Advanced Lane Finding

Self Driving Car Nanodegree

Aman Ahluwalia - 18 August 2017

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.

Rubric Points

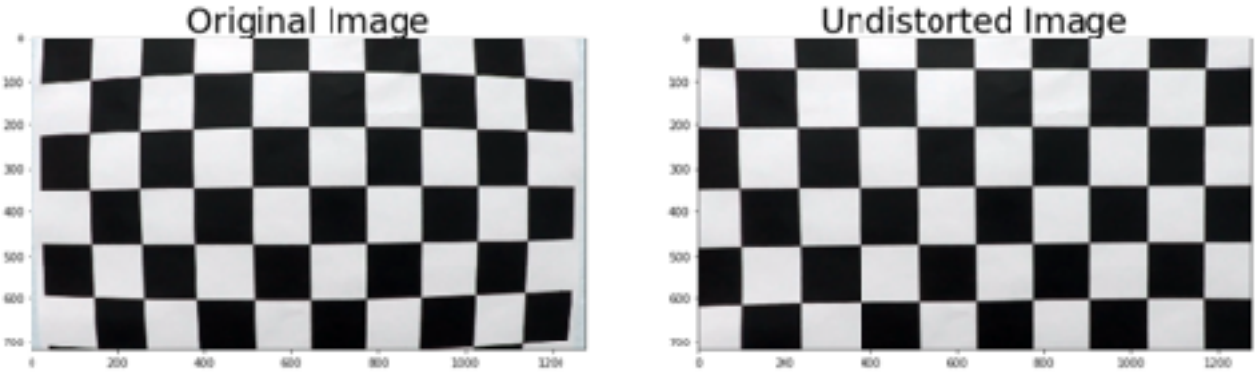
Camera Calibration

Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

The code for this step is contained in the first and the second code cell of the IPython notebook advanced_lane_lines.

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. 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 detection.

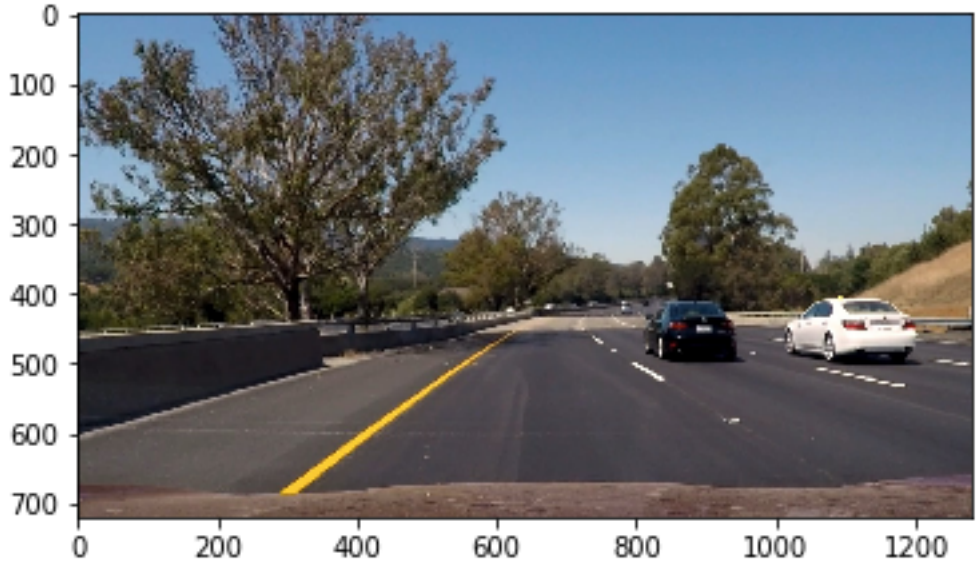
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



Pipeline (single images)

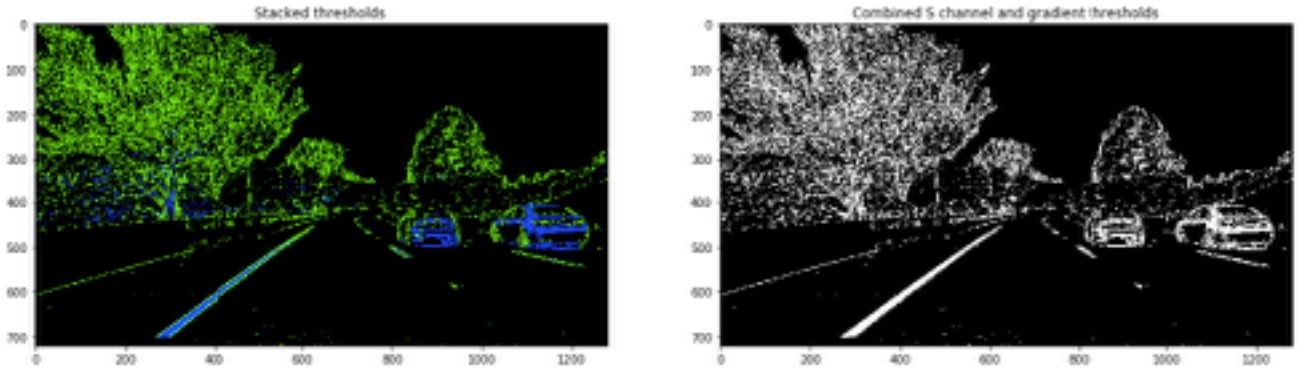
1) *Provide an example of a distortion-corrected image.*

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



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 (thresholding steps at the fourth code cell of the IPython notebook advanced_lane_lines). Here's an example of my output for this step,

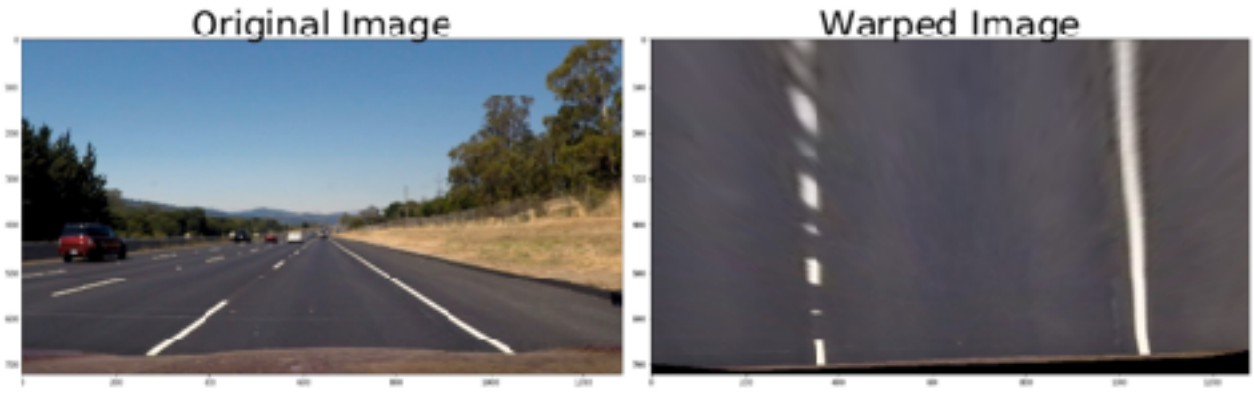


3) *Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.*

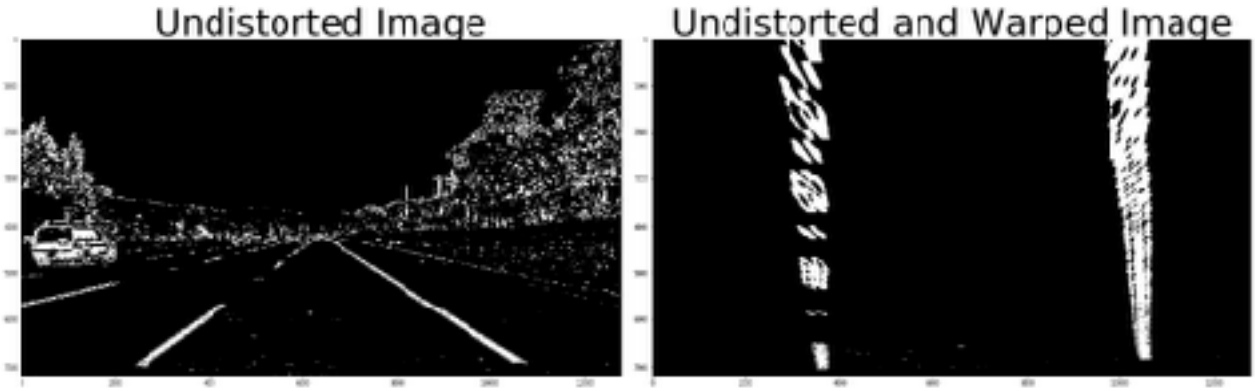
The code of perspective transform is present in the fifth code cell of the IPython notebook advanced_lane_lines. It aims to find the M and Mine, the transform and the inverse transform matrix respectively. The source and the destination points to identify M are,

Source	Destination
260 , 680	350 , 700
1040 , 780	950 , 700
668 , 440	950 , 0
612 , 440	350 , 0

Then in the next code block I verified that my perspective transform was working as expected by checking on one of the test images, as in the warped counterpart of the image the lane lines should appear parallel.



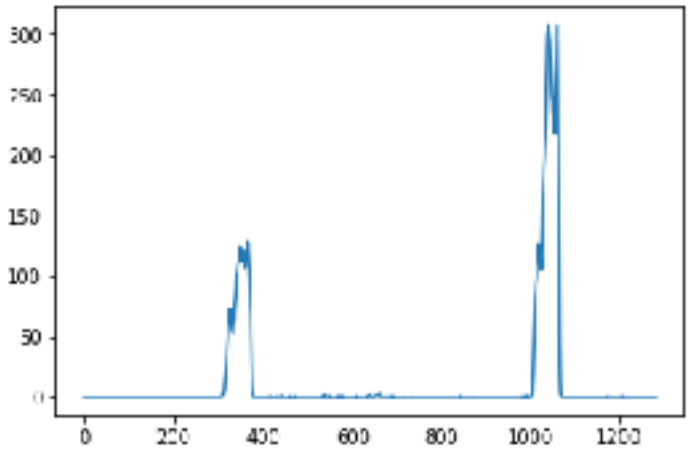
Then we brought all the above steps together and checked on one of the test image, to analyse the binary warped image,



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

In code cells from 7 to 11 of the IPython notebook advanced_lane_lines I did some other stuff and fit my lane lines with a 2nd order polynomial.

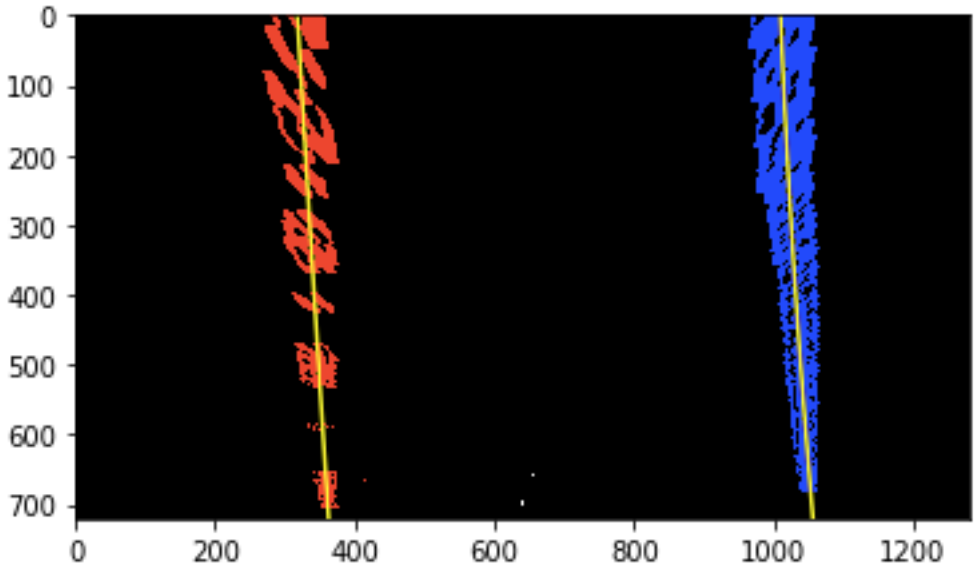
I first take a histogram along all the columns in the lower half of the image,



Then i used a sliding window approach, placed around the line centers, to find and follow the lines up to the top of the frame.

We will be Skipping the sliding windows step once you know where the lines are.

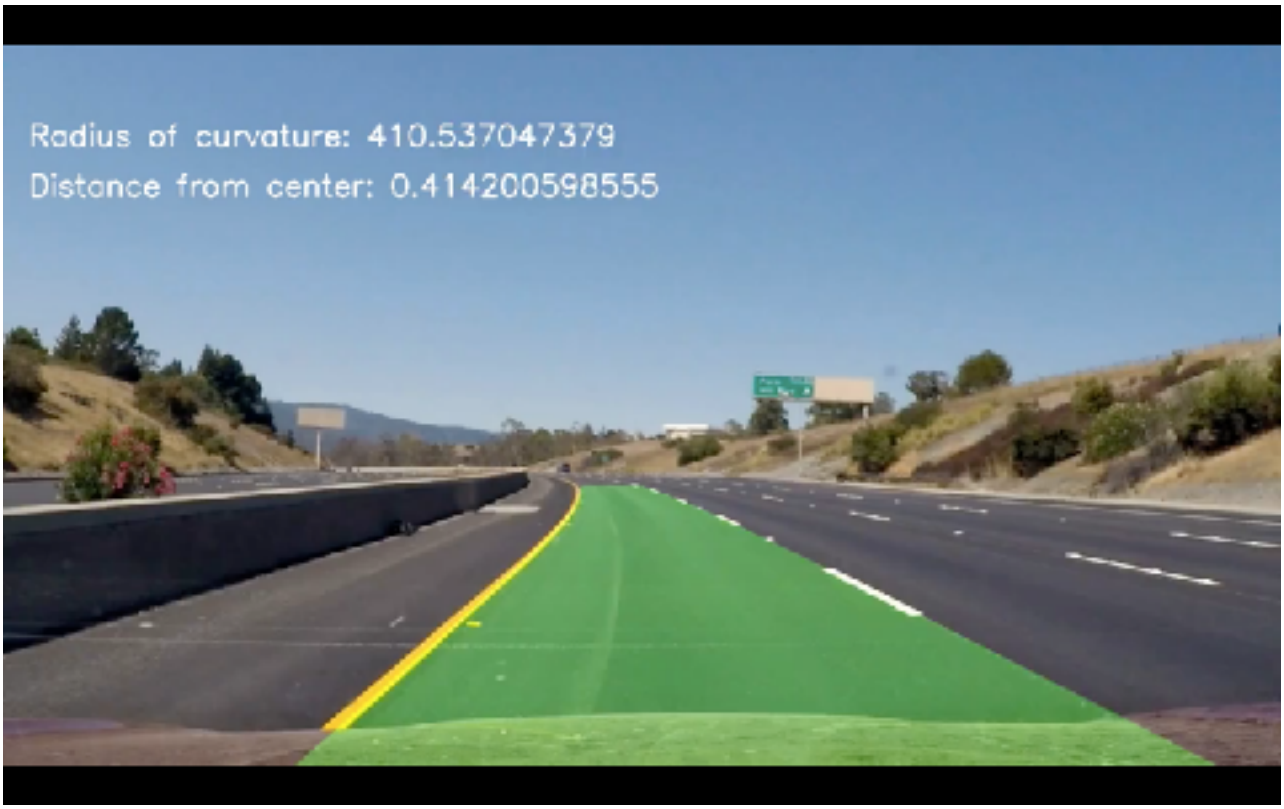
In my implementation, sort of a sanity check, i am reusing the main method after every 100th frame, to maintain consistent result.



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.

It is implemented in the 7th and the 10th code cell of the IPython notebook advanced_lane_lines. Though i have defined a Line class where i am calculating the radius of curvature in the line class, but we are averaging the two in the 11th code cell for the visualisation purpose.
The lane position is calculated in meters in the 10th code cell. It is the absolute distance from the centre, as the camera is placed in the centre of the vehicle.

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



Pipeline (video)

Here's a the video of the final output.

Writeup / README

You're reading it!

Discussion

The first four parts until the transformation were simple, now came the part where we need to identify the lines. Though it was clear that something like sliding window approach would work, which can be implemented in two ways, one was with counting the number of pixels with value 1, another way to approach the sliding window method is to apply a convolution, which will

maximize the number of "hot" pixels in each window. I preferred to move with the first approach as it was more clearer and was more inclined to give the desired results.

Now after determining the lines in one of the image we, don’t need to repeat the above steps from scratch for every frame of the video, as the roads move consistently in one of the directions. Now you know where the lines are you have a fit! In the next frame of video you don't need to do a blind search again, but instead you can just search in a margin around the previous line position.

But i analysed using the above method that some risky frames, consider turns or where our binary image didn’t provide the necessary output, we were going offload, plus the major problem was of the flicker which i was facing. So i solved this problem by taking the running average of the last frames, paying 50 percent importance to the current frame. For that i created a class called Line, which represented one of the lines (i.e. left or right), and we were averaging from last averaged fits. Further for the above stated reasons and sort of sanity check, i ran the naive method of fitting the polynomial as explained above for every 100th frame.

For radius of curvature, i noticed in some frames the radius was not same for the left and right line, though they didn’t varied much still i relied on taking the average of both.

The hardest part of this project was to handle the sudden change in light conditions. Though i have used HSL color space, and extracted the saturation part, plus for the gradient i have used the red color part of the input image, but still this is the area of the project which can be expanded further. As we know now a days there are various color channel options available, and we can resort to try each approach to analyse the best results. As we know the problem is with the lighting we can resort to the color channels which have lighting as one of the channel, to narrow our investigation.