Report
Yi Zhu

Advanced Lane Finding Project

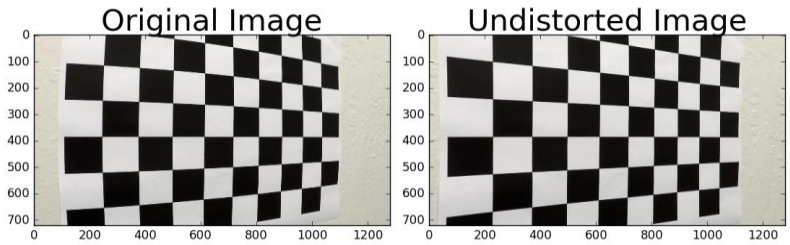
The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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.

- ## Camera Calibration

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

  I start by preparing "object points", which will be the (x, y) coordinates of the undistorted chessboard corners . Thus, `objp` is just a replicated array of coordinates, and `objpoints` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detectionI then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result



Pipeline (single images)

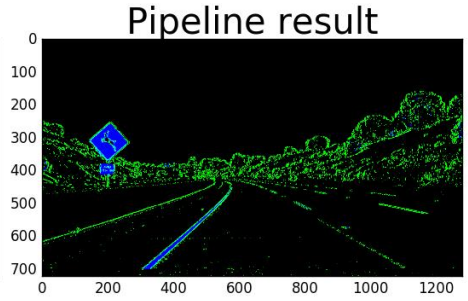
1. Provide an example of a distortion-corrected image.

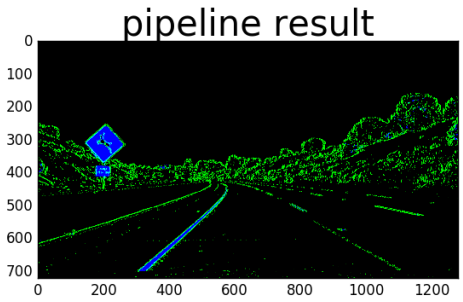
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

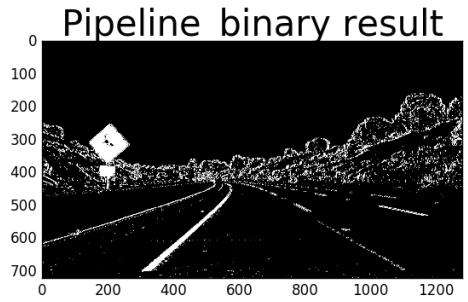
Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I first generate a pipeline showing what color and gradients detect respectively. Then, I used a combination of color and gradient thresholds to generate a binary image. Here's an example of my output for this step.


undistorted
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

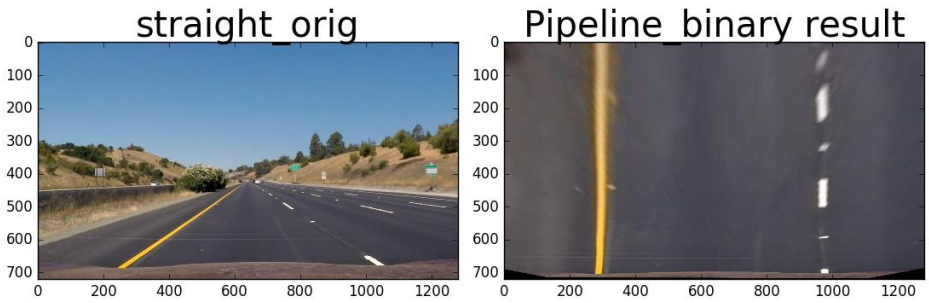

Pipeline result
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


pipeline result
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Pipeline binary result
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

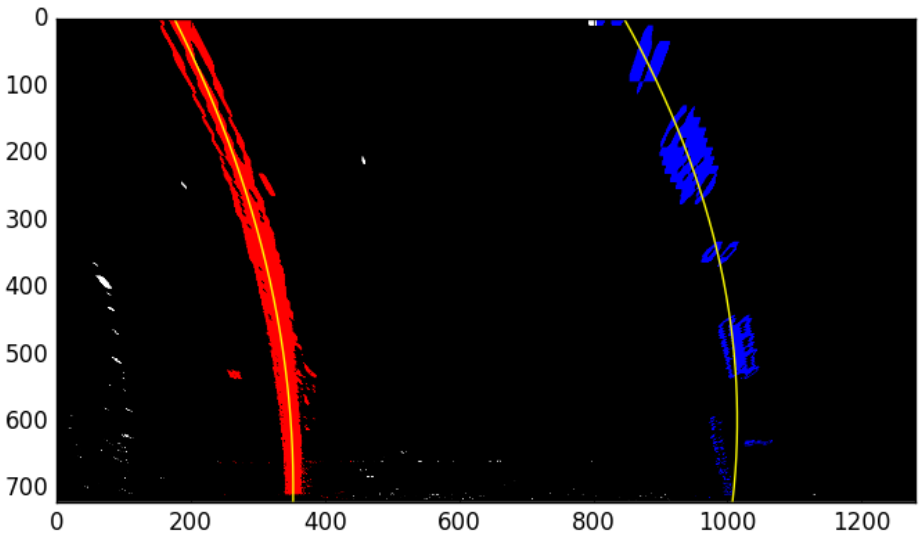
Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

I choose [247,677],[1039,672],[588,453],[692,453]] to be my source points and [[270,700],[980,700],[270,0],[980,0]] to be my destination points, then use cv2.getPerspectiveTransform(src, dst) to get the transformation matrix M.(In notebook cell 128). We can see transformed lanes have parallel boundary lines.



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

Then I used window sliding technique introduced in lesson to identify the path of sliding window and then fit my lane lines with a 2nd order polynomial kinda like this: we can see even the lines are not straight but they are still parallel.



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.

I did this in notebook starting cell 136.

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

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?

The output seems pretty good. However, the main problem with this type of method is the issue of robustness. During the implementation of the pipeline, there are several important parameters which are manually tuned. This could be a big problem if we change an another car or even in a different road. The possible solution could be we first classify the condition of the road when reading an input or a series of image and then the parameters value are given according to the classification result. This, I believe, in some sense reduce the issue of robustness.

With the help of first review, I noticed there are several frames not performing very well. So I

adopted sanity check technique. In detail, I used a class called line() to keep track of lane found in previous 10 frames. Due to the smoothness of the lanes, I reject the new lane if it deviates a lot from the average of previous 10. I basically calculated the average coefficient and new coefficient , then compare the difference norm to see if it exceeds some threshold. Luckily, this method successfully vanish the significant erroneous frame in last submission.

With the hint of second review, I added two more sanity check. First I compared the poly constructed by left and right lines in successive frames, if the similarity measure is above some threshold, I will keep the original fit line. Second, I added a left-right line distance check, if the distance is above some threshold, then keep the original fit.