Project Report

Advanced Lane Finding

Name: Zeyu Tang

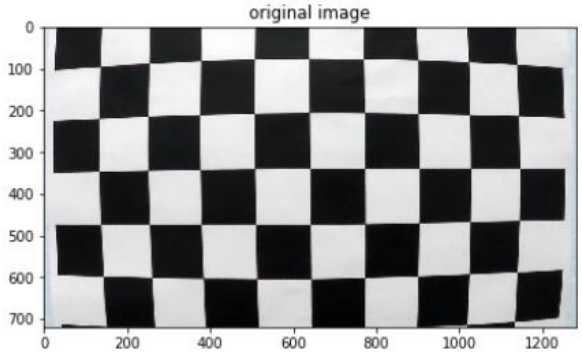
The goal is to write a software pipeline to identify the lane boundaries in a video from a front-facing camera on a car. Here are my steps.

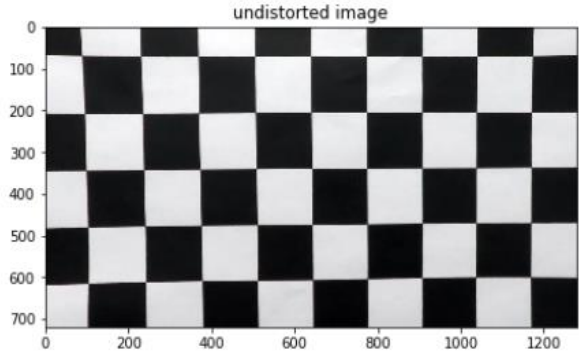
Camera calibration

I use *cv2.findChessboardCorners()* function in OpenCV to get all corners on grayscale of all calibration images. These corners are image points. Then I set the coordinate frame on the chessboard and get all object points corresponding to image points.

After that, I use *cv2.calibrateCamera()* function in OpenCV to get coefficients of matrixes distortion and calibration.

As an example, I use these matrixes to undistort ‘calibration1.jpg’. Here is the result.



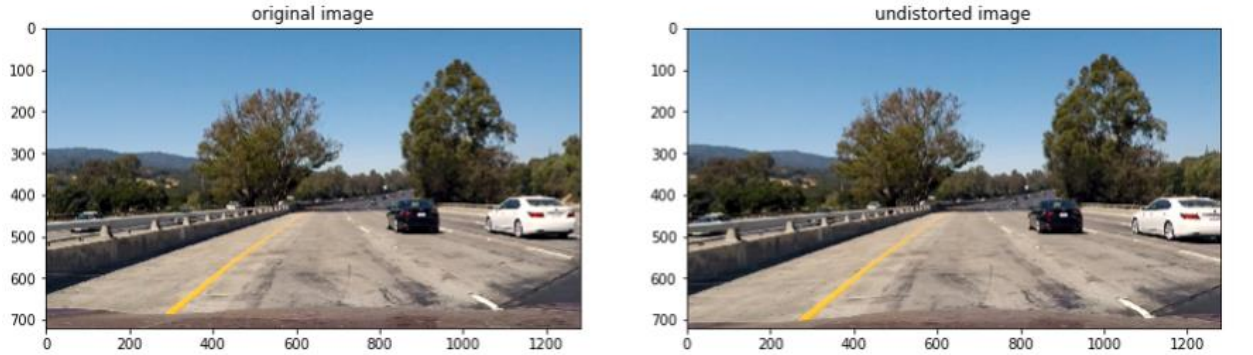


Pipeline

Image processing

Undistort image

I use matrixes got from camera calibration to undistort the input image. Here is an example.



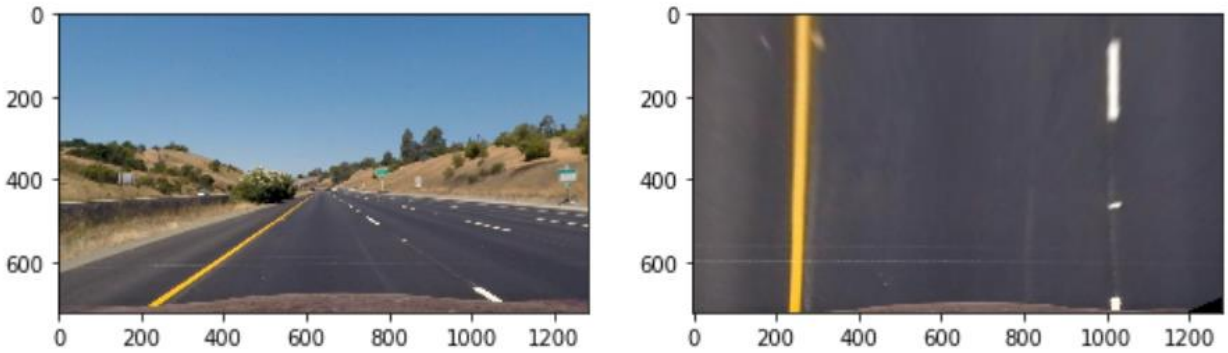
Get M matrix of perspective transform

I use ‘straight_lines1.jpg’ of test images to get the M matrix of perspective transform, which is used in processing all images.

I manually select 4 points in the images as source points and set four distination points, and perform the perspective transform by using *cv2.getPerspectiveTransform()* function to get the warped image.

Then I slightly tune the coordinates of 4 souce points to make the lines in the warped image nearly vertical and straight.

Here is the result of perspective tranform.

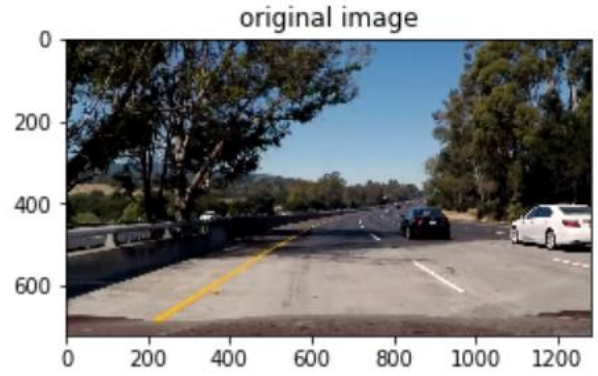


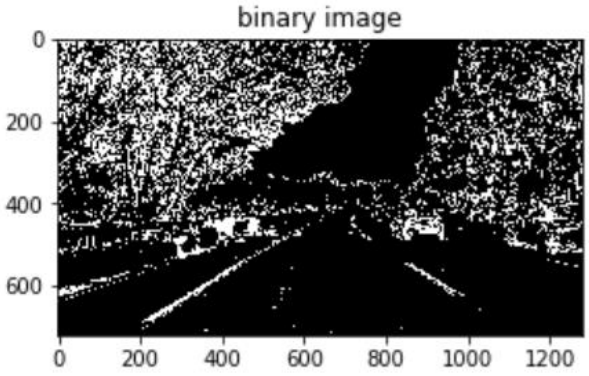
Theresholding

I use the combination of y gradient on G channel of GRB image and S channel of HLS image to mask the undistorted image. The theresholds are showed below.

	Max thereshold	Min thereshold
y gradient of G channel	100	20
S channel value	255	170

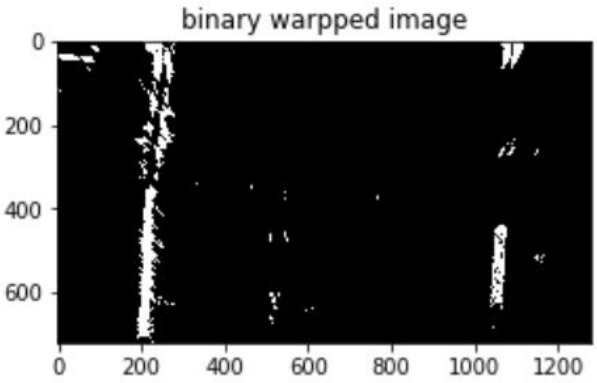
Here is example of apply theresholding. The source image is ‘test5.jpg’.





Wrap image

I use the M matrix gotten above to wrap the binary image. Here is the wrapped image of the binary image above.



Search useful pixels

Get the histogram and find the x values of left and right line on the bottom of image

By summing each column of the bottom half of binary wrapped image gotten from the image processing, I get the histogram of the bottom half of the image. Then from the right half and left half of the image, I choose the x value of the column with the highest value in histogram as the x-position of bases of left and right line.

Then I decide whether the right and left lines are detected from the former frame by comparing the base x values of current and former frame. If the input is a single image, then the lines in it are always not detected from the former frame.

Find useful pixels by sliding 9 windows

If lines in current frame is not detected in the former frame, I use sliding windows to find useful pixels.

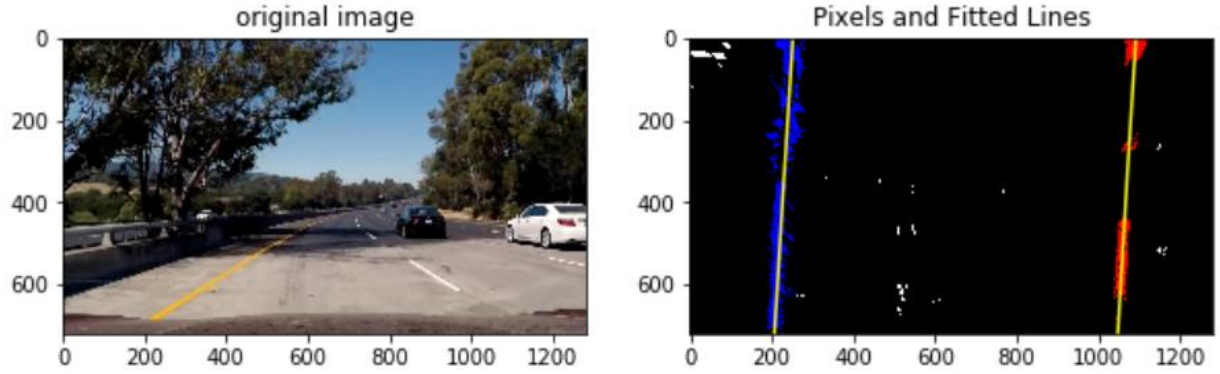
The image is divided into left and right part evenly. For each part, 9 longitudinally connected windows with margin of 50 are used to find nonzero pixels. For each part, from the bottom of the image, the first window is centered at the x-position of bases of the line, and following windows are centered at the center of previous window's center unless the previous window find more than 50 pixels, in which case the following window is centered at the mean positions of those pixels found in the previous window.

Find useful pixels by searching around the fitted lines in the previous frame

If lines in current frame has been detected in the previous frame, I just simply searching the area around the fitted lines in the previous frame with the margin of 50.

For every single test image, the pipeline uses sliding windows to find useful pixels.

Here is an example of results. Useful pixels are colored. The fitted lines in the image are second order polynomial to these pixels. The source image is 'test5.jpg'.



Define a class, Line (), to further process these useful pixels.

Get fitted results, curvatures and the position of the vehicle

In the class, I define a function *UpDate()* to do all the calculations.

The function takes in positions of useful pixels and use them to get coefficients of second order polynomial. According to these coefficients, the curvature can be calculated based on the following formula:

$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

And the position of the vehicle with respect to center L can be calculated by applying:

$$L = ((x_{left} + x_{right} - 1280) * xm\_per\_pix$$

Where x is the x value on fitted lines when set y value as the image height (719), xm_per_pix is the factor to convert pixel to meter.

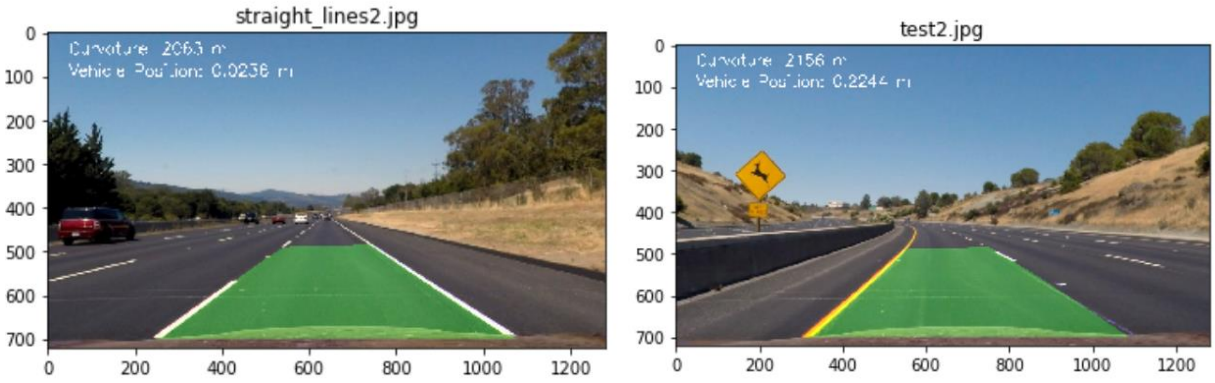
Filter noise

Besides, the function stores the fitting coefficients used in the last 5 iterations. And if the new fitting coefficients of current frame have mild difference with the average coefficients of the last 5 iterations, the current frame will be considered as valid frame. In valid frames, new fitted coefficients are used for finding and displaying lanes, calculate curvatures and vehicle position. In other frames, the average coefficients of the last 5 iterations are used to do these things. And only fitted coefficients in valid frames are stored.

If the stored set of fitted coefficients is less than a minimum number, which I set 5, the stored data will be considered to be weak, and the current frame will be consider as valid frame.

Results

I uses my pipelines on test images. And here are some of results. All final output images can be seen in the folder ‘output_images’.



Then I use my pipeline on ‘project_video.mp4’. The output looks good, and is saved as ‘project_video_output.mp4’.

Conclusion and Discussion

The pipeline works well on testing images and project video.

However, there are some issues.

Firstly, although the binary image after undistorting and thresholding clearly shows lanes and with few noise, the wrapped image of it may have much bigger noise, because the wrapping process may enlarge some noising area. To avoid it, I tried to choose the destination points of perspective transform to be close to lanes, to mask out as much noising area as possible.

Secondly, the thresholding method can be further improved to make the pipeline work well for ‘challenge.mp4’. But it requires lots of effects on trials different combinations of parameters and methods. I will keep working on it after the project.

Thirdly, there are always some situations where another vehicle passes the camera and get very close, and my lane finding function fail to keep tracking lanes correctly. In my pipeline, I choose to not consider frames with these situations as valid frames. That works for all videos provided. But it not a practicable way because if there were lots cars keep passing the camera, the pipeline would fail to work since lots of frame are abandoned.