

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

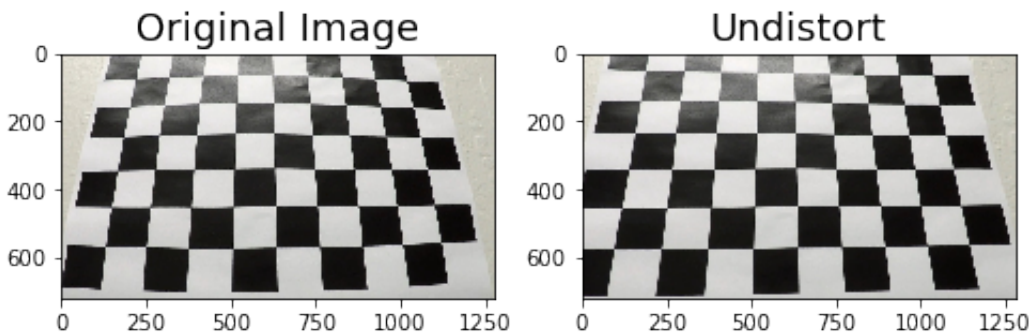
You're reading it!

Camera Calibration

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

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:

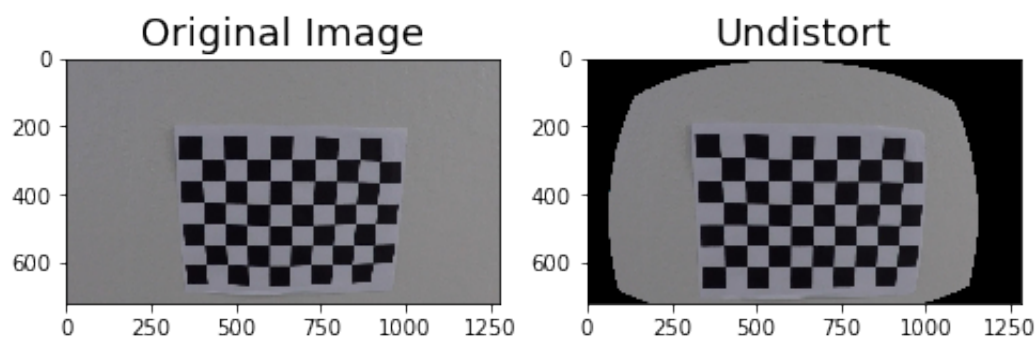


See more picture in Jupyter notebook.

Pipeline (single images)

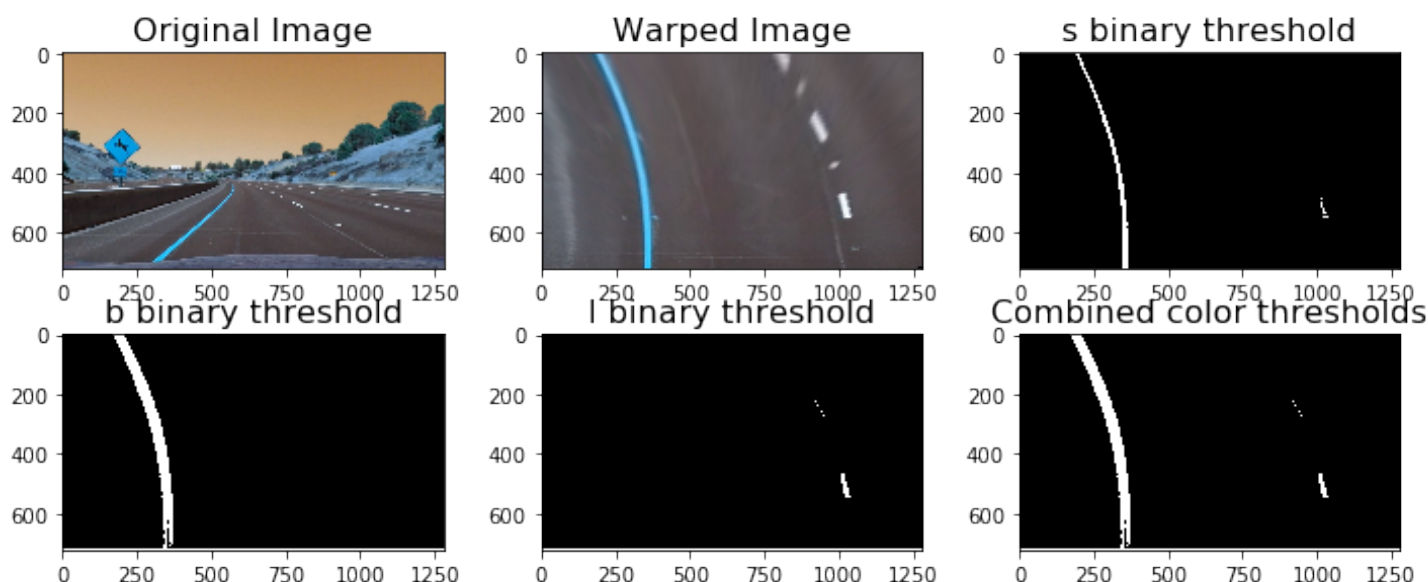
1. Provide an example of a distortion-corrected image.

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 . Here's an example of my output for this step.

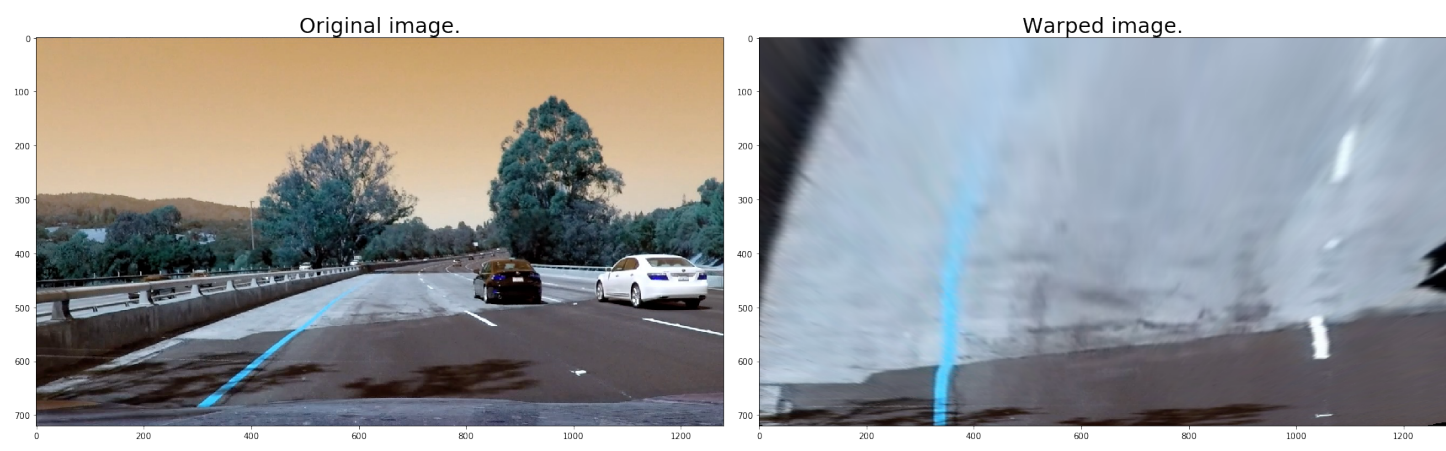


3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform includes a function called `warper_image()` , 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_image()` 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:

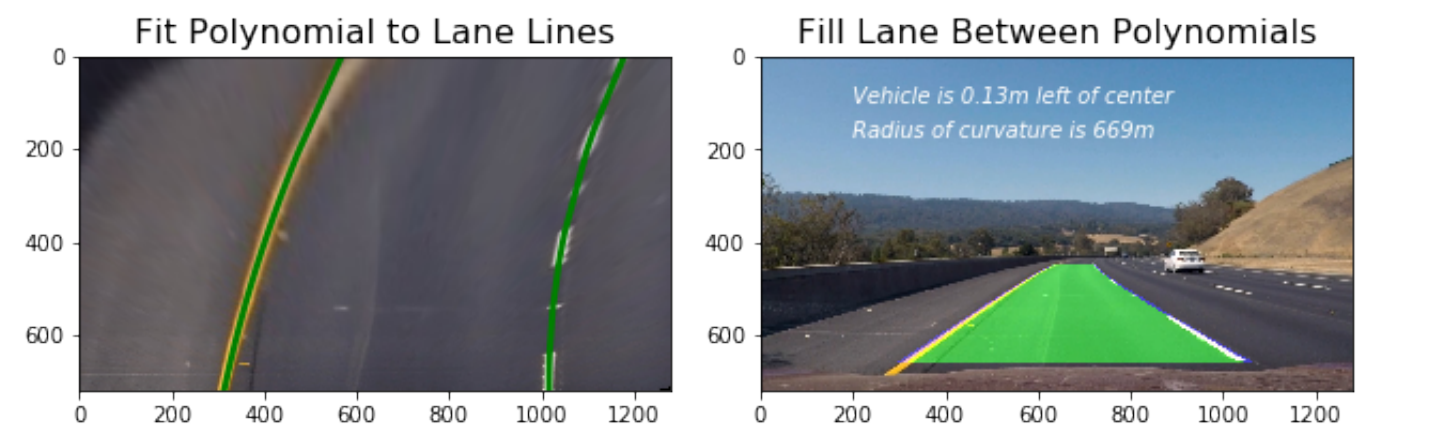
```
src = np.float32([[590,450],[695,450],[300,660],[1030,660]])
dst = np.float32([[300,0],[1000,0],[300,720],[1000,720]])
```

I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



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

I use the combined image function to get isolated lane line, then I fit a polynomial to both of the lane line. Then I calculate the radius of curvature. In the function `fill_lane()`, lane lines are detected by identifying peaks in a histogram of the image and detecting nonzero pixels in close proximity to the peaks. I summary a function by `pipeline()` for convenience.



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.

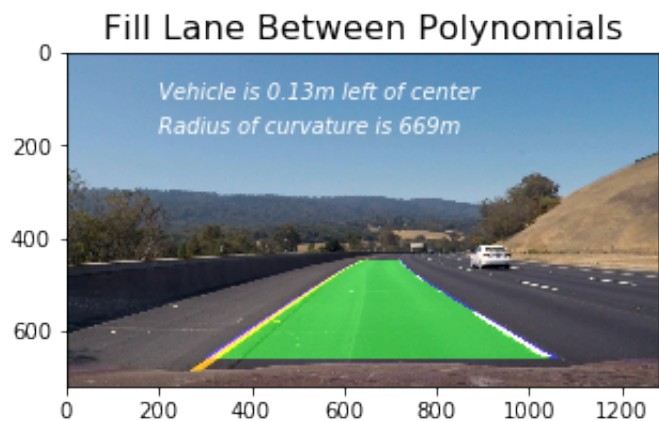
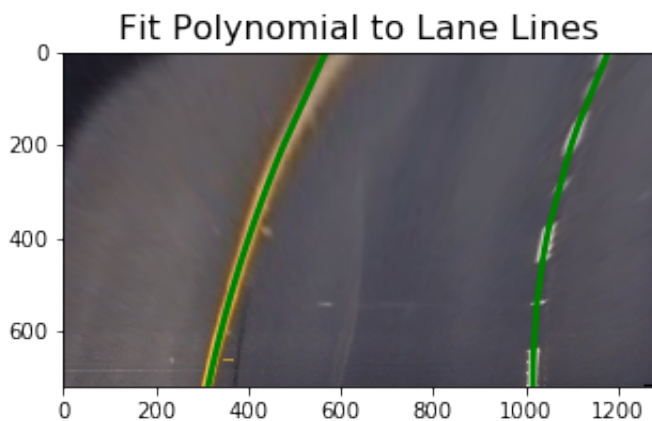
I trans the pix into meter by 2 constant `xm_per_pix` & `ym_per_pix` and the `curverad` calculated after the `polyfit` function be confirmed, and the `curverad` calculated by the follow function:

$$\rho = \frac{(1 + y'^2)^{\frac{3}{2}}}{|y''|}$$

```
# Measure Radius of Curvature for each lane line
ym_per_pix = 30./720 # meters per pixel in y dimension
xm_per_pix = 3.7/700 # meteres per pixel in x dimension
left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)
left_curverad = ((1 + (2*left_fit_cr[0]*np.max(lefty*ym_per_pix)/2 +
left_fit_cr[1])**2)**1.5) \
/np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*np.max(lefty*ym_per_pix)/2 +
right_fit_cr[1])**2)**1.5) \
/np.absolute(2*right_fit_cr[0])
```

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

see Jupyter Notebook



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?

Where will your pipeline likely fail?

When I late time submit the project, my final video is distorted a lot. I check the code and found that `result = cv2.addWeighted(image, 1, newwarp, 0.5, 0)` make a wrong to use as `result = cv2.addWeighted(undist, 1, newwarp, 0.5, 0)`, so I change back to the right one.

When I tried on challenge and harder_challenge_video, and find it doesn't work well, especially when it met the wall and the shadow, it will be mistaken as lane. Later I will take more HUE factor into consideration as the color of lane line could only be yellow and white.

What could you do to make it more robust?

After I submit this project last time, reviewer share a good reference to me which make me know how to make a more smoothly pipeline. I would like to read this paper and evolution my pipeline to much robust.

Here is a very detailed paper about it:

[paper](#)

9/30 Update

Correct the camera calibration to make the wraped image normal.