ADVANCED LANE FINDING PROJECT

P2 Writeup
Daniel Vizcaya

Reflection

1. Camera calibration

    The code for this section can be observed in the first cells of the notebook project_2.ipynb. First, all calibration images were loaded. Each image was converted to grayscale and then the corners of the chessboard were calculated using the findChessboardCorners cv2 functions. The points of the corners of all images were saved in the imgpoints variable. It is worth mentioning that the function did not found the corners for 3 images. Nevertheless, this should not affect the performance, given the number of calibration images.

    After that, the variables created were used to calibrate the camera using the cv2 function calibrateCamera.

    A corners_unwarp function was created to evaluate the performance of the distortion over the calibration images. The result of this exercise was satisfactory, and some examples can be found below.

2. Pipeline

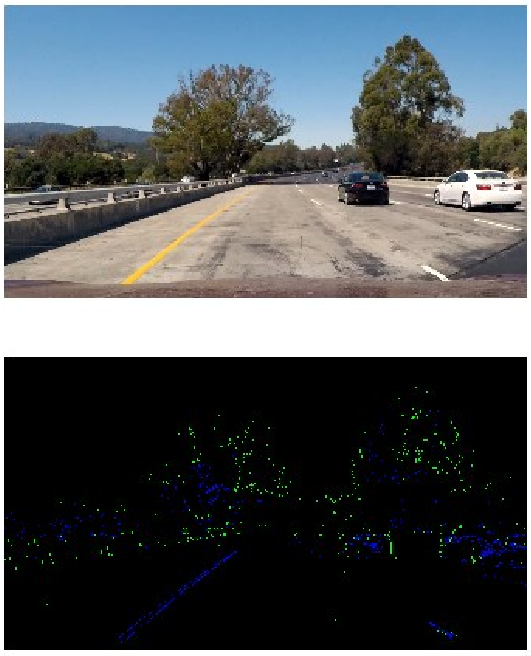
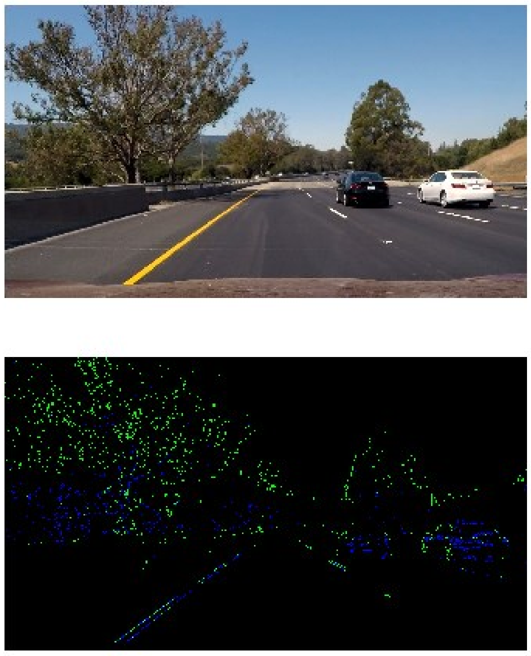
    The pipeline that process all images and video frames was developed under the function image_pipeline. Inside this pipeline, other functions were called as will be explained below.

    - Distortion corrected image

      The first step of the pipeline is to correct the distortion of the image, using the parameters found when calibrating the camera. Following can be found some examples of images before and after the distortion correction.

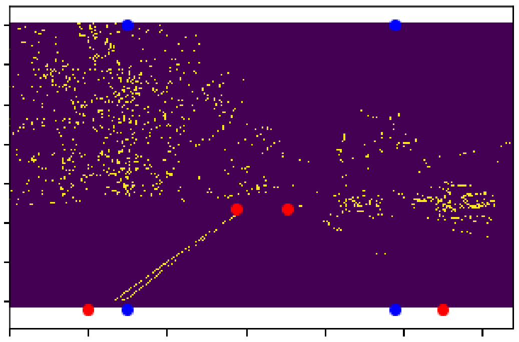
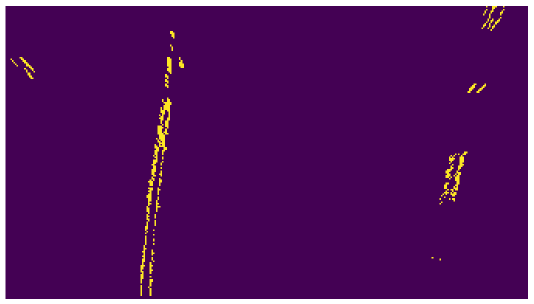
- Color and gradients thresholds

  The operations for color and gradients thresholds can be found on the function called transform_image. For the color transformation, the image was converted to HLS, the s channel was selected, and the points were keep if the values were over 170, up to 255. The Sobel in x was also evaluated. The thresholds in this case were 20 and 100.

- Perspective transformation

  The perspective transformation of an image was performed in the image_pipeline function. The source points were defined as [[(1280/2)*.9, (720/2)*1.3], [(1280/2)*1.1, (720/2)*1.3], [1100,720], [200,720]]. In the following example images, the red dots represent the source and the blue points the destination.

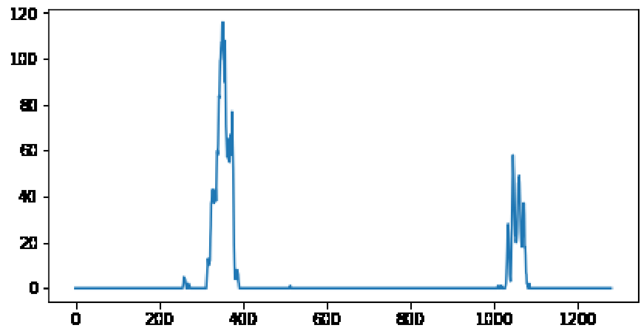
- Lane-line pixels and polynomials

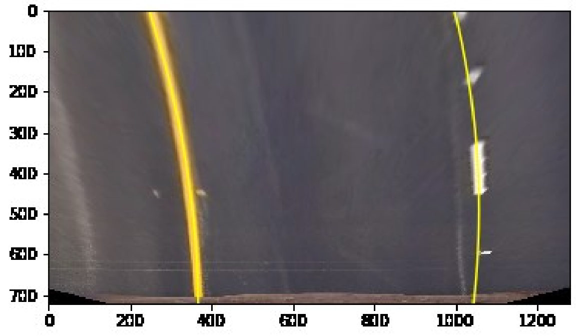
    In order to identify the lane-lines polynomials, the first thing was to calculate the histogram of the half bottom section of the image. This was done on the image_pipeline function. After that, the polynomials were calculated on the get_polys function.

    The get_polys function receives as input the image, the histogram and the polynomials of a previous frame (optional). An option for debugging the images is also available. The process is different depending on whether there are polynomials of previous frames, or not.

    In the case there are no previous polynomials, the first thing was to find the points that were located in a window defined by the half bottom of the image, and the greater point on the histogram +/- the margin (for both left and right). After that, a first polynomial was fitted. Then the process was repeated, increasing the vertical scope by the window height, and the horizontal scope was defined by the polynomial curve +/- the margin.

    When there was a previous frame, there was no loop but instead, the whole image was checked at once, where the pixel search was limited to the previous polynomial +/- the margin. In order to avoid drastic changes, the x value of the polynomial at y=0 was compared with the one from the frame before. In case the change of the x position was over the margin, a correction was made by fitting the polynomial with both the current and the old pixels. This rule was very helpful to avoid problems in some frames were the color and gradient thresholds included pixels that were not the actual line.





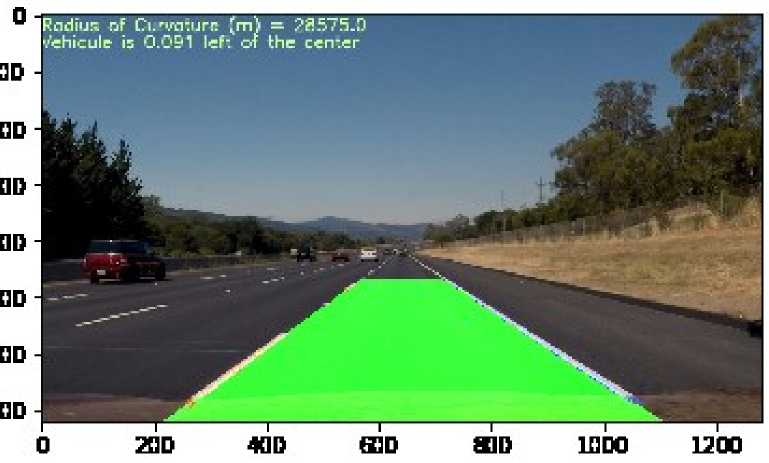
- Curvature and car offset

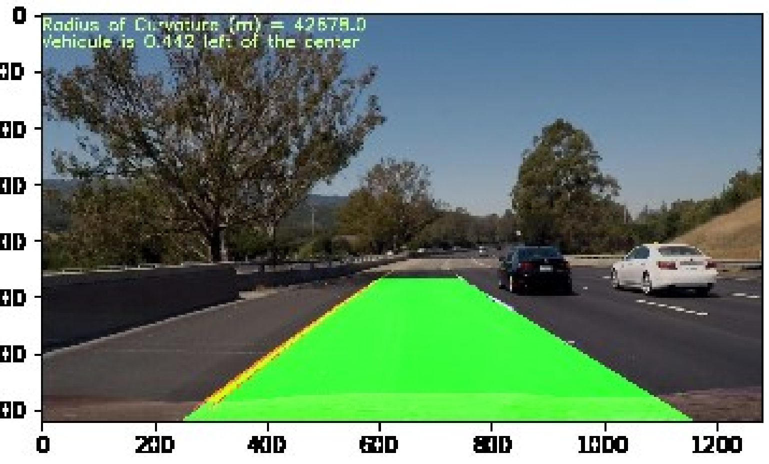
  The curvature of the polynomial was calculated using the measure_curvature_real function. This function received the pixels of the images for both lines. Those pixels were converted to meters considering 30 meters per 720 pixels in the y dimension and 3.7 m per 700 pixels in the x dimension. After the pixels were converted, a new polynomial was fitted, and the curvature was calculated with the same equation presented on the lessons.

  The car offset was calculated on the get_car_offset function. This function calculates the x value for y = max(y) for both lines (based on polynomials). After that, the distance between the center of the image and each of the lines lower point was measured and converted to meters.

- Final result

  The following are examples from the test images.


Radius of Curvature (m) = 28575.0
Vehicule is 0.091 left of the center


Radius of Curvature (m) = 42678.0
Vehicule is 0.442 left of the center

3. Pipeline (Video)

The pipeline used for the video was the same one described above. The only difference was that for this case, a class called ProcessVideo was created. The purpose of this was to be able to save the polynomials information between frames.

The final video can be found under the output_videos/project_video.mp4.

4. Discussion

The first challenge faced when developing the project was that even though the thresholds for color and gradients were stablished after tuning several images, there were some frames on the video were the result was not as good. This affected the result when using only the window search technique and was the reason of applying the option of searching based on the polynomial of the previous frame.

Another issue was that sometimes, wrong pixels were detected on the upper section of the image, and that completely changed the polynomial behavior, so a restriction was made as explained before. This solution also helps when there were not many pixels on the lower part of the image.

The pipeline is still likely to fail when the first frame is a curve or in a place with a lot of contrast as the initial polynomial could be detected incorrectly. It can also fail when a series of frames completely change the behavior of the polynomial as no reset technique was created because it was not necessary for the video project.

In order to make it more robust, I would add a reset technique and more validations that verify that the result of the polynomial makes sense.