Advanced Lane Finding Project

I have attached the following with the project:

- Writeup.md
- Writeup.pdf
- Project_2.ipynb(Jupyter Notebook)
- Project_2.html

The goals / steps of this project are the following:

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

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

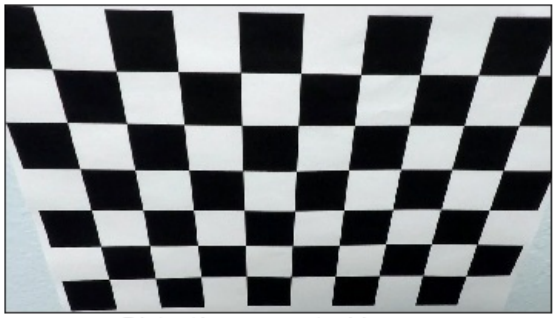
Camera Calibration

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

- The camera matrix and distortion coefficients were calculated using the cv2.findChessboardCorners() and cv2.calibrateCamera() methods.
- Object_points were calculated by processing a np.array and reshaping it in order to obtain the object_points matrix.
- Image_points were calculated using the chessboard corners obtained using the findChessboardCorners()

Attached are the original and un-distorted images:

Original image



Distortion corrected image

Pipeline (single images)

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

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:


Original image


Distortion corrected 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.

I used a combination of color and gradient thresholds to generate a binary image. Here's an example of my output for this step. (note: this is not actually from one of the test images)


Original test image


Processed test image

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

The notebook contains the code in order to perform the Perspective transform on an image. * The method perspective_transform() performs the required steps in order to do a obtain a "bird-eye" view. * Method cv2.getPerspectiveTransform() is used in order to obtain the transformation matrix. * Method cv2.warpPerspective() is used in order to obtain the "bird-eye" view of the road image.

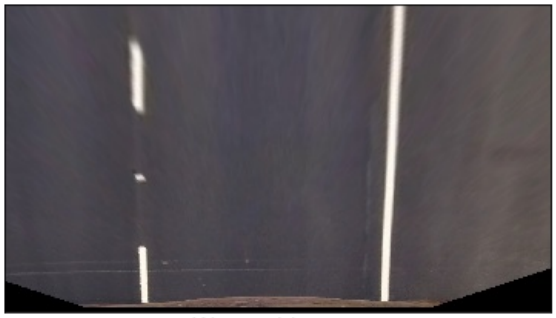
```
src = np.float32([[190, 700], [1110, 700], [720, 470], [570, 470]])
bottom_left = src[0][0]+100, src[0][1]
bottom_right = src[1][0]-200, src[1][1]
top_left = src[3][0]-250, 1
top_right = src[2][0]+200, 1
dst = np.float32([bottom_left, bottom_right, top_right, top_left])
```

This resulted in the following source and destination points:

Source	Destination
190, 700	290, 700
1110, 700	910, 700
720, 470	920, 1
570, 470	320, 1

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

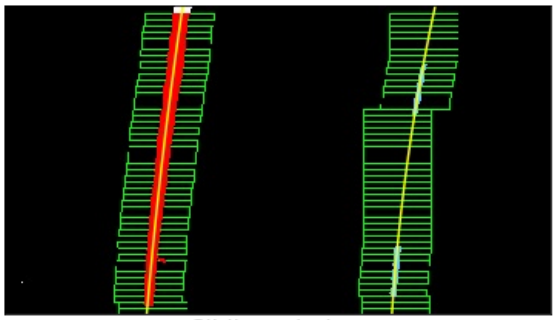

Original image


Warped image

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

I've referenced to the lessons provided by Udacity in order to compute the lane line pixels in the warped and color transformed images. * First step was to plot a histogram of the image. * Then, split the histogram into 2 so that left and right lanes can be identified by the left and right peaks of the histogram. * A sliding window mechanism was used in order to detect the non-zero pixels in and image along the histogram peaks. * The lane line pixels were stored in an array and their positions were fit with a polynomial.


Original image


Sliding window

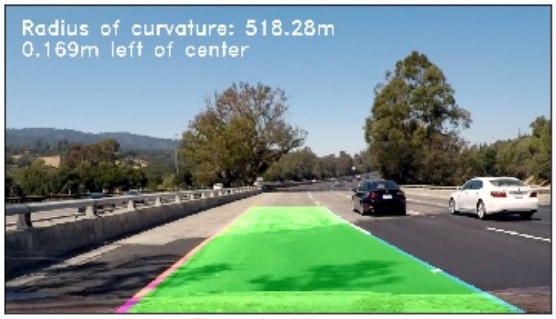
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

- First the lane width were determined by using the US standard line for lane width.
- 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]) 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])
- The above methods were used in order to determine the radius of curvature 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.

The image was transposed over the initial image in order to gain the images with lane lines marked and radius of curvature for the lane lines marked.


Original image



Detected Lane

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!).

Here's a link to my video result

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've used the techniques that were taught to me during the lessons provided in the course.

Shortcomings:

- The methodology fails when there is a sharp change in the lighting conditions(harder challenge video).
- The methodology fails when there is a difference in the color gradient of the lane(challenge video).