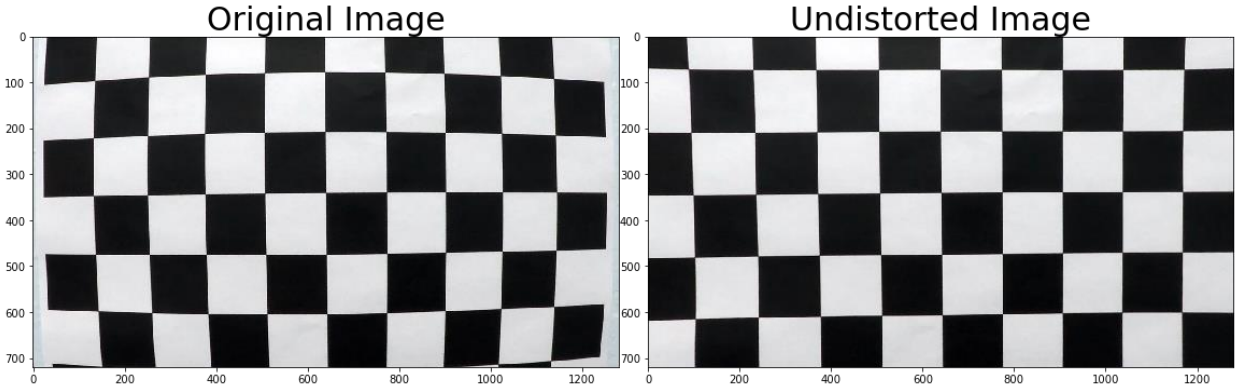
Report
Fan Zhou

Please see the notebook for all generated images.

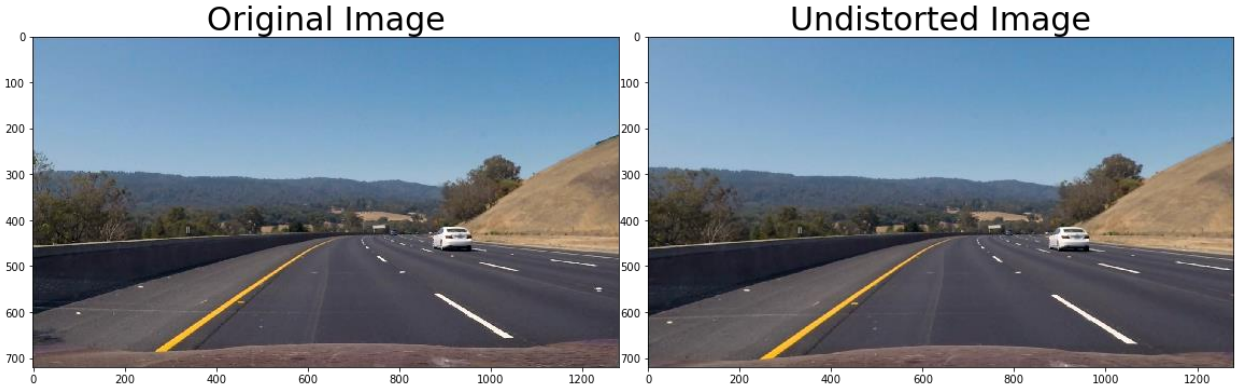
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

In the calibration, I first initialize the objpoints and imgpoints lists. For the chessboard is fixed on the (x, y) plane, z=0. It is easy to generate repeated coordinates for all images. For each image in the cameral_cal folder, I first convert it to grayscale. By running cv2.findChessboardCorners function, I can find corners and append it into imgpoints list. Then I pick one image and run cv2.claibrateCamera function to get the camera matrix (mtx) and distortion coefficient (dist), respectively. Once I have mtx and dist, I can calculate the undistorted image by running cv2.undistort function.



Pipeline (single images)

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



It can be carefully seen two images have some differences, like car hood and positions of white lines at the edges of the image.

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.

Unlike the method in the video (processing color space first then performing perspective transform), I did perspective first and then selected proper color space. Please see the “Perspective Transform”, “Sobel”, and “HLS, LAB color space”. Finally, the L-channel from HLS and B-channel from LAB are used to extract the lane pixels.

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

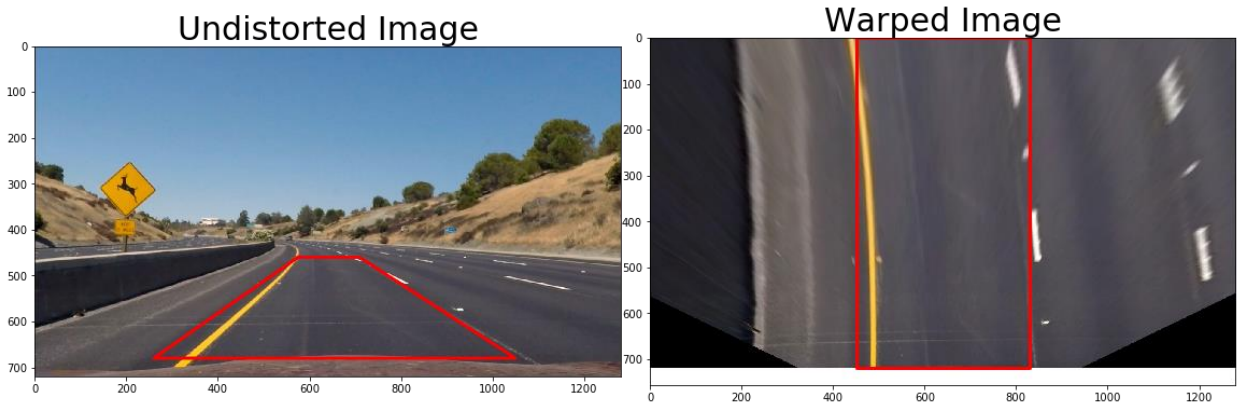
Please see the “Perspective Transform”, I defined the function “warp” that needs image, source points and destination points as inputs. Transform matrix (M) and inverse transform matrix (Minv) can be calculated by

```
M = cv2.getPerspectiveTransform(src, dst)

Minv = cv2.getPerspectiveTransform(dst, src)
```

The source points I use are

```
src = np.float32([(575,460),
                  (710,460),
                  (260,680),
                  (1050,680)])The destination points are

dst = np.float32([(450,0),
                  (w-450,0),
                  (450,h),
                  (w-450,h)]),
```

where h and w is number of rows and columns of the image.

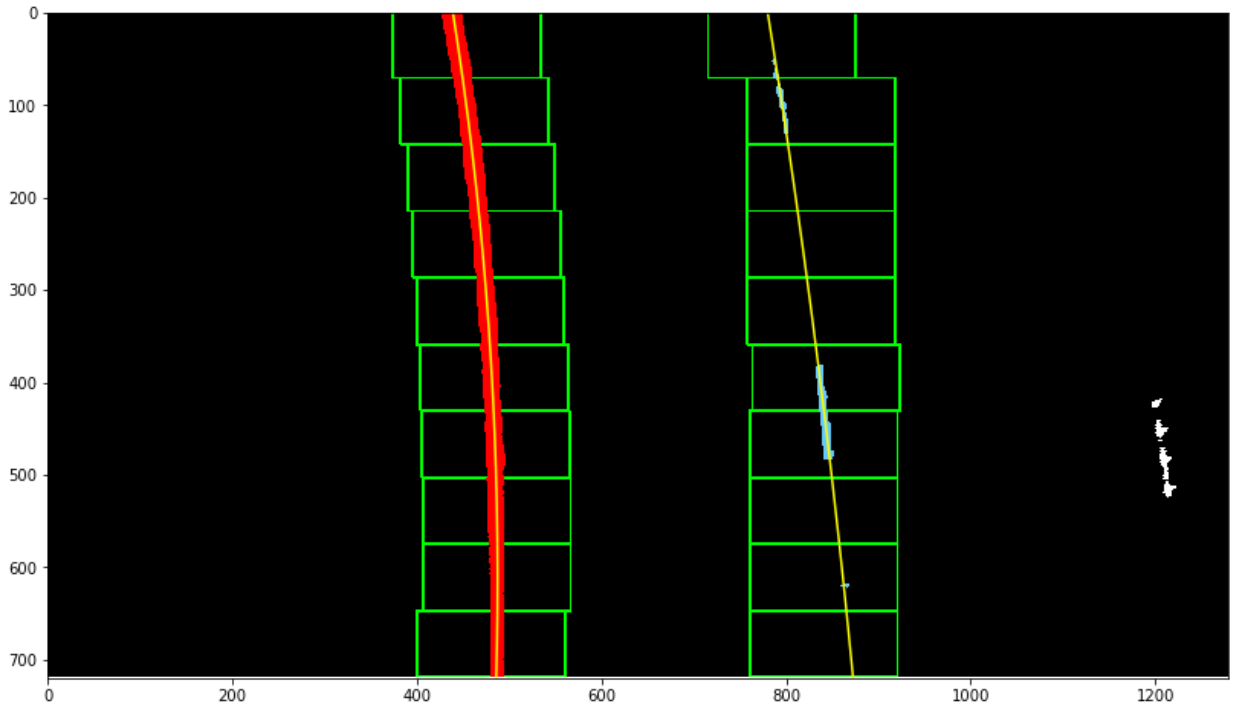


4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

In the sliding window polynomial fit, I apply the same method illustrated in the video to find the non-zero pixels on the lanes, and then fit these pixels with polynomial function. For example, for the following image, the fitted lines are is yellow and the fitting coefficients from higher order to lower order are

Left lane (red points): array([3.95647011e-05, -1.28836159e-01, 5.35005806e+02])

Right lane (light blue points): array([8.16899253e-05, -1.49263891e-01, 9.10094595e+02])



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.

Please see “Radius and Deviation from Lane Center Calculation”.

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

I draw two short lines here to indicate the lane center and the car position, respectively. The distance between them is defined as deviation. For example in the following image, the deviation is 0.385m.

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?

I found the following issues:

1. One line has few data point while the other line is continuous. For example, the left yellow line is continuous and clear while the right white line is non-continuous and sometimes unclear. This causes the line with fewer data point has very different fitting parameters compared the continuous line.

2. When the car has several continuously steep turns, the detection program doesn't work well, because in the perspective view the lines can't be described by the 2nd order polynomial function. Higher order polynomial fitting is necessary. In addition, sliding window method may not extract correct pixels during the steep turn since the desired lines change their positions with respect to the car camera (not 1/4 position assumed in the code). For example in the steep right turn, the left line moves to the center of the image while the right lane may totally disappear.

3. When the road is narrow, other edges besides the lane lines are also easily detected by the sliding window method. This causes inaccuracy in finding correct lane width.

4. In the video it is assumed the lane is flat, so the trapezoid coordinates can be fixed value, however, when the car is running on the slope (up or down), the trapezoid shape needs change to adapt the lane shape, otherwise the lane filling is not accurate.