Advanced Lane Finding

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.

Rubric Points

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

Writeup / README

Camera Calibration

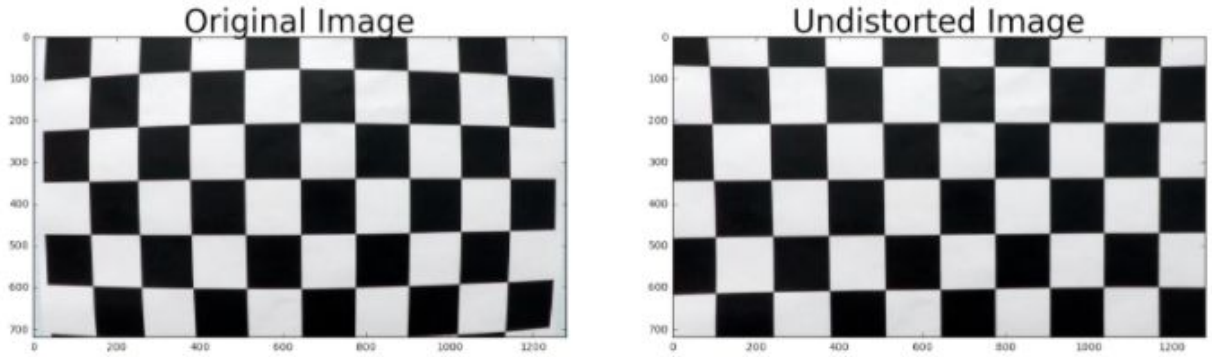
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

All the code is inside Advanced-lane-finding.ipynb

Camera calibration is contained in the first two code cells. In the first code cell we get the objpoints and imgpoints.

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`(cell 1, line 28) will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `imgpoints`(cell 1, line 29) will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.

Then in the second code cell I used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function (cell 2, line 4). I applied this distortion correction to the test image using the `cv2.undistort()` (cell2, line 5) function and obtained this result:



Pipeline (single images)

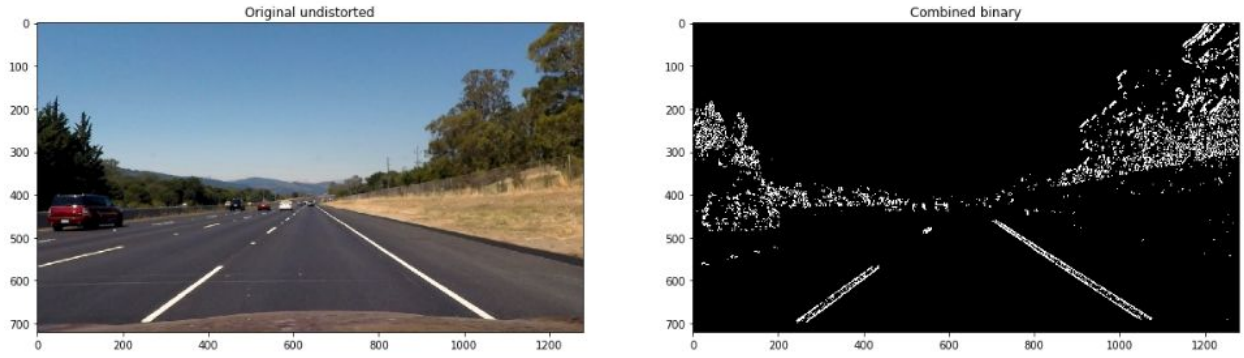
1. Provide an example of a distortion-corrected image.

In the third cell, I will undistort the example images. For undistorting I will use camera and distortion matrix. The undistorting itself happens in cell 3, line 7. One example can be seen below:

One can notice the effects of undistortion in the edges of the picture.

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.

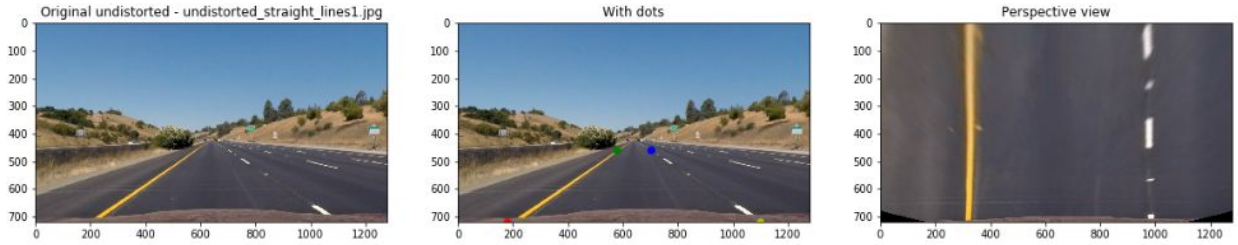
I used a combination of gradients in x and y direction, gradient magnitude and gradient direction. In addition I also thresholded the S channel in the HLS image. For there I made functions abs_sobel_thresh (cell 4, line 4-17), mag_thresh (cell 4, line 19-31), dir_threshold (cell 4, line 33-43) and HLS_threshold (cell 4, line 45-54). For finding the threshold parameters, I made a function update (cell 4, line 64-72) which was made from Ahmad Anwar's example which he posted in the Facebook group. I combined the thresholding together into a function combined_binary (cell 5, line 15-26).



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

Perspective transform was done in cell six. First I had to search for the parameters for which I drew them on the original image. The searching part was done in cell 6 lines

9-51. For warping there is a function warp_image cell 6 lines 3-7. One example can be seen here:



For warping there are two matrices:

src = np.float32([[180, 720],

[577, 460],

[703, 460],

[1100, 720]])

dst = np.float32([[300, 720],

[300, 0],

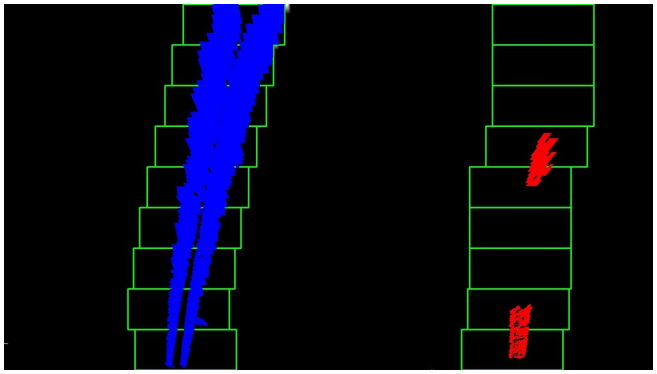
[980, 0],

[980, 720]])

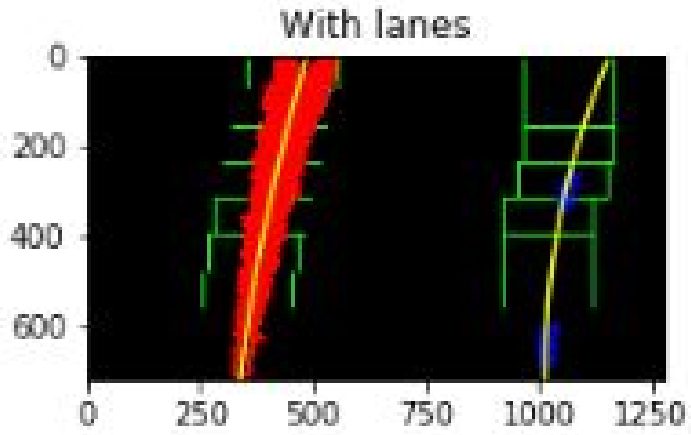
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

For identifying lane-line pixels and fitting their position with a polynomial I used the approach recommended by udacity. First I found the histogram peaks from the lower half of the binary picture (cell 7, line 21-22). Second, a sliding window search was used for identifying lane-line pixels (cell 7, line 44-65). Third, after the lane-line pixels had been identified I fit a second order polynomial to both of them (cell 7, line 81-82).

Identified lane-line pixels can be seen here:



With lanes drawn on there:



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.

Calculating the curvature and position from the center is done in cell 7 function curvature_and_distance_from_center (line 113-132). To get the curvature, I basically modified the polynomials to fit to real world distances. For calculating the distance from center I looked at the bottom of the image to see where the lanes end. Then I looked how much the center of the lanes in the bottom differs from the center and then converted that difference into meters.

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

Drawing the result back on the image was done in function draw_onto_original (cell 7, line 93-111). One example can be seen here:

Pipeline (video)

The same pipeline was used as described above. The only thing that was added was line class (cell 8). In the cell class I used last n lanes and discarded the ones where curvature was too different or where there were too few datapoints. Processing an image was done in cell 9 and video was processed in cell 10.

Here's a link to my video

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?

It seemed to me that simply averaging over the past n lanes and discarding lanes if curvature was too different or there were too few datapoints seemed to be sufficient.

There are definitely some settings where thresholding the image to get a binary image fails. It already had serious problems in some of the frames in the video. A lot of shadows or the color of the asphalt may make it perform worse.

About improvements. The code can be made a lot more efficient. For example I could use a look-ahead filter. To make it more robust, I could make more sanity checks about curvature, parallelism and use smoothing with more thing besides only using points from last n frames.