Advanced Lane Finding Writeup

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

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

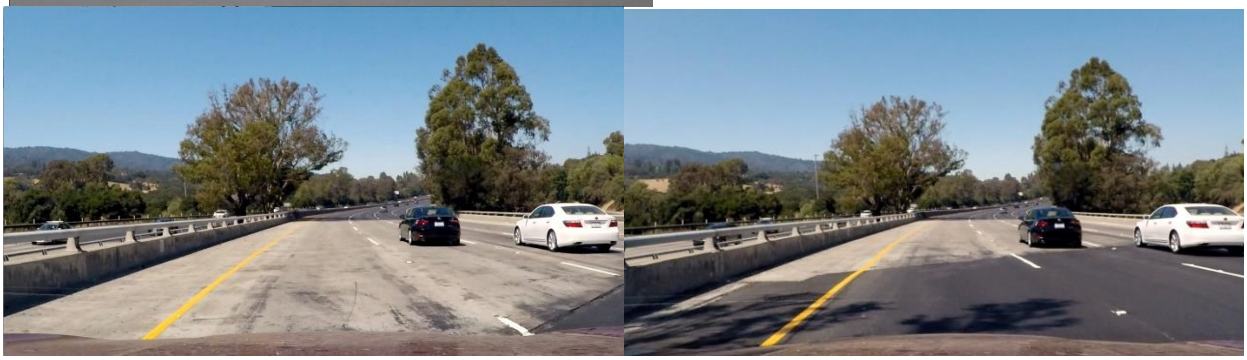
1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

The code for this step is contained in the first few code cell of the IPython notebook titled advanced_lanes.ipynb. 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.

An example of the output of the find corner function and a distorted and undistorted road image are below.



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.

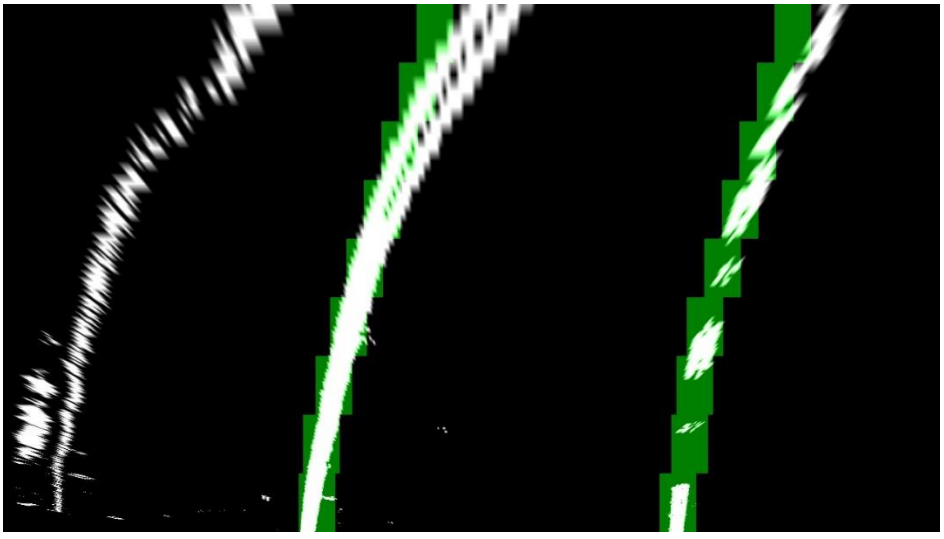
I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at lines 11 through 16 in `cell 7 in the python notebook with the process_test_images function.`). Here's an example of my output for this step.

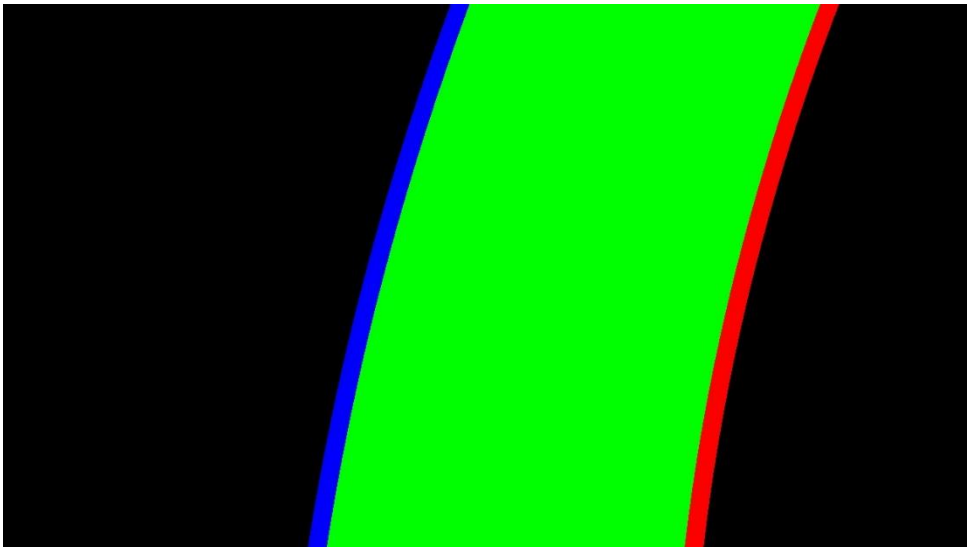
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform takes place in lines which appears in lines 22 through 39 in the 7th code cell of the python notebook. The `warper()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I verified that my perspective transform was working by outputting the transformed binary image. The lines should be parallel with equal distance at the bottom an top of the image to show that it is working. An example of the transformed binary image is below.

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

In order to identify the lane pixels I created a line tracker class using the sliding window technique which averages the pixel locations at the left and right lanes using convolutions then moves "slides" the window a set "margin" distance horizontally with each vertical window. An output of this process can be seen below. Then using the values found in the windows I used polyfit to fit a 2^{nd} order polynomial to the lanes. The line tracker class can be found in code cell 6 and the polyfit can be found in lines 41 to 77 of code cell 7.



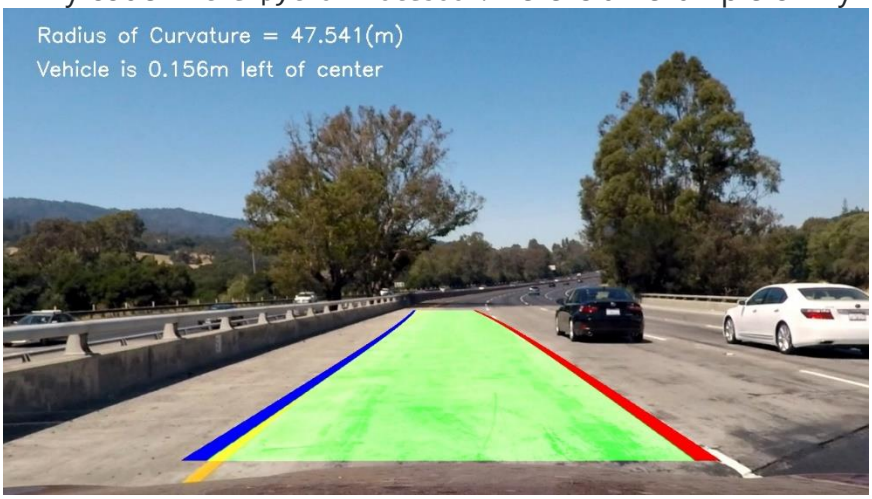


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

The radius of curvature was calculated using in code cell 7 of the python notebook in lines 118 to 133. The radius of curvature was calculated by using mathematics provided in the lesson. The equation was calculated as shown below:
curverad = ((1 + (2*curve_fit_cr[0]*yvals[-1] * ym_per_pix +curve_fit_cr[1])**2)**1.5)
/np.absolute(2 * curve_fit_cr[0])

The curvature and position of the vehicle were then both added to the image.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly. I implemented this step in lines 79 through 116 in my code in `the python notebook`. Here is an example of my result on a test image:



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?

The binary image that was created could be improved. Shadows are showing up strongly in the binary image and causing the line fit to falter. There would also probably be a problem if there was a car in the lane in front of the camera as it would show up in the binary image and mess up the lane lines. There would be problem if there were unclear lane markings or weather obscures the road.

The window sliding could be improved upon. The current margin does not allow for sharp lane turns to be captured, however when the margin was increased the line was more likely to respond to noise and jump around wildly. There is also a problem where when there are dashed lines and there is a gap in the lane the window does not find anything and will default by jump to the left the full value of the margin. This could be solved simply by telling the window to default to the same horizontal value as the previous window if no values are found or by continuing until it finds the lane again at future windows and going back to the previous windows to create a smooth line between existing values.

The transform would probably not be accurate if road was sloped up or down as perspective changes. The solution to this problem is complex and would probably need a dynamic solution that first reads the slope of the road and the drivable area then calculates suitable transform values.