

Advanced Lane Finding Project

The aim of this Project , detects the lane lines and find the curvatures of left and right lines with car position. I used below steps to finish the project.

The goals / steps of this projects:

1. Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
2. Apply a distortion correction to raw images.
3. Use color transforms, gradients, etc., to create a thresholded binary image.
4. Apply a perspective transform to rectify binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Warp the detected lane boundaries back onto the original image.
8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

We have many test images to show the results of each steps.

Images Inputs and Results:

You can find input test images in this folder:

\test_images

I will show my sw solution on these input test images before doing video process.

\test_images\test1.jpg

\test_images\test2.jpg

\test_images\test3.jpg

\test_images\test5.jpg

\test_images\test6.jpg


You can find output result images in this folder:

\output_images

There two main programs to shows the result of Project.

The first one, "Test_Single_Image.ipynb" which shows all steps on test images.

If you run below jupyter Python file you can see for the given input image.

•  Test_Single_Image.ipynb

It easy to adjust the input reading image at the lines 14 and 377.You have to write same input image name.

lines 14 → Distorted_test_image = mpimg.imread('test4.jpg')

lines 377 → test_image = mpimg.imread('test4.jpg')

The second one is "Advanced_Lane_Lines_Project_2.ipynb" which generates the output video.

Step-1: Camera Calibration:

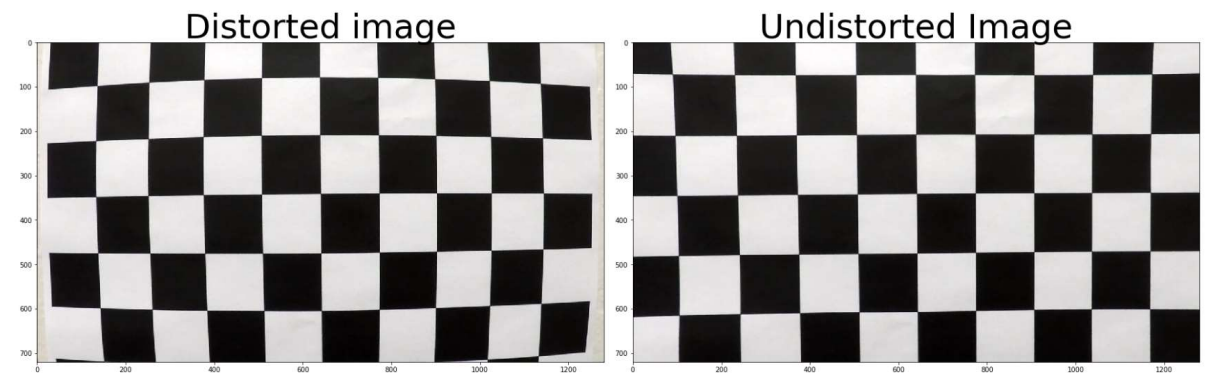
I will use images under this folder to find calibration coefficients.

\camera_cal

I used 20 chessboards images, taken at different angles and distances. Each of them has nine by six corners to detect.

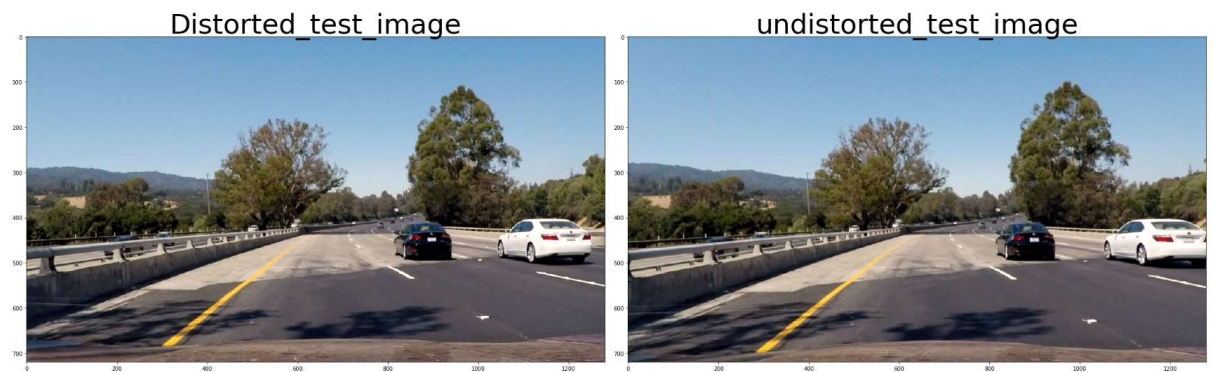
Test_Single_Image.ipynb

Lines →13-60



Step-2: Apply a distortion correction to raw images.

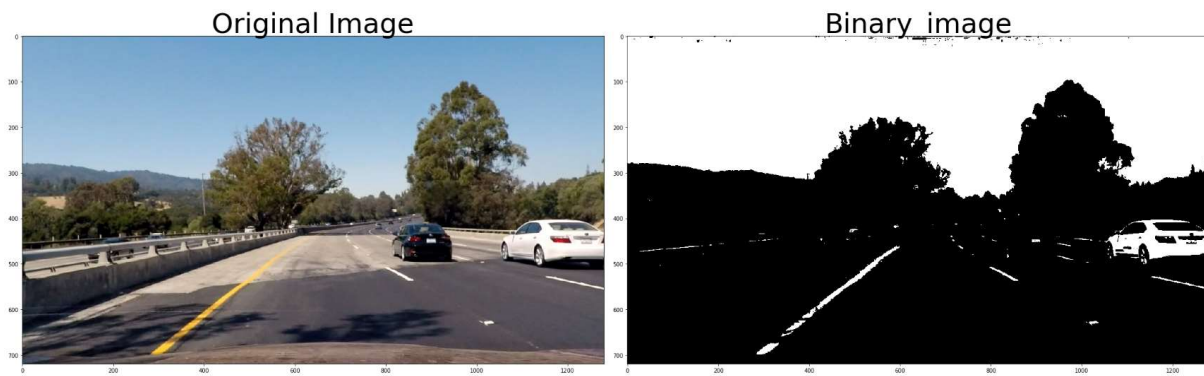
Lines →61-71



Step-3: Use color transforms, to create a thresholded binary image.

I used Color transform which depends on yellow and White line colors. We have Binary_Conversion_Yellow_White(image) function to achieve this binary conversion.

Lines → 82-129

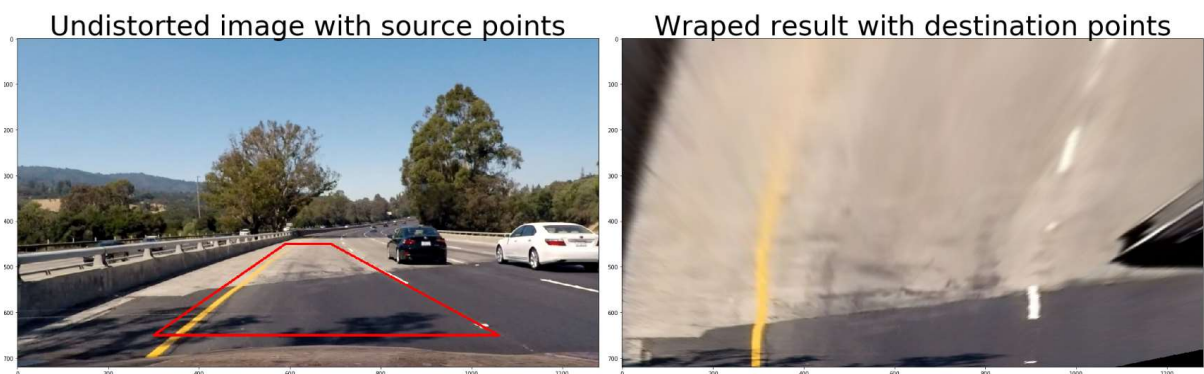


Step 4: Apply a perspective transform to rectify binary image ("birds-eye view").

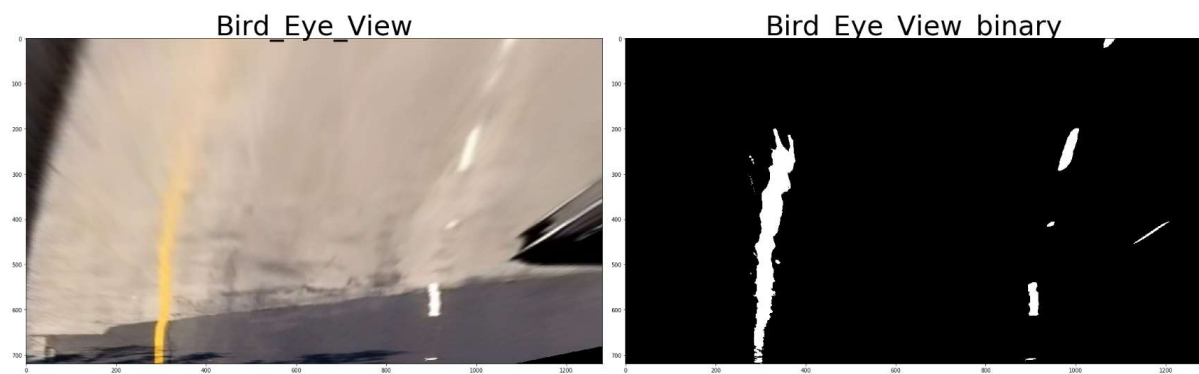
It was not easy to find the source and destination points to catch the best perspective image. Because the lines must be parallel after perspective transform.

```
src = np.float32([
    [590, 450],
    [690, 450],
    [1060, 650],
    [300, 650]])

dst = np.float32([
    [250, 0],
    [896, 0],
    [896, 720],
    [250, 720]])
```



After this we can apply binary transform for Binary View image. We use Yellow and White binary conversion function.

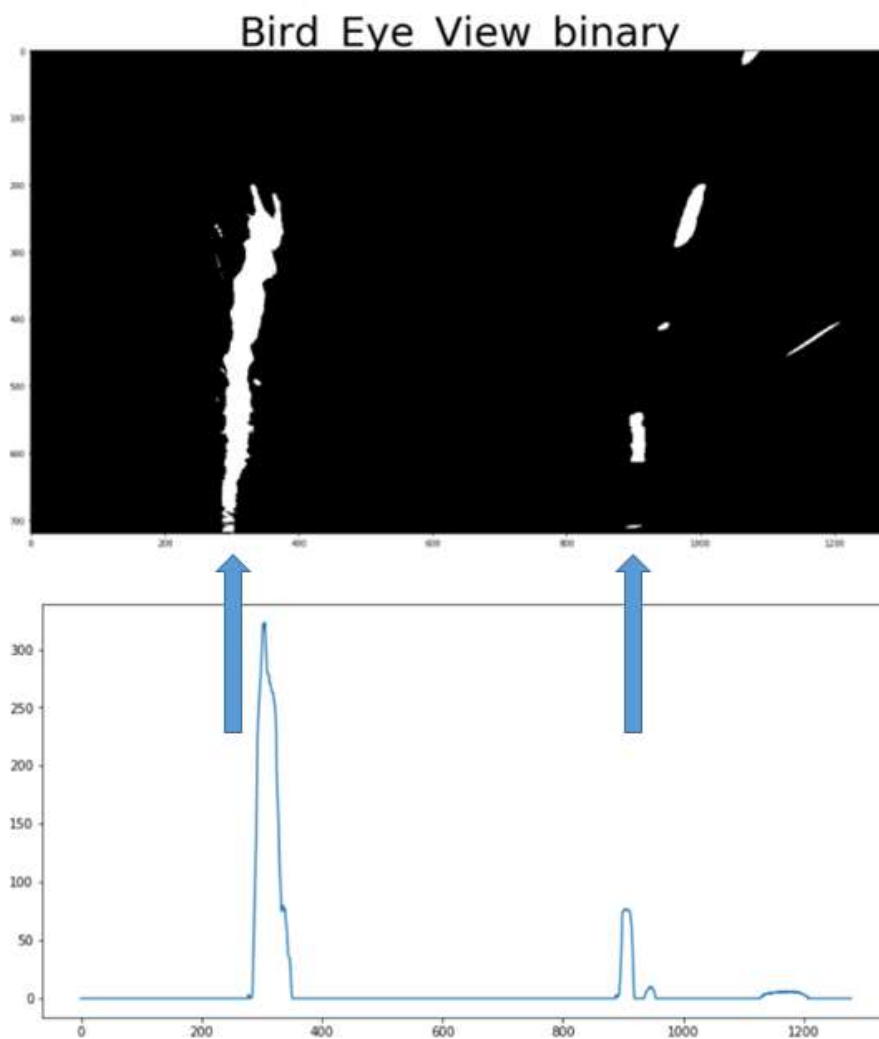


Step 5: Detect lane pixels and fit to find the lane boundary.

In order to find starting points of the left and right lines, we can use histogram at the bottom half of the binary wrapped image.

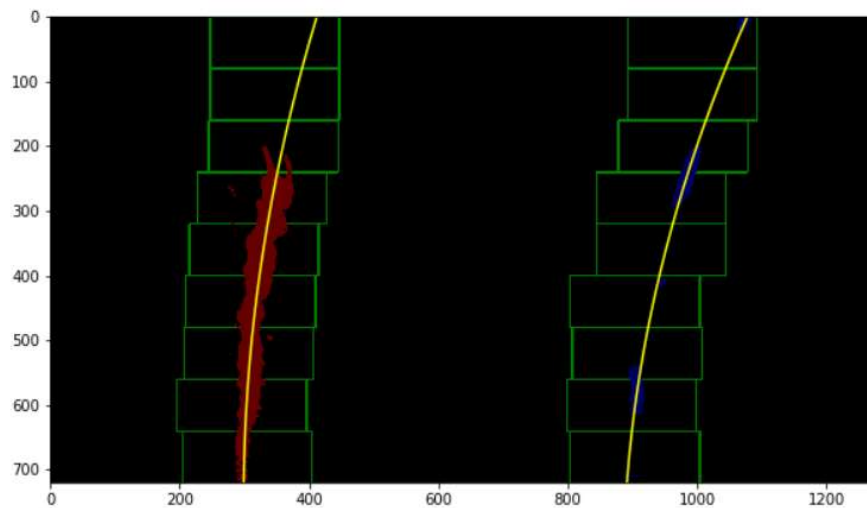
Histogram Result;

Line:194



After finding histogram, we can use sliding window approach to detect the lane lines.

Sliding window approach result:



Step 6: Determine the curvature of the lane and vehicle position with respect to center.


We have a special formula to find left and right lines curvatures.

Lines:189-329

```
327 #Calculate the new radius of curvature
328 left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
329 right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
```



In order to see the video creation please run

•  Advanced_Lane_Lines_Project_2.ipynb

The video Result:

```
#####

Frame_lane_line = Line()

#Video Processing...
from moviepy.editor import VideoFileClip
from IPython.display import HTML
white_output='Project_2.mp4'
clip1 = VideoFileClip("project_video.mp4")
clip1.reader.close()
clip1.audio.reader.close_proc()
white_clip = clip1.fl_image(Frame_lane_line.pipeline)
%time white_clip.write_videofile(white_output,audio=False)

[MoviePy] >>> Building video Project_2.mp4
[MoviePy] Writing video Project_2.mp4
10% |██████████| 129/1261 [00:29<04:42, 4.01it/s]
```

Reflections:

- ✓ It is the second time with Python program, so this code can be more simple.
- ✓ I think The binary threshold must be independent from yellow and White line color. Because many types of roads could be just White color lane lines. So gradient or combined color thresholds must be applied to image.
- ✓ Class structure can be more powerful, for example I can read camera matrix from file. In my program, I calculate the camera matrix. I should read them from file.
- ✓ If there is no any right line, how can we find the curvature? On the other hand I can use another sliding window to find line points.