Advanced Lane Finding

Mohit Pandey

2018
January

Chapter 1

Introduction

Finding Lane Lines on the Road

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.

Chapter 2

Writeup/Readme

2.1 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. Project code is at *https://github.com/diamondspark/Advanced-Lane-Finding.git*

Chapter 3

Camera Calibration

3.1 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 IPython notebook located in "project-advanced-lanes.ipynb" (Cell# 2). I start by preparing "object points" on 9x6 chessboard images, 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:

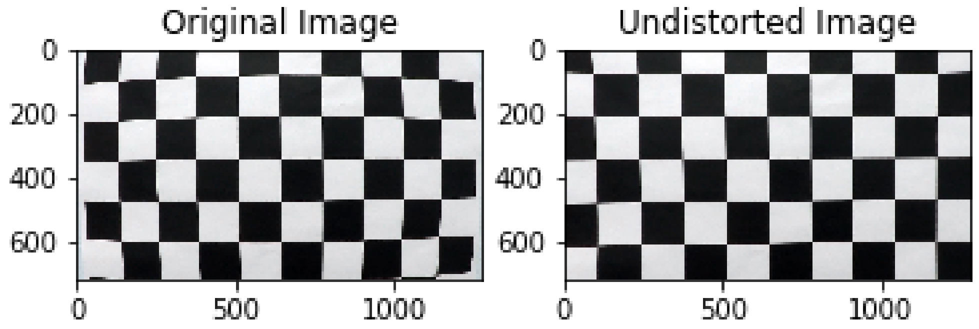

Figure 3.1: Sample Undistorted Chessboard

Chapter 4

Pipeline (single images)

4.1 Provide an example of a distortion-corrected image.

The output of this step will be an undistorted image. In future step we will do perspective transform on the undistorted image. Here is the output of the same.

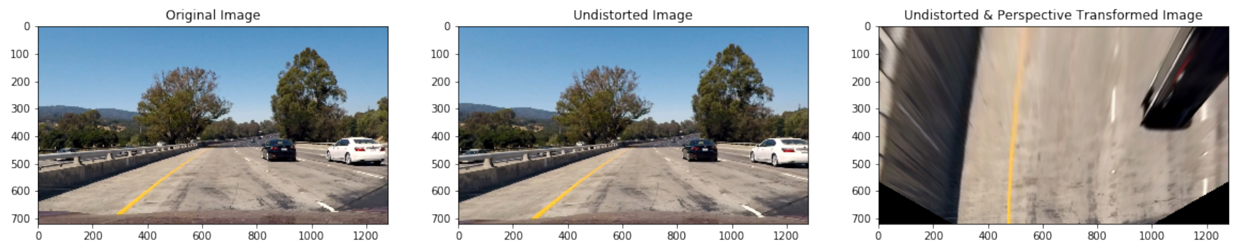

Figure 4.1: Undistorted and Perspective Corrected Image

4.2 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. These experiments are duly labelled in jupyter notebook in Cell#5,6 and 7. Here are some of the outputs from the said experiments.

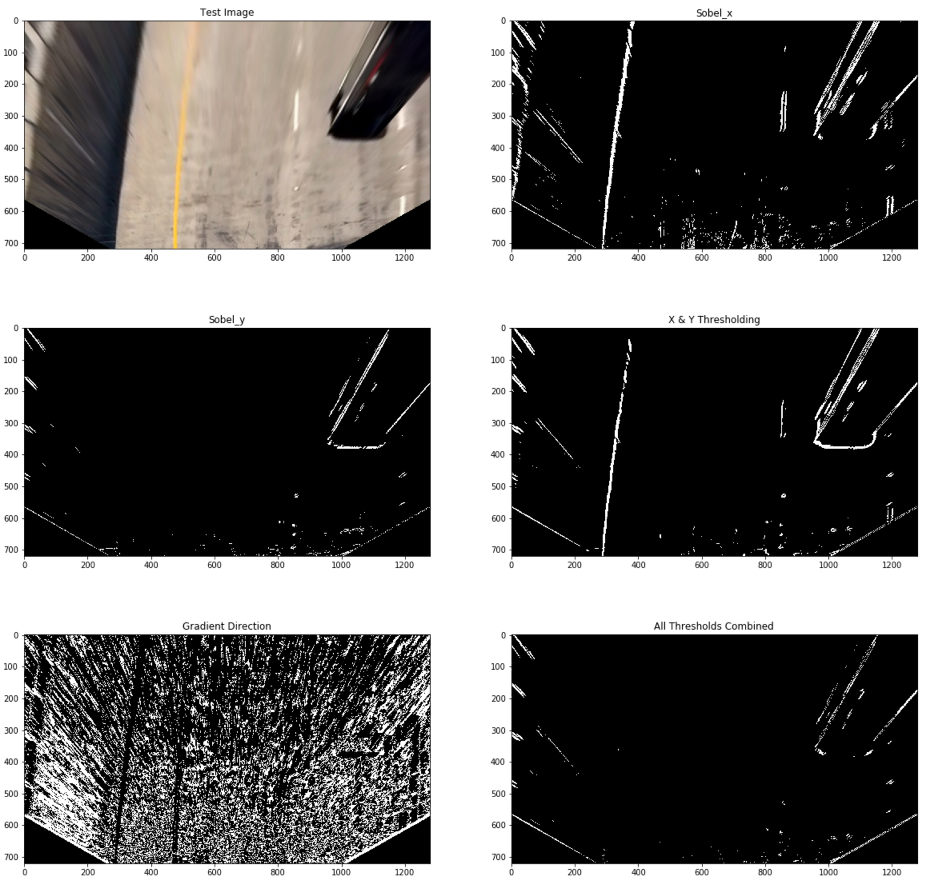

Figure 4.2: Different experiments with edge detection

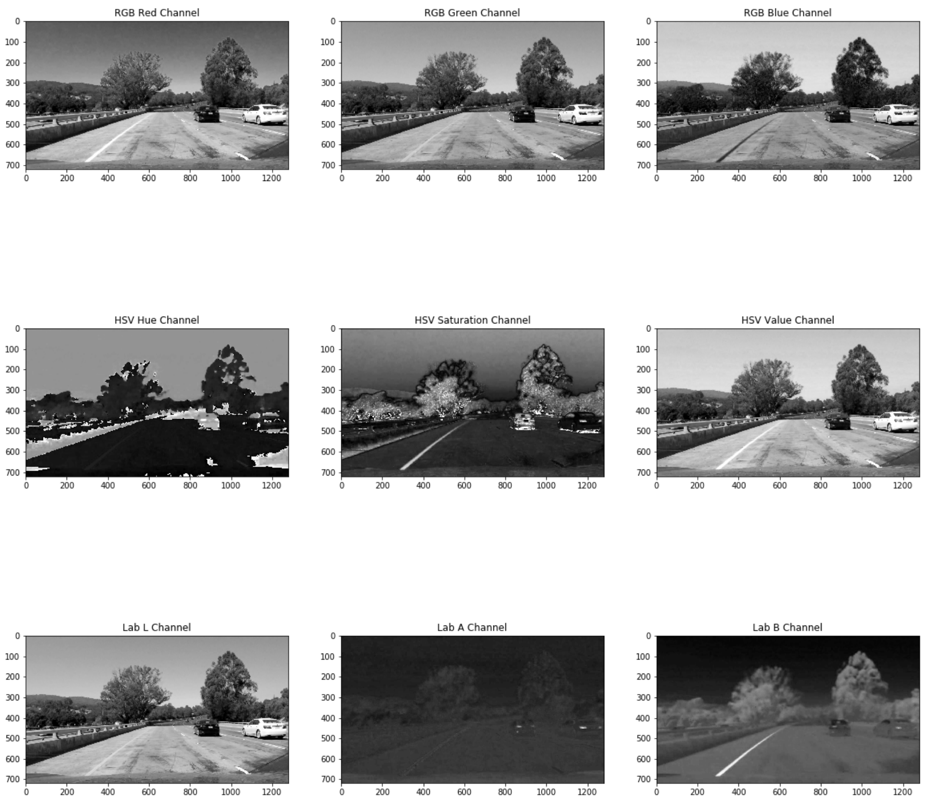

Figure 4.3: Different experiments with Color Spaces

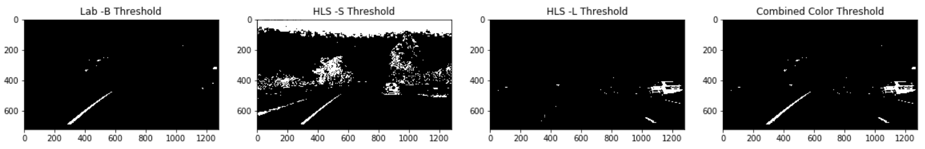

Figure 4.4: Outputs of Color Thresholding

4.3 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 includes a function called *warp()*, which appears in Cell#4 of jupyter notebook. The warper() function takes as inputs an image (img), as well as source (src) and destination (dst) points. I chose the hardcode the source and destination points in the following manner:
src = np.float32([(296,634),(995,634),(716,461),(559,461)])
dst = np.float32([(440,720),(840,720),(840,0),(440,0)])

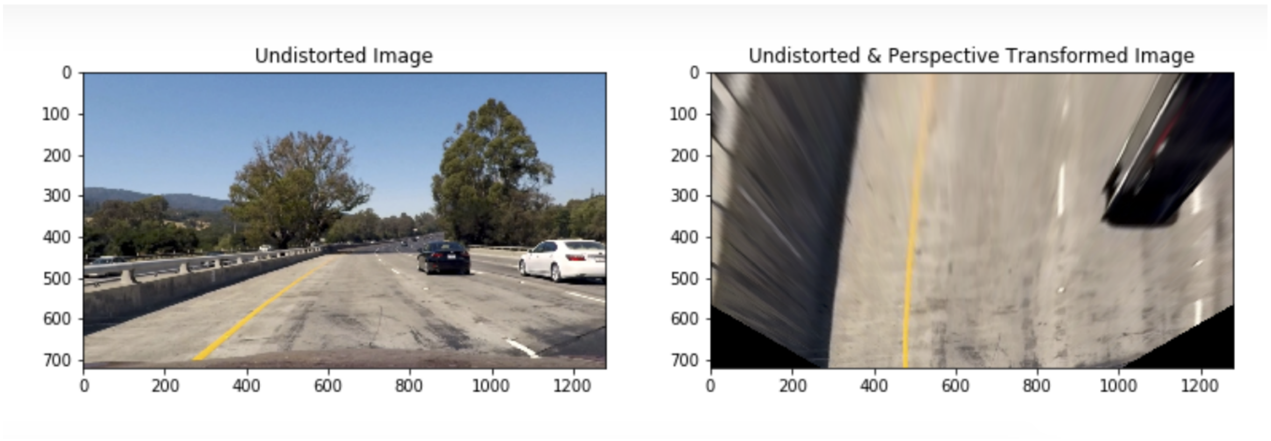

Figure 4.5: Output of perspective correction after selection of ROI

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

Fitting a second order polynomial was performed in functions *sliding_windows* and *polyfit*. This is well commented in Cell#11 and 13 respectively of the jupyter notebook.
It helps immensely to calculate a histogram on the binary thresholded image to get a tentative location of edge (lane lines).

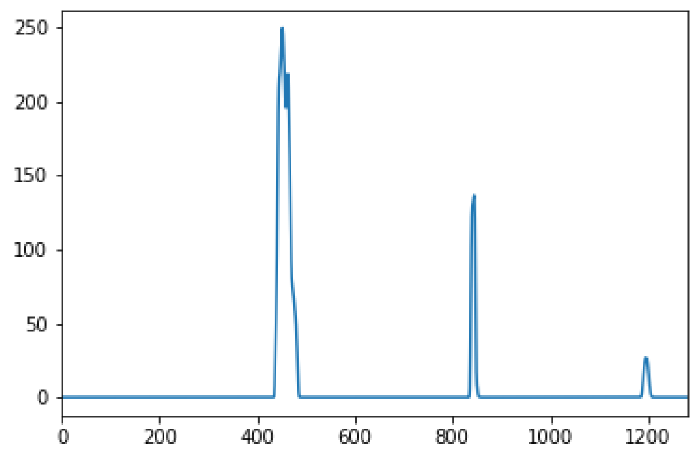

Figure 4.6: Histogram of thresholded Test Image

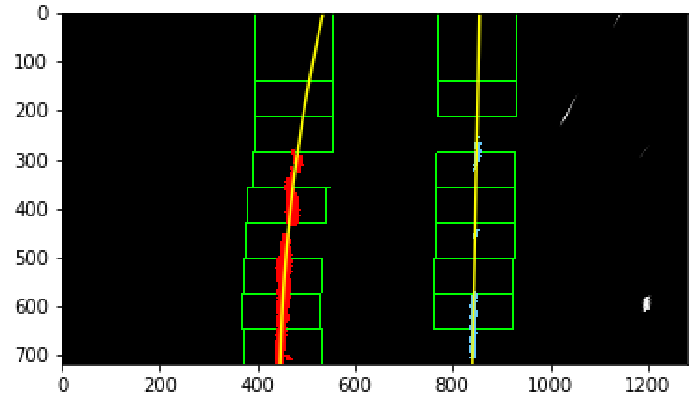

Figure 4.7: Output of sliding windows() on Test Image

The sliding window method attempts to find the base of left lane and the right lane in the image. To do so I modify the method provided in Udacity coursework. We look to fit 10 windows in each quarter of the histogram generated. This essentially makes blocks across the lane lines. We then use numpy's polyfit() method to fit a second order polynominal.

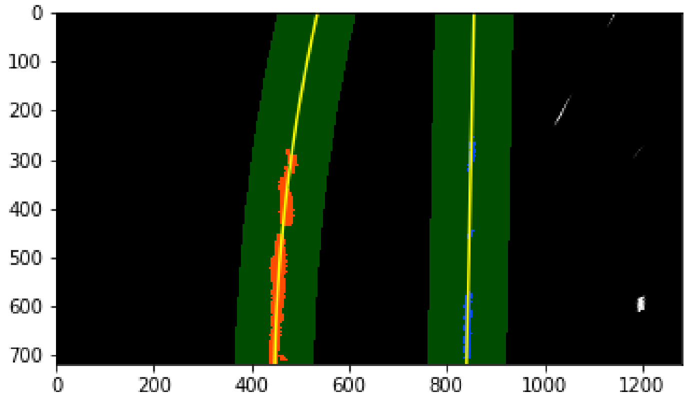

Figure 4.8: Output of Polyfit on Test Image

Our polyfit method looks for lane lines within a small range of the lane detected in previous frame.

4.5 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.

In the method calc_curv_rad_and_center_dist in the Cell#14 of the jupyter notebook we calculate the ROC and Center offset for the vehicle.
From the lecture notes, Radius of curvature is defined as

$$R_{curve} = \frac{[1 + (\frac{dx}{dy})^2]^{\frac{3}{2}}}{|\frac{d^2x}{dy^2}|} \tag{4.1}$$

This was implemented as follows
curve_radius $= ((1+(2*fit[0]*y\_0*y\_meters\_per\_pixel+fit[1])**2)**1.5)/np.absolute(2*fit[0])$
Since we are working with pixels and pixel distance as a measure of ROC doesn't make physical sense, we use *y_meters_per_pixel* as a factor to convert from pixels to meters.

We then calculate position of car wrt lane center like following
lane_center_position = (r_fit_x_int + l_fit_x_int) /2
center_dist = (car_position - lane_center_position) * x_meters_per_pix

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

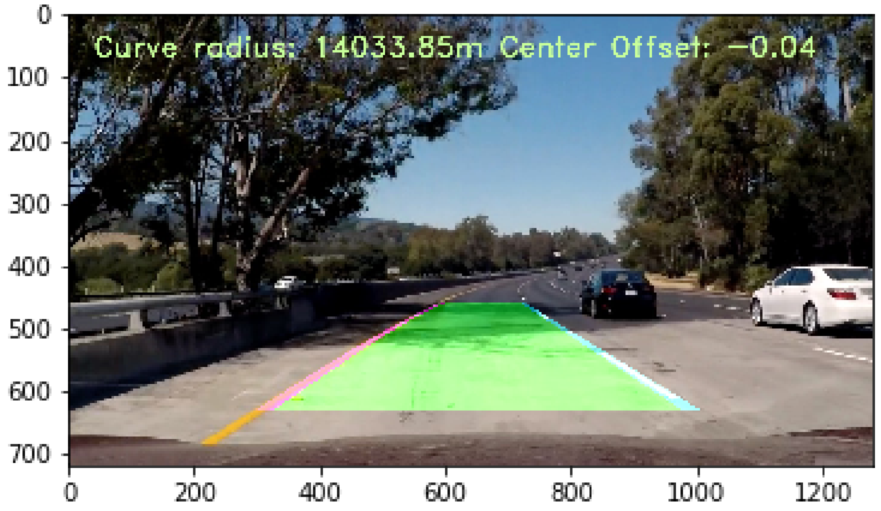

Figure 4.9: Radius of Curvature, Lane Marking, Dist from Center on Test Image

Chapter 5

Pipeline (video)

5.1 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!).

The video can be found at this link.
https://github.com/diamondspark/Advanced-Lane-Finding/blob/master/project_video_output.mp4

Chapter 6

Discussion

6.1 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 most significant problem to begin with was to deal with different lighting conditions, shadows and decolorization on the road. We tried to solve this by trying different color spaces. A mix of HSV and LAB color space solved this problem for us.
Another problem is that we hard-coded the ROI. This is never a good solution. If the camera is mounted on a SUV vs a sedan, we will have different height and hence our ROI shouldn't be the same.
One future work to make the pipeline robust is to make a dynamic iterative thresholding system that can learn over time the best parameters to thresholding parameters.