of the road at the position of the car. Due to error/detection measurements, the curvature detected may be different between the two lines of the road, therefore I will calculate and report them separately. Given the line equation $x = Ay^2 + By + C$ the radius can be calculated as:

$$R_c(y) = \frac{(1 + (2Ay + B)^2)^{3/2}}{2|A|} \tag{1}$$

  and depends on the location y on the trajectory. Since until this moment I have worked with distances expressed in terms of number of pixels, to obtain the radius in meters we need first to convert the pixel distance into meters. Due to the particular image perspective, the pixel distance in meters the y and x directions are different. For conversion I use here the factors provided by Udacity, i.e. 1 pixel in the y direction corresponds to 0.041 meters while in the x direction to 0.0053 meters.

- The relative position of the car in respect to the center of the lane. This is defined as:

$$X_c(y) = \frac{L_x(y) + R_x(y)}{2} - 640 \tag{2}$$

  where X_c represents the distance in pixel and 640 is half of the image horizontal pixel size. To convert to meters we simply multiply the result for the conversion rate in the x direction.

6.2 Results on test images

In Fig. 9 I provide the results of the above pipeline on the test images given. For each of them the original image is superimposed with the lines detected, a shadowed space between the lines, the estimated curvatures and the estimated position of the car relative to the center of the lane.

6.3 Result on the testing video

For the results on the test video, please refer to the appropriate file.

Figure 9. The result of the pipeline on the 6 test images.

7 Conclusions

I found this project very challenging but also very useful to understand some of the issues that a self driving car engineer needs to take care of.

The pipeline I created achieves very good results on the test images and on the simple video, while it has some issues on the challenge and hard videos. More sophisticated approaches are then required, for instance:

- When the roads are slower and of different steepness, the perspective transformation needs to be optimized by choosing a smaller portion of the road.
- For more variable light conditions, lines can be more difficult to be identified using the combined threshold described here, and more complicated threshold combinations will have to be taken into account.
- When some vehicle blocks the vision of the lines, one would need to identify such situations and remove that part of the image from the line identification.