# 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

### Camera Calibration

#### 1. 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 in the file `camera\_cal.py `

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

objp

objpoints

the same for each calibration image. Thus,

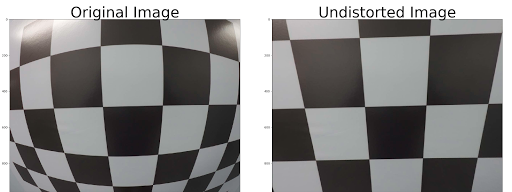
is just a replicated array of coordinates, and

will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. will be appended with the (x, y) pixel position of each of the corners in the image plane with

imgpoints

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 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):**

#### Has the distortion correction been correctly applied to each image?

I used the camera matrix and the distortion coefficients obtained in the previous step and used the “cv2.undistort” function to undistort images in the video stream. I used pickle to save the values from the previous step and loaded them for use in the pipeline.

|  |  |
| --- | --- |
|  |  |
|  |  |

#### 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.

The code used to experiment with color, gradients, and thresholds could be found in the image\_gen.py. For the limits I chose a combination if Sobel-X, Sobel-Y, magnitude & directional gradient thresholds and S-channel HLS color transform gave the best results.

#### Has a perspective transform been applied to rectify the image?

The code for my perspective transform is includes a function called warper() , which appears in lines 1 through 8 in the file example.py (output\_images/examples/example.py) (or, for example, in the 3rd code cell of the IPython 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(

[[(img\_size[0] / 2)

[((img\_size[0] / 6) [(img\_size[0] \* 5 / [(img\_size[0] / 2 +

= np.float32(

- 55, img\_size[1] / 2 + 100],

- 10), img\_size[1]],

6) + 60, img\_size[1]],

55), img\_size[1] / 2 + 100]])

dst

[[(img\_size[0] / 4), 0],

[(img\_size[0] / 4), img\_size[1]],

[(img\_size[0] \* 3 / 4), img\_size[1]],

[(img\_size[0] \* 3 / 4), 0]])

This resulted in the following source and destination points:

|  |  |
| --- | --- |
| **Source** | **Destination** |
| 585, 460 | 320, 0 |
| 203, 720 | 320, 720 |
| 1127, 720 | 960, 720 |
| 695, 460 | 960, 0 |

I verified that my perspective transform was working as expected by drawing the

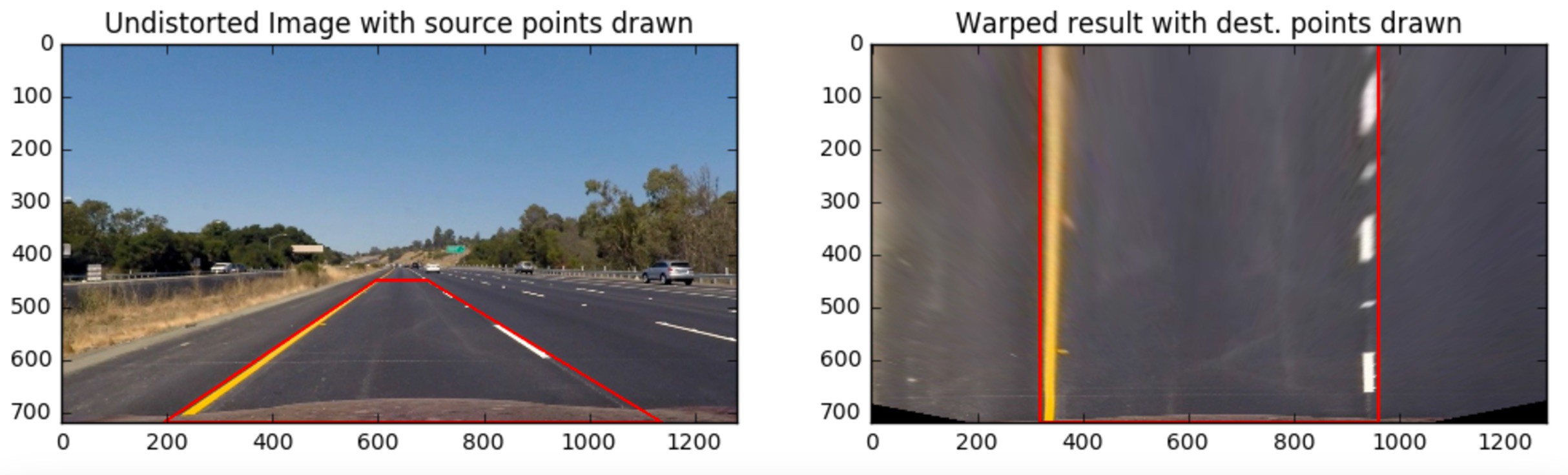
and

points onto

a test image and its warped counterpart to verify that the lines appear parallel in the warped image.

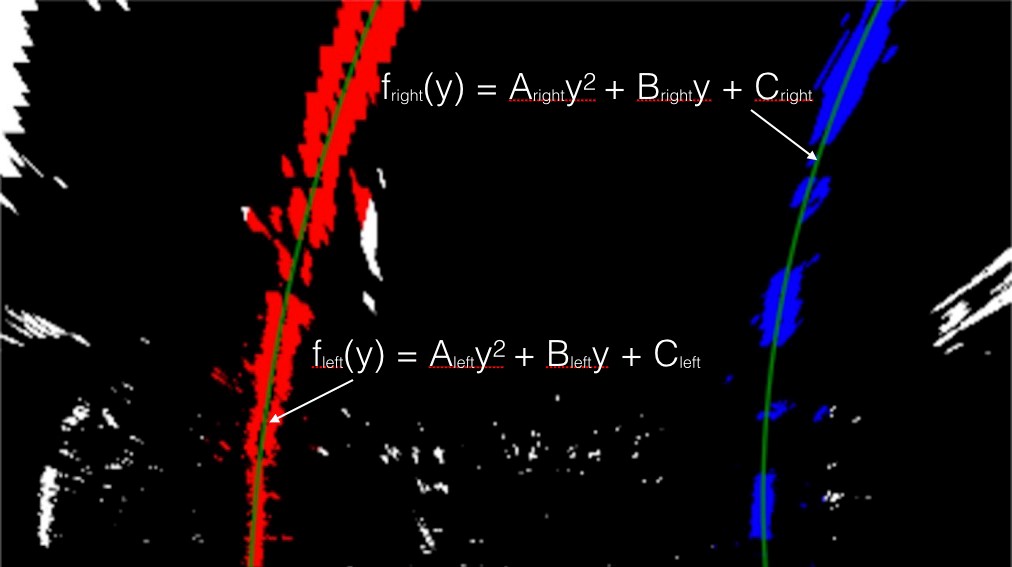
src

dst



#### Have lane line pixels been identified in the rectified image and fit with a polynomial?

Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:



#### Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

Yep, sure did!



**Pipeline (video)**

#### 1. Does the pipeline established with the test images work to process the video?

It sure does! Here's a link to my video result



## README

#### 1. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?

You're reading it!



# Discussion

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.