UDACITY

‹ Back to Self-Driving Car Engineer

# Advanced Lane Finding

REVIEW

CODE REVIEW

HISTORY

## Meets Specifications

## Great Job!👏

It is obvious that you have put a lot of thought and hard work in to this project, and the results are very impressive! The pipeline does an excellent job of identifying the lane lines in the project video, and the writeup is very detailed and thorough, making your project a pleasure to review!😃

### Writeup / README

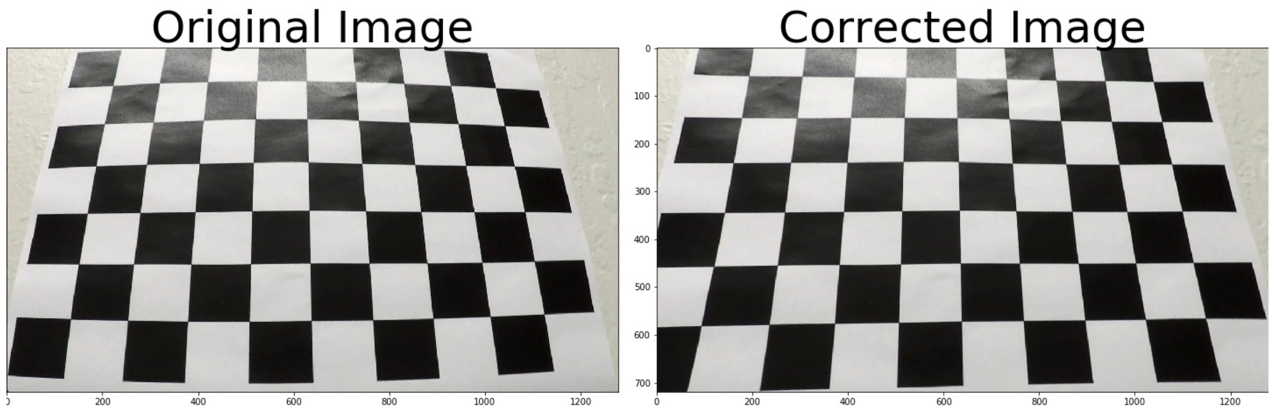
**The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.**

The writeup clearly documents the steps taken to complete the project with references to where in the code each rubric item was implemented.

### Camera Calibration

**OpenCV functions or other methods were used to calculate the correct camera matrix and distortion coefficients using the calibration chessboard images provided in the repository (note these are 9x6 chessboard images, unlike the 8x6 images used in the lesson). The distortion matrix should be used to un-distort one of the calibration images provided as a demonstration that the calibration is correct. Example of undistorted calibration image is Included in the writeup (or saved to a folder).**

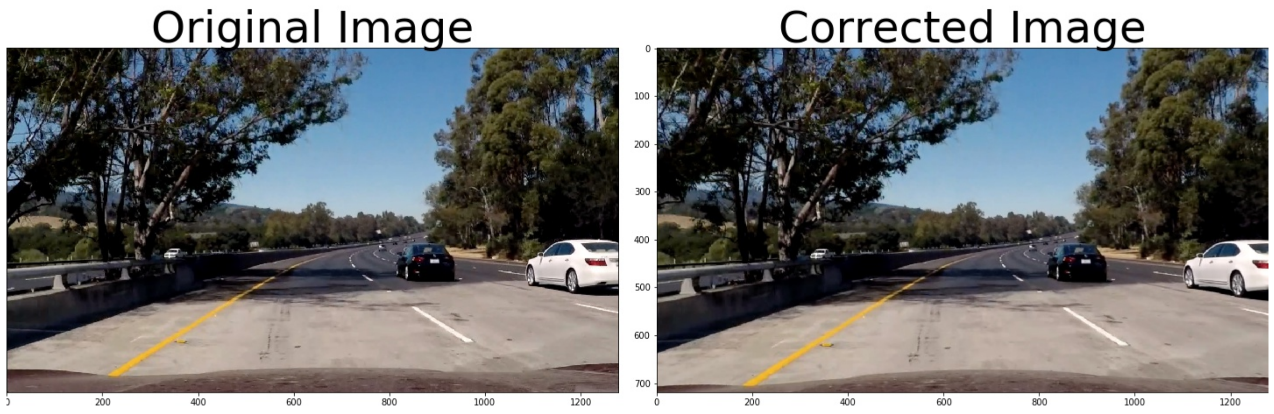
The chessboard images have been used to calculate the camera calibration matrix and distortion coefficients, and these were used to remove distortion from one of the calibration images, demonstrating the effect.



## Pipeline (test images)

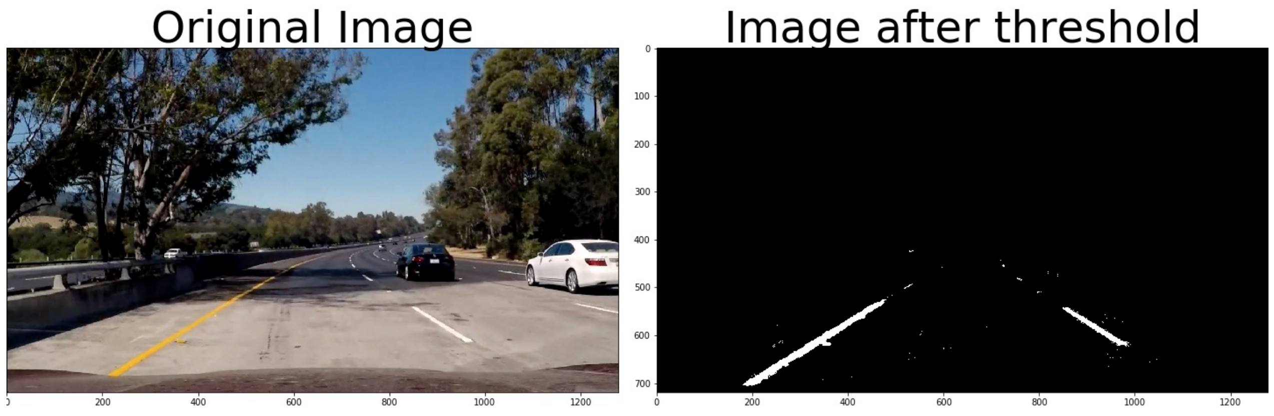
**Distortion correction that was calculated via camera calibration has been correctly applied to each image. An example of a distortion corrected image should be included in the writeup (or saved to a folder) and submitted with the project.**

The undistorted test image shows that the distortion correction has been properly applied to the road images, and I can see that this was applied in the pipeline as well. Nice work!



**A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.**

Good job applying a combination of color and gradient thresholds to create a binary image with clearly visible lane lines.

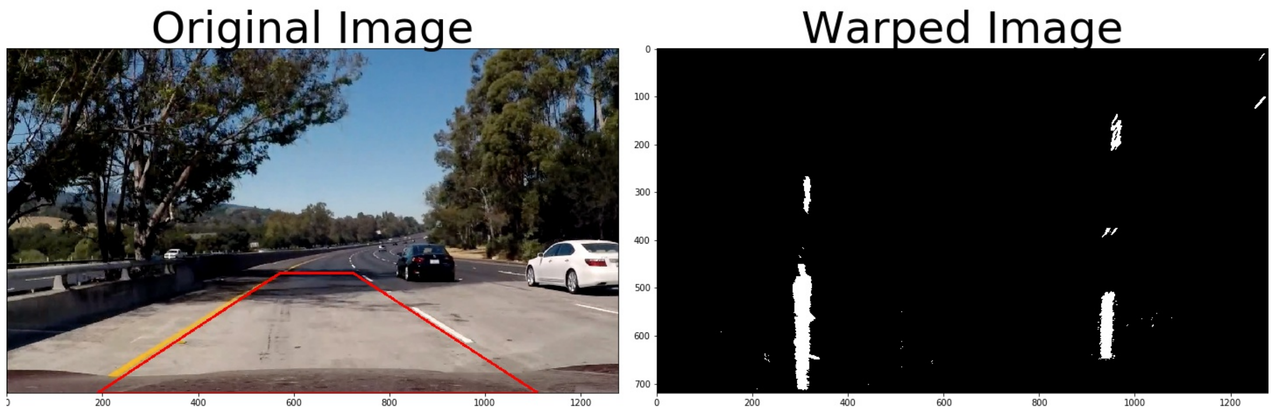


## Suggestion:

You can also try applying thresholds based on the b channel of Lab (for yellow lines) and the L channel of LUV (for white lines) which do a nice job of picking out the lane lines when there are shadows and color changes on the road.

**OpenCV function or other method has been used to correctly rectify each image to a "birds-eye view". Transformed images should be included in the writeup (or saved to a folder) and submitted with the project.**

Good job with the perspective transform, the lane lines appear very close to parallel in the warped image.



**Methods have been used to identify lane line pixels in the rectified binary image. The left and right line have been identified and fit with a curved functional form (e.g., spine or polynomial). Example images with line pixels identified and a fit overplotted should be included in the writeup (or saved to a folder) and submitted with the project.**

Nice job using a histogram and sliding window search to find the locations of the lane lines, and using the previous detections to perform a targeted search in subsequent frames.

## Suggestion:

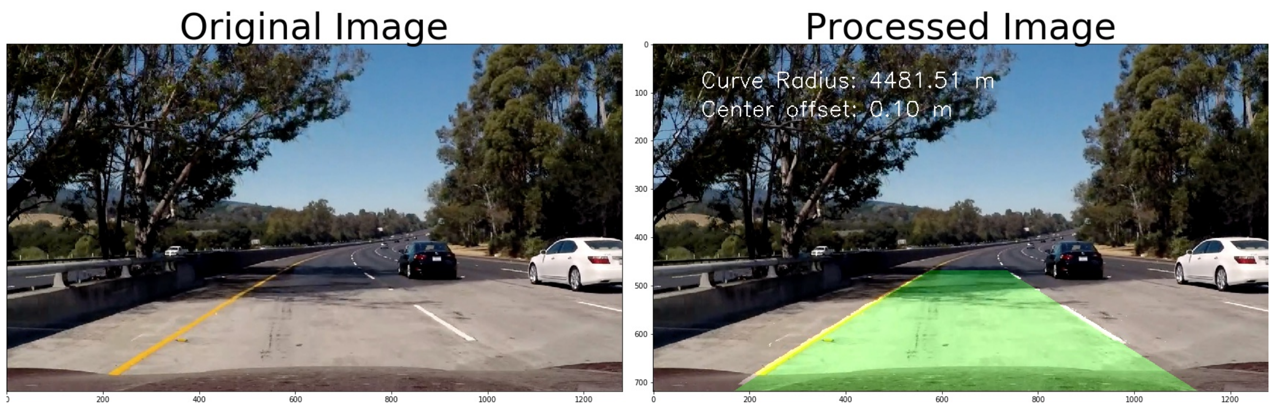
You can also try applying sanity checks in every frame by making sure that the two detected lines are the right distance apart (based on the known width of a highway lane) and that their curvatures make sense in relation to each other. If a sanity check fails in a single frame, you can reuse the confident detections from prior frames.

**Here the idea is to take the measurements of where the lane lines are and estimate how much the road is curving and where the vehicle is located with respect to the center of the lane. The radius of curvature may be given in meters assuming the curve of the road follows a circle. For the position of the vehicle, you may assume the camera is mounted at the center of the car and the deviation of the midpoint of the lane from the center of the image is the offset you're looking for. As with the polynomial fitting, convert from pixels to meters.**

Well done here.👏 The values in the video look like pretty reasonable estimates of the curvature of the road and the position of the vehicle in the lane.

**The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.**

The warped back image looks good and shows that the lane detection is working.



## Pipeline (video)

**The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.**

The result video looks great! The pipeline is able to accurately map out the true locations of the lane lines throughout the entire video, and doesn't fail in the presence of shadows and pavement color changes. This is a very impressive result!👏

Still I hope you will continue to work on this project and improve your methods for thresholding and lane detection, even though you have met all of the rubric specifications. Here is a tip that can help you get faster feedback when refining your pipeline (this can be useful in the next project as well).

If you want to test the video processing pipeline on a short segment of the video without waiting for it to process the entire video, you can use the `.subclip()` method from VideoFileClip. For example, if you want to test the part of the video where the car goes over the second bridge, you could use:

```
VideoFileClip("project_video.mp4").subclip(38,43)
```

And it will process the 5 second clip between 38 and 43 seconds.

## Discussion

**Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.**

Great job reflecting on the project and pointing out some areas for further improvement.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review