Writeup Template

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!

Source Files for Referece

- Advanced*Lane*Finding.py
- Advanced*Lane*Finding.ipynb

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 third code cell of the IPython notebook located in "./Advanced*Lane*Finding.ipynb" (or in lines 51 through 78 of the file called `Advanced_Lane_Finding.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 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.

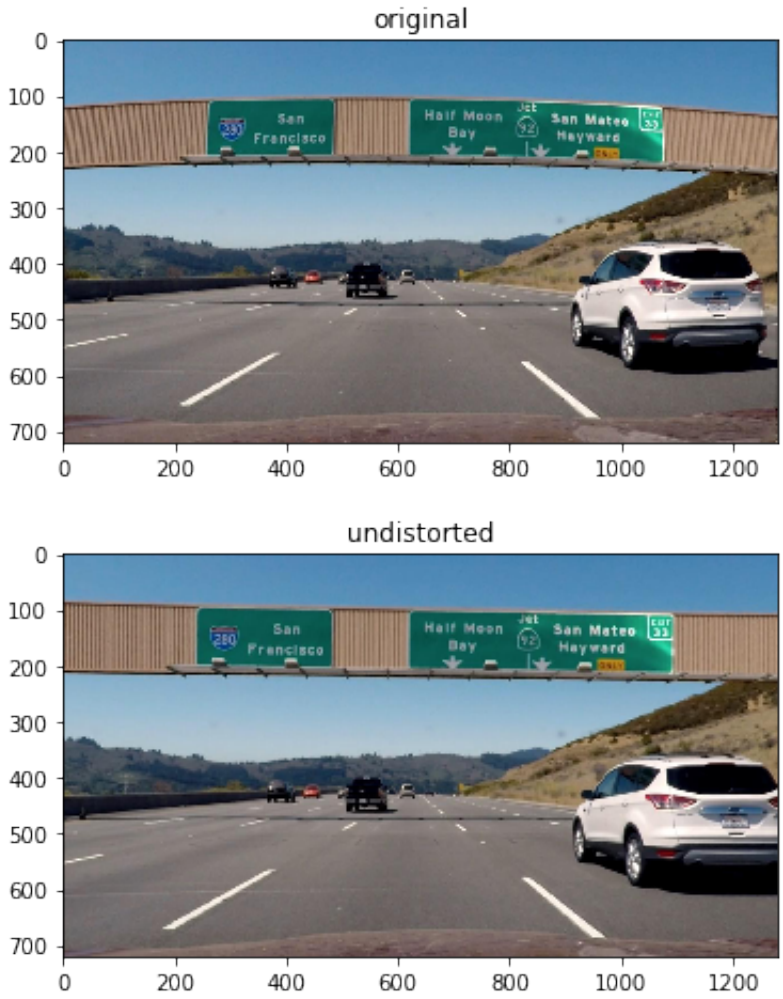
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function to produce undistorted images.

The result of applying threshold is shown in next section, after the dicsussion on perspective transformation. Note that the calibration parameters were stored to disk (calibration_params.p) so that this step does not have to be repeated.

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 (thresholding steps at **lines 156 through 168** in `Advanced_Lane_Finding.py`).

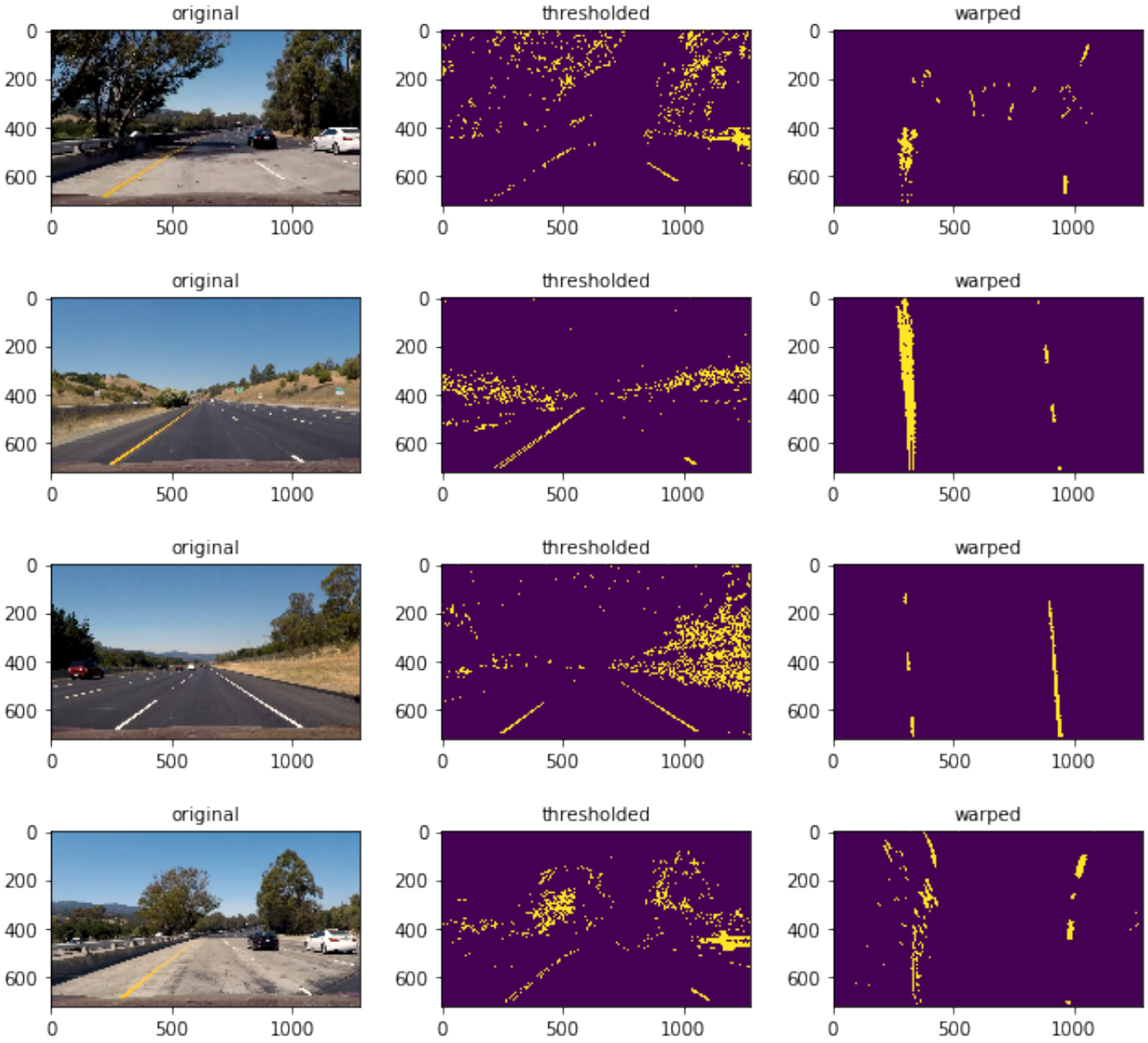
```
combined[ (black_threshold_img == 0) & (color_threshold_img == 1) & (gradx_binary == 1) | (white_threshold_img == 1)] = 1
```

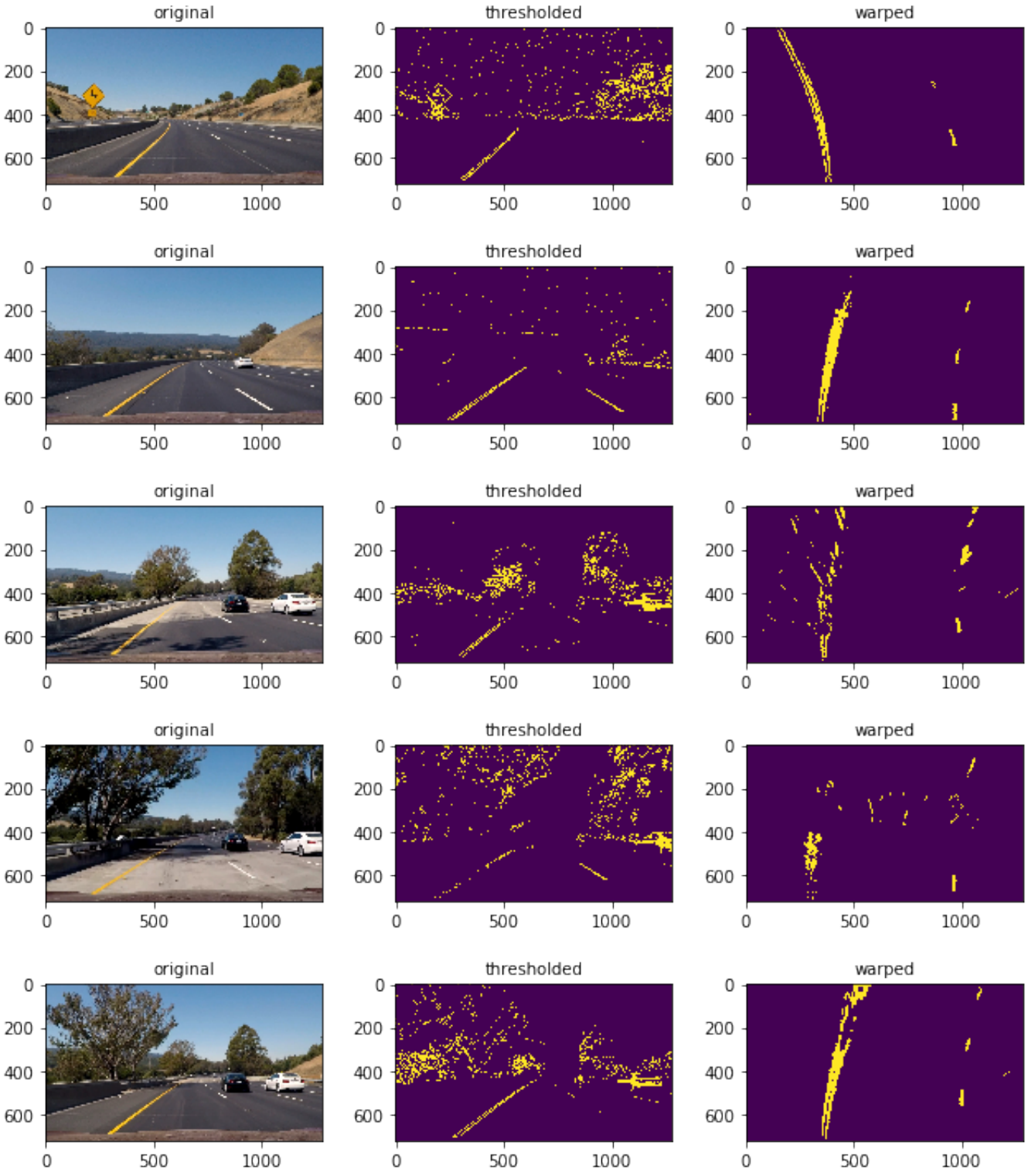
The lane pixels can be projected by applying S channel threshold (from HLS image) as the pixels are expected to be dark. Gradient (along x) allowed filtering for pixels that form near-vertical lines. However this was not sufficient as shadows from trees were dark and satsified the S channel thresholds. Moreover some of the white lane portions were not dark enough. To retrieve the white lane pixels and eliminate shadow pixels, additional threholds on white and black color were applied.

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 is contained in the method warp_perspective from the class Pipeline (**lines 444 to 453** of the file `Advanced_Lane_Finding.py`

The following shows the stages (of the pre-processing pipeline) being applied to each example test image. The original image is first subjected to thresholding (thresholded) followed by perspective transformation (warped).



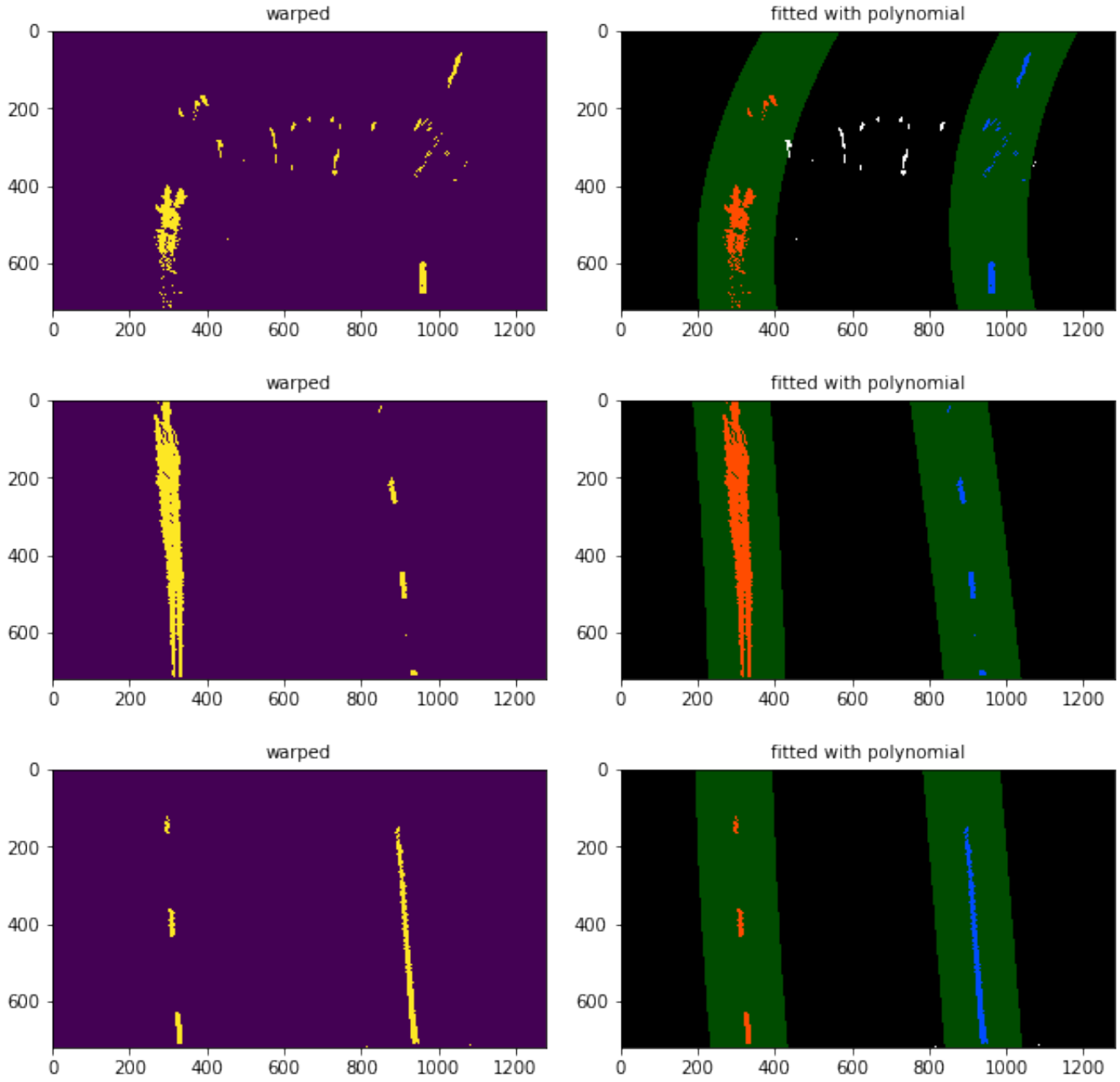


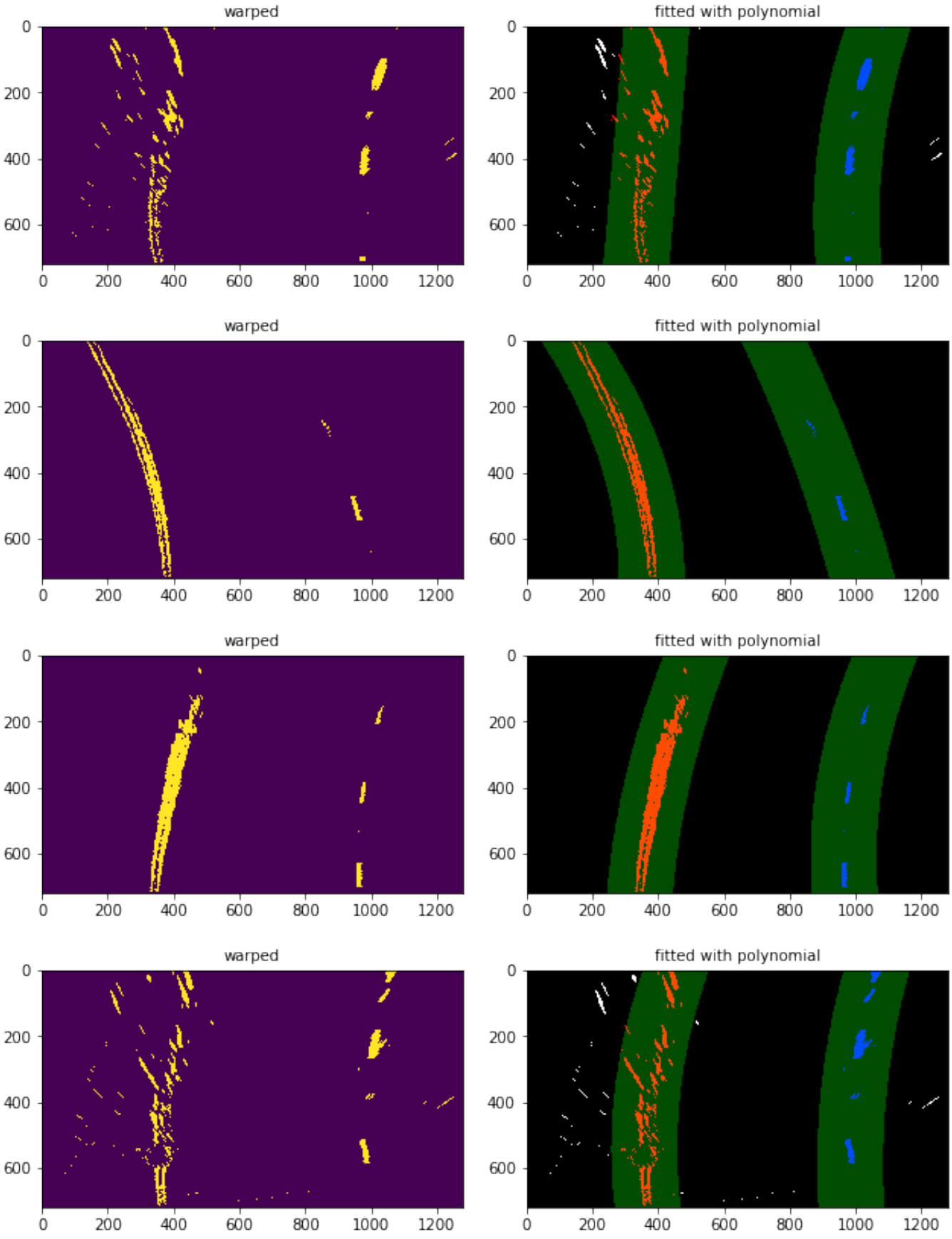
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

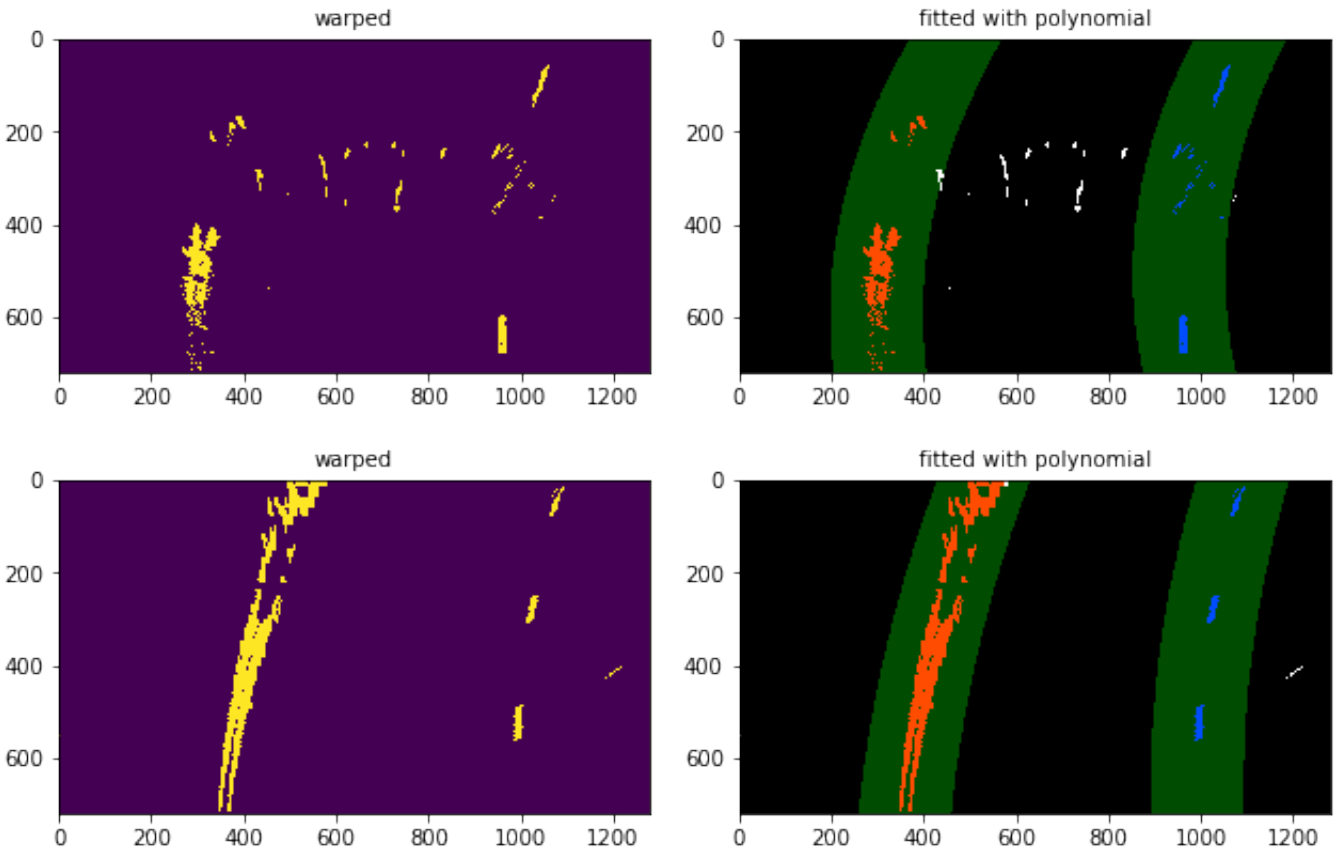
The logic for detecting lanes is contained within **lines 455 to 467**. Initially, the pipeline does not have prior information about the location of lanes. It uses sliding window search (**lines 171 to 246**) to detect the left and

right lanes. We do not need to repeat the sliding window algorithm over the complete image, on each consecutive image frame from the given video. Instead if the lanes from previous image are known, then we restrict the search to the vicinity of the previously detected lanes (**lines 249 to 291**). However, it may happen that the lanes may change positions significantly or change shape when a turn is being approached. Thus the lanes detected in a restricted search area are compared with the lanes from prior image frame using a similarity threshold (**lines 400 - 406**). We compare the radius of curvature of the lanes to detect changing lane orientation (or turns) and repeat a full image sliding window search if similarity doesn't hold.

The following shows the warped test images and their corresponding fitted 2nd degree polynomials.




warped
fitted with polynomial
warped
fitted with polynomial
warped
fitted with polynomial
warped
fitted with polynomial



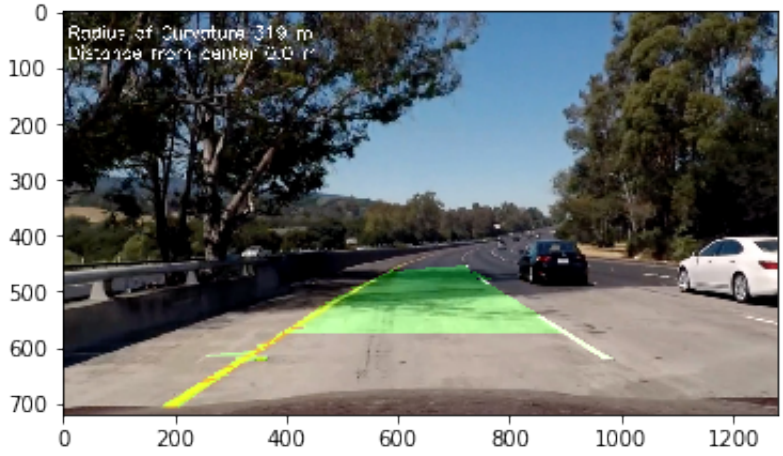
5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

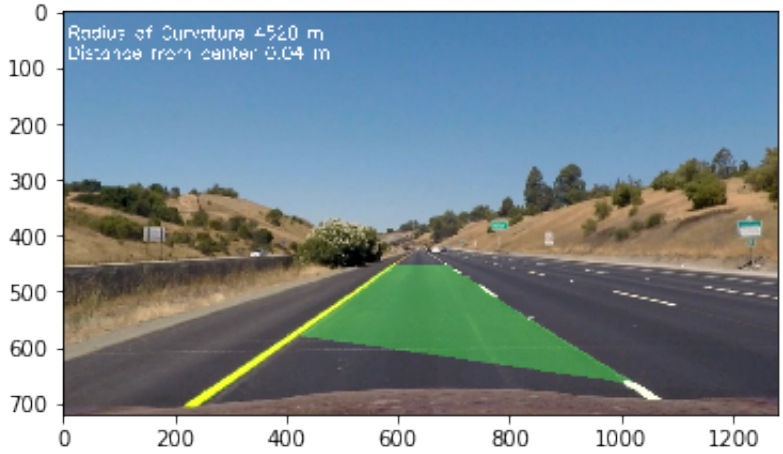
Radius of curvature was computed in the pixel space using calculus from the forumalae presented in the lecture notes. In the code, this computation can be found within **lines 395 to 397** of `Advanced_Lane_Finding.py`.

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

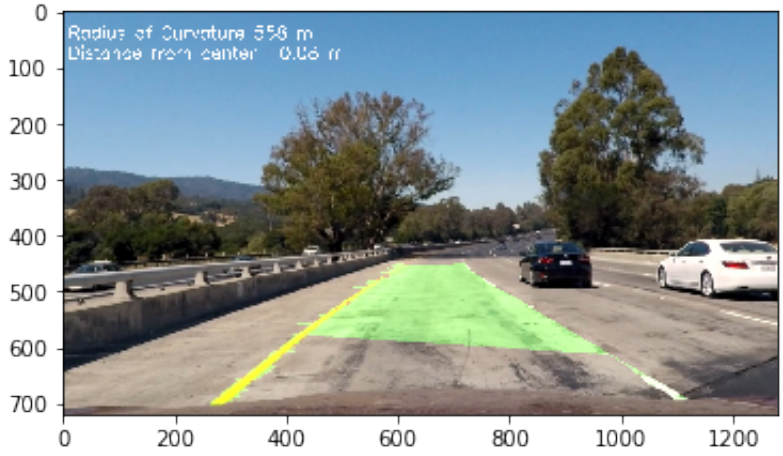
The pipeline consisted of 1. Remove distortions in input image. 2. Apply thresholds to retrieve pixels that form the lanes 3. Apply perspective transform to form a bird's eye view. 4. Fit polynomial (2nd degree) to the pixels from each lane. 5. Apply inverse of (3) to project back on the original image.

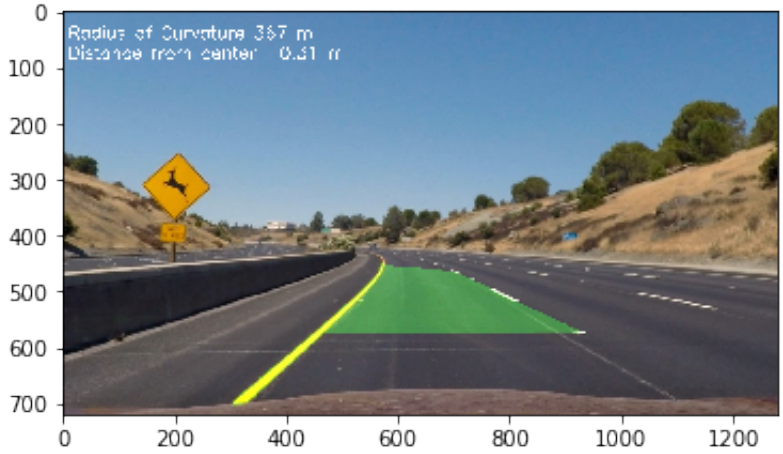
I implemented the pipeline in **lines 469 through 509** in my code in `Advanced_Lane_Finding.py` `process()`. Here are examples of the pipeline being applied to the test images:


Radius of Curvature 319 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

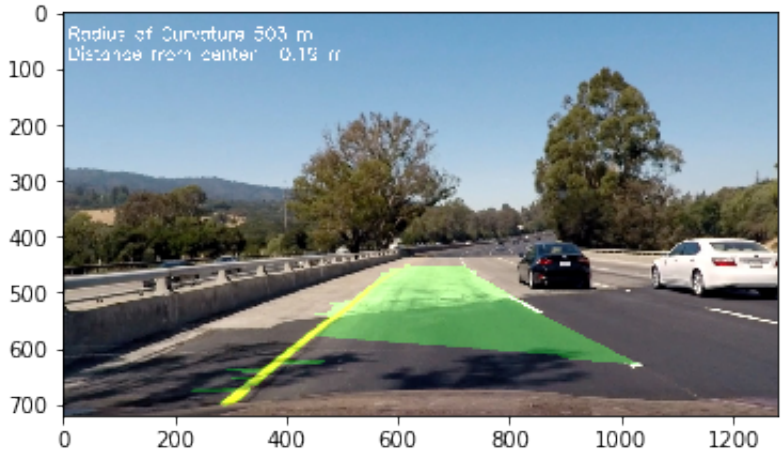

Radius of Curvature 4520 m
Distance from center 0.04 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

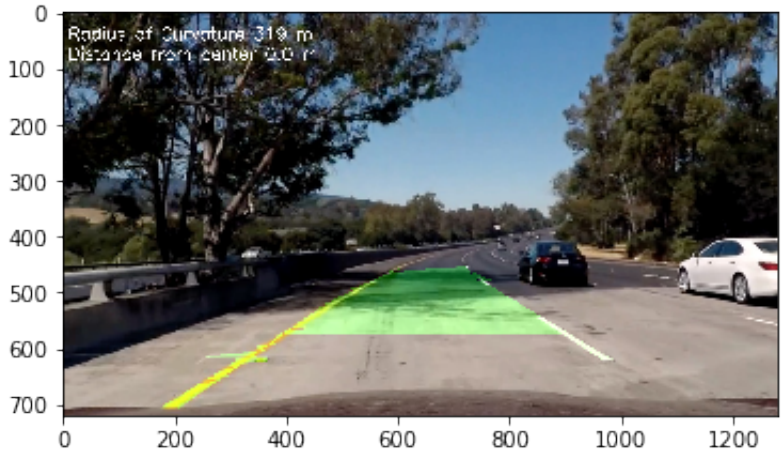

Radius of Curvature 3471 m
Distance from center 0.02 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

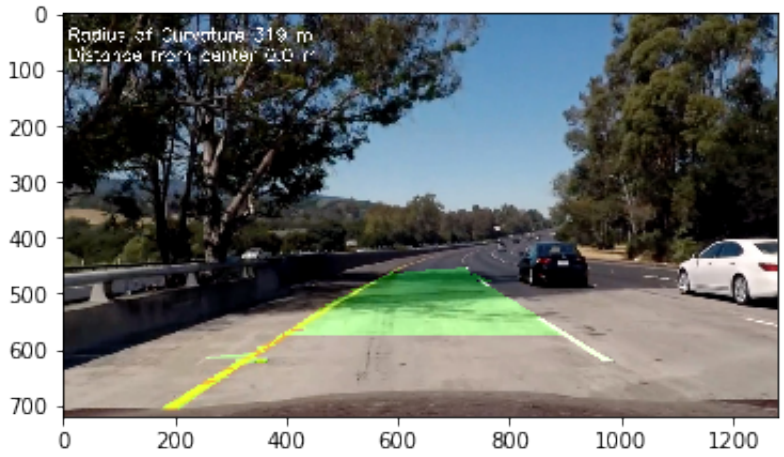

Radius of Curvature 556 m
Distance from center 0.06 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature 357 m
Distance from center 0.31 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature 457 m
Distance from center 0.06 m
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200







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?

I think the pipeline is likely to fail on hilly road where the a perspective changes significantly across frames and thus a single perspective transformation matrix cannot be applied. Instead, the tansformation matrix needs to be recomputed, whenever necessary (that is when lanes are not found to be parallel in the warped image).

Secondly, the white lanes are not present in each frame, as being discontinuos. It is possible that no lane is detected (even with sliding window search over all of image).

Thirdly, in heavy traffic conditions, the lanes may not be visible to a long enough distance or may not be visible at all. The challenge videos were taken in very light traffic conditions that allowed complete visibility of the road, which cannot be assumed.