# Advanced Lane Finding

## 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.

## Project Structure

My project includes the following files:

- advanced_lane_lines.ipynb file containing the code used in performed in the project.
- T1P4.pdf file containing the writeup explanation for each step.
- output_images folder for the output images from the advanced_lane_lines.ipynb file for the images in camera_cal and test_images folders.
- test_images folder used in the advanced_lane_lines.ipynb file.

- project_video_result.mp4 showing the final result for lane finding.

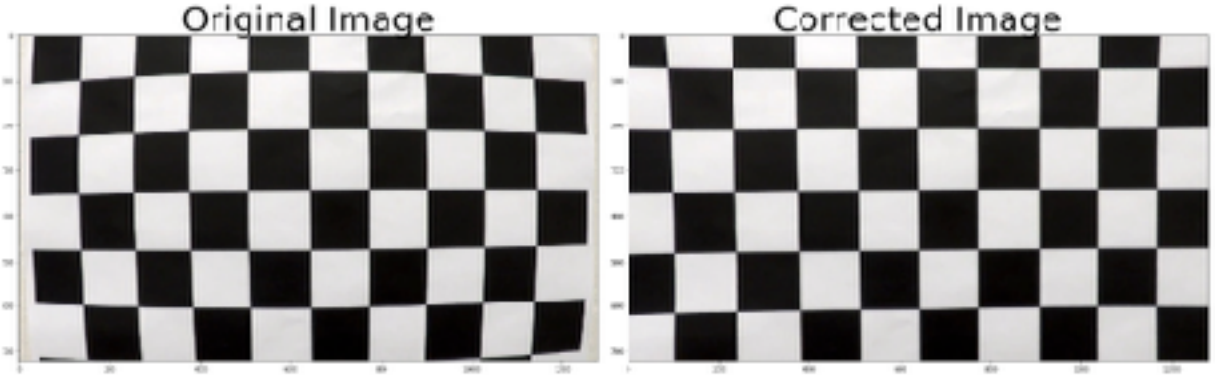
## Camera Calibration

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

The code for this can be found in the third cell. The first step is the camera calibration. This is necessary to compensate for distortion of cameras in the real world.

For this we use the 20 images provided in the project files, these images are chessboard images from varying angles. We use the OpenCV function cv2.findChessboardCorners to locate the corners and then we use cv2.calibrateCamera() to find the arrays that describe the distortion and then use cv2.undistort(). The implementation can be found in the fifth cell of the notebook.

Chessboard Example

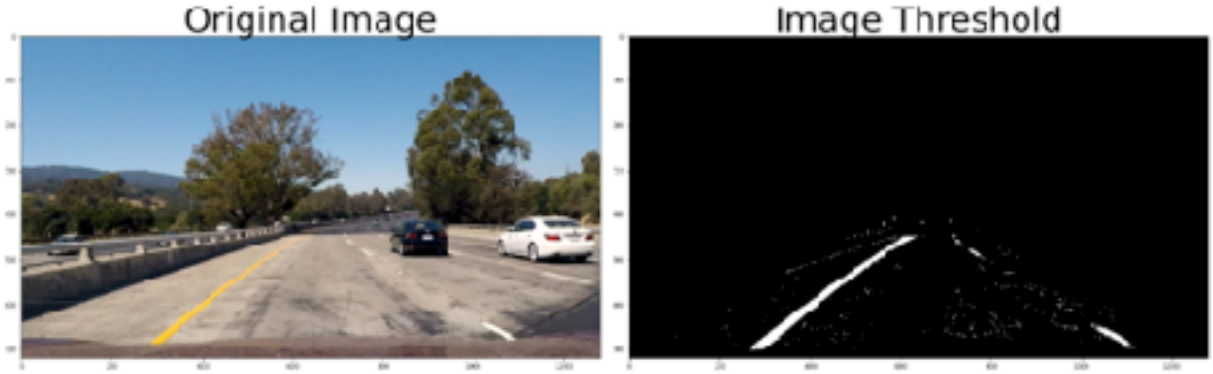


# Pipeline

## 1. Distortion-corrected image.


**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 thresholded binary image the code for this can be found in the seventh cell of the notebook. Here's an example of my output for this step.



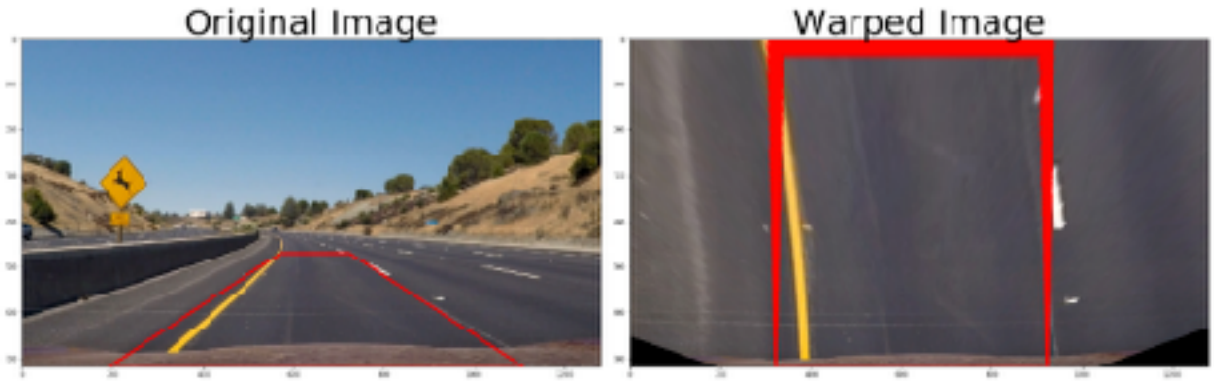
**3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.**

The code for my perspective transform is located in cell nine of the notebook. I chose to hardcode the source and destination points in the following manner:

This resulted in the following source and destination points:

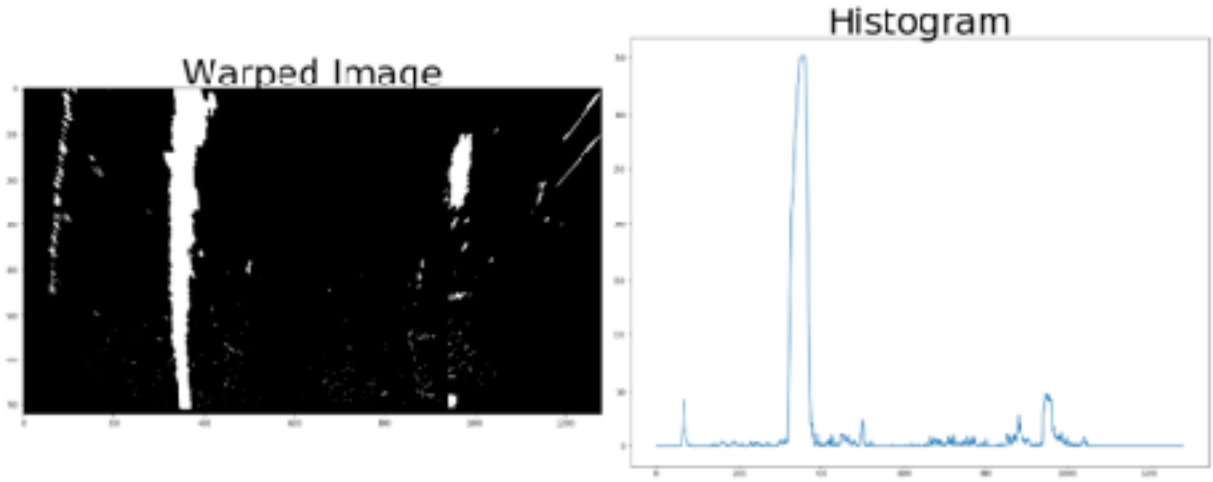
| Source | Destination |
|---|---|
| 570, 470 | 320, 1 |
| 725, 470 | 920, 1 |
| 190, 720 | 320, 720 |
| 1110, 720 | 920, 720 |

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

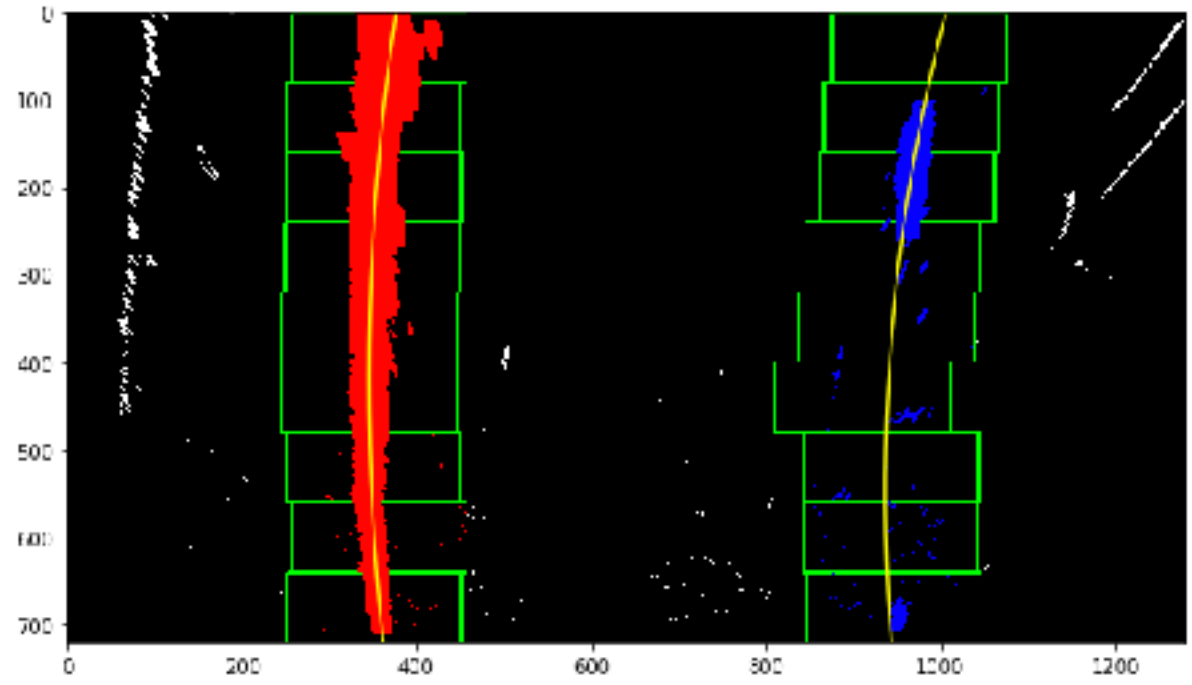


**4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?**

I used a histogram to identify the base as seen in the image below and cell ten of the note book.




Using the histogram above I was able to calculate the sliding window with the left and right base. I performed a polynomial fit to get the left and right lanes. The code can be found for this in cells eleven and twelve.



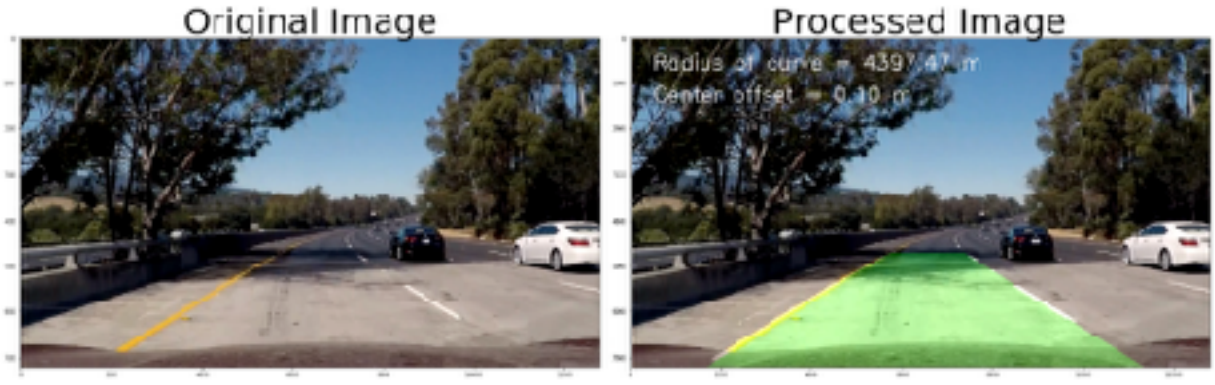
**5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.**

This is done in cell fourteen of the notebook using the code based in the Udacity lesson measuring curvature. We define our conversions in x and y space from pixels too meters and then fit new polynomials to x and y in world space.

The position of the vehicle with respect to center is calculated in my offset_from_center() function which we assume the camera is mounted in the center of the vehicle and we then calculate the center of the image and the middle of the lines.

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

This was done in cell fifteen and visualized in cell sixteen of the notebook.



## Pipeline (video)

**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 is located in the project file and named project_video_result.mp4

## Discussion

**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?**

My code works well with the project_video.mp4 but has poor results with the challenge_video.mp4 and harder_challenge_video.mp4. This is likely due to change in lighting and larger curves.

To improve the project I would like to:

- Work more with thresholding
- Calculate the radius better.
- Explore différent edge detection methods.
- Work for solutions where there are no lane lines or weather conditions such as heavy rain or snow covered roads that severely limit visibility.