Pablo Sauras Perez Reflection.

Term 1 – Project 4 – Advanced Lane Finding

1. Files Submitted

This project submission has the following files:

- **P4_AdvanceLaneFinding.ipyng** the code of the project.
- **output_images/** is a folder that contains the outputs of the project (images and video).
- **project_video_result.mp4** is the output video of this project.
- **PabloSauras_Reflection_P4.pdf** is this report.

Camera Calibration

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is based on the code of the Ipython notebook located in "./examples/example.ipynb".

Cell 2: I determine the object points (3D real world points) (`objpoints`) and image points (2D points) (`imgpoints`) using the function `cv2.findChessboardCorners`. Once the corners are detected successfully, `objpoints` and `imgpoints` can be filled. The outputs images are in **output_images/calibration_corners_found*.jpg**

Cell 3: `objpoints` and `imgpoint` are then used to compute the camera calibration and distortion coefficients using `cv2.calibrateCamera()`.I applied this distortion correction to the test image using `cv2.undistort()`.
I also save the calibration matrix (`mtx`) and the distortion coefficients (`dist`) in a pickle file.
I obtain the following result from this process (**Figure 1**):

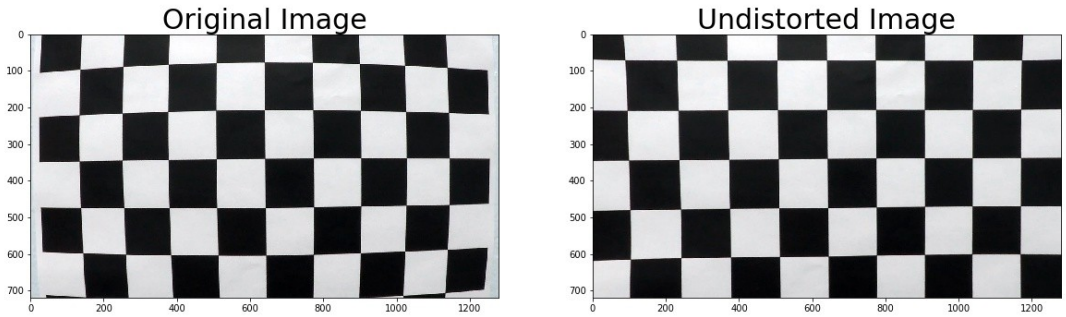

Figure 1: Chessboard undistorted image result

Pipeline (test images)

1. Provide an example of a distortion-corrected image.

Cell 4: To demonstrate the last step, I apply the distortion correction to each of the test images. **Figure 2** shows one example (rest of images in **output_images/undistorded_test*.jpg**)

Figure 2: Distortion correction result.

2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient threshold to generate a binary image. In particular, I my binary image is the result of thresholding the `V channel` and `x gradient` of the undistorted images. This can be seen in the `pipeline_threshold` function (**Cell 5**) . **Figure 3** shows an example of the resulting binary images (rest of images in **output_images/binary_test*.jpg**).

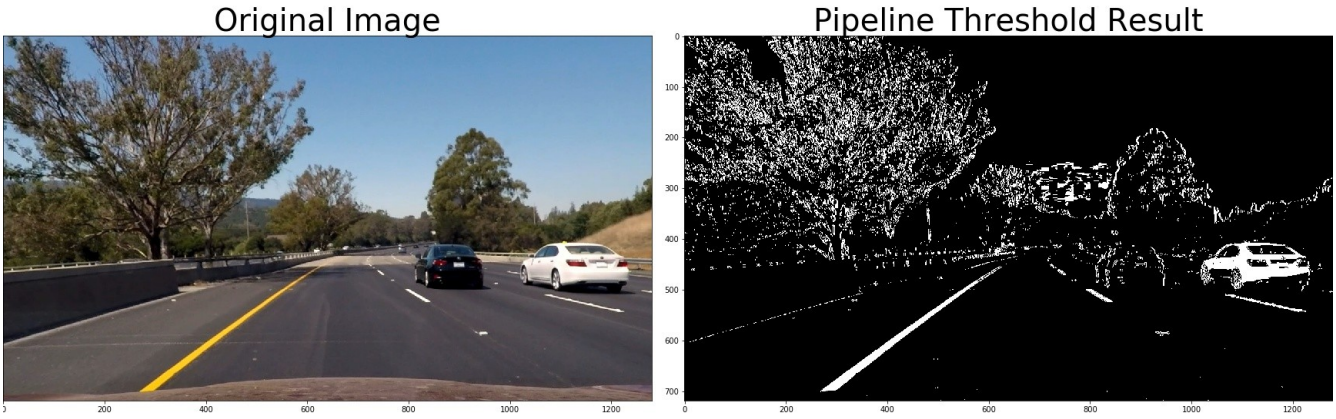

Figure 3: Binary image after color and gradient thresholding.

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

Cell 7: The function `corners_unwarp()` applies the perspective transform to get the "birds-eye view" of the images.

I hard-coded the source `(src)` and destination `(dst)` points, as shown in Table 1.

Table 1. Source and destination points for perspective transform

Source	Destination
580, 460	200, 0
195, 720	200, 720
1127, 720	950, 720
705, 460	950, 0

Figure 4 shows an example of the results of this process (rest of images in **output_images/color_warp_test*.jpg**).

Figure 4: Perspective transform result.

As it can be seen the lane lines appear to be parallel in the warped image

In addition, I apply the same process to the binary images. An example of the results can are shown in **Figure 5** (rest of images in **output_images/warped_binary_test*.jpg**).

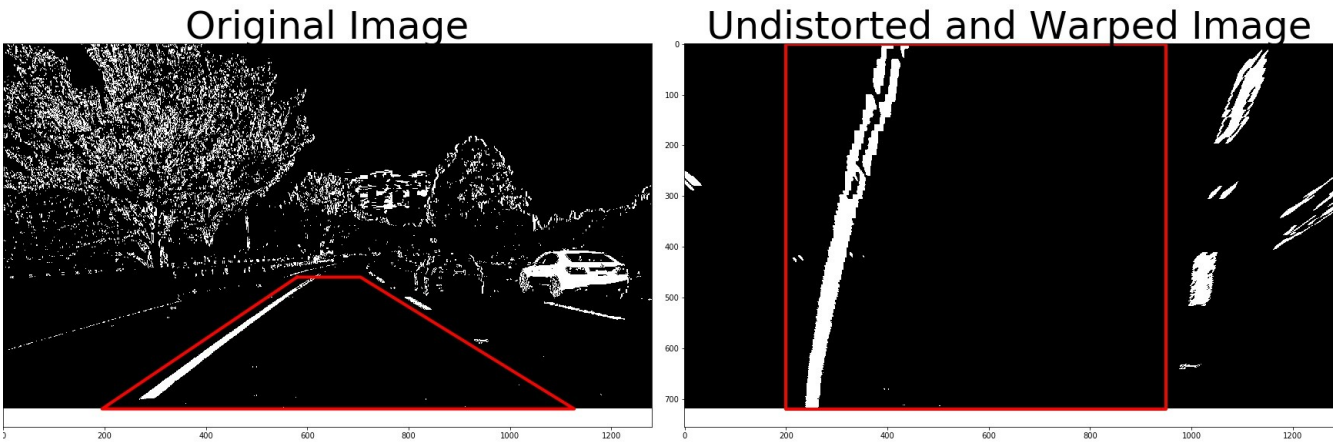

Figure 5: Bird-eye view binary image

4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Cell 10: In order to identify lane-line pixels and fit their positions with a polynomial I apply the sliding window method and find peaks in the histogram. I use the result of the sliding window to fit a 2nd order polynomial for the left and right lane-lines. This is done in function `lane_boundary_sl_wi()`.

An example of the results can be seen in **Figure 6** (rest of images in **output_images/lane_lines_test*.jpg**).

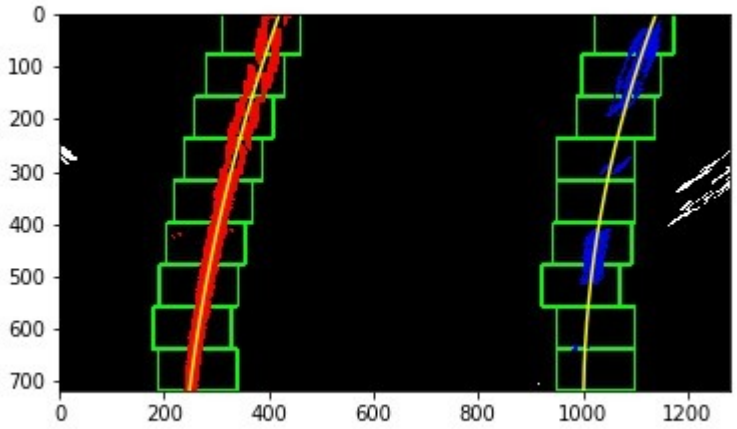

Figure 6: Lane-line pixels detection and polynomial fit.

5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

Cell 13 defines the function `rad_cur_px()` which calculates the lane-lines radius of curvature in pixels. In order to convert that radius to meters I define function `rad_cur_m()` (**Cell 15**).

The radius calculation is based on the formula see in the lesson:

$$R_{curve} = \frac{[1+(\frac{dx}{dy})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

Cell 17: The offset of the vehicle with respect the center of the lane is calculated in the function `lateral_offset()`. For this, I assume that the camera is mounted at the center of the car, such that the lane center is the midpoint at the bottom of the image between the two lines that I have detected. The offset of the lane center from the center of the image is the distance from the center of the lane.

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

Figure 7 shows an example of the pipeline results (rest of images in **output_images/result_test*.jpg**).

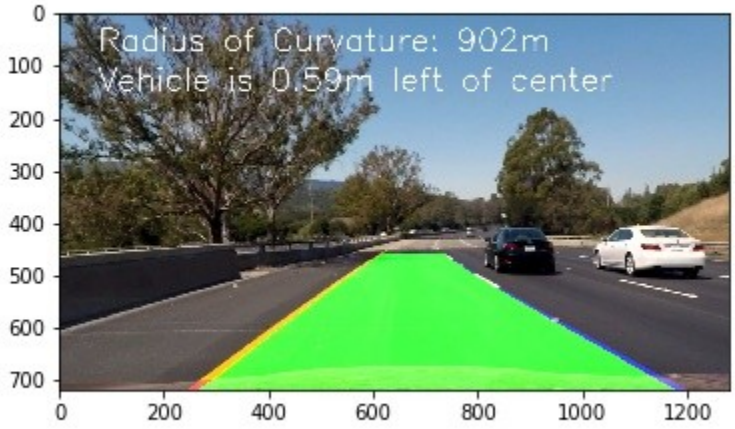

Figure 7: Pipeline result

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

A video of the project result can be found in the following link

https://github.com/pablosaurasperez/CarND-Advanced-Lane-Lines/blob/master/project_video_result.mp4

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

From my point of view two main approaches of this pipeline are subject to discussion.

- **Color and gradient thresholding.** I based my thresholding only in **V channel** and **X gradient** thresholding. This gave me good results for the test images and the project video. However, my guess is that this thresholding may be affected in **bright** images. It may be worth to consider **H channel** of the images or other kind of channels such as **L** or **S.**
In addition, other kind of thresholding such as **gradient magnitude** or **direction** could help. For example, gradient direction may help in filtering out lines that are not likely to be a lane-line based on the angle.

- **Perspective transformation.** I have selected the source points of the perspective transformation manually. In this way, my trapezoid covers an enough long horizon to detect the lines. However, this horizon is fixed. I have seen the harder_challenge video and notice that my horizon would go further away from the road. Thus, this approach would not work and another trapezoid should be considered. There may be automatic corner detection techniques that help to define this trapezoid.