Advanced Lane Finding Project
Writeup Report

by Liang Hu

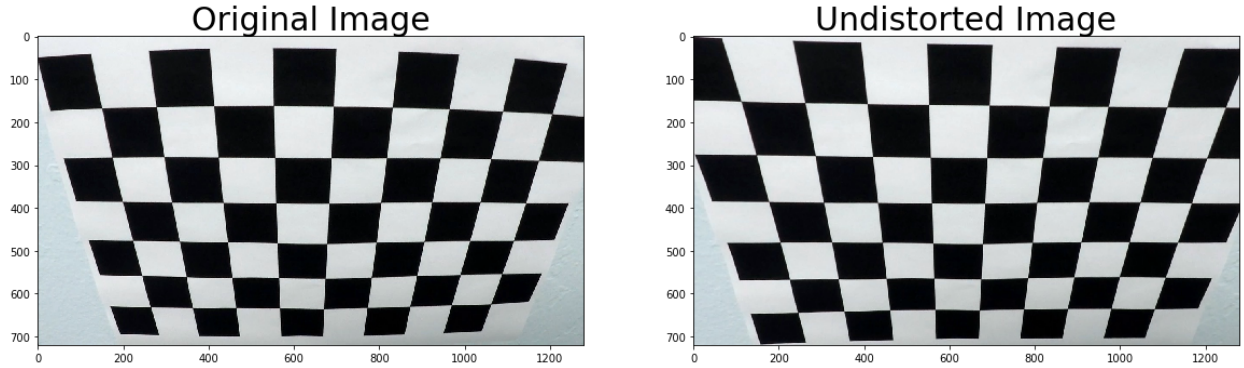
0. Project goals and steps

- 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.

1. Camera Calibration

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 “calibration2.jpg” using the cv2.undistort() function and obtained this result:



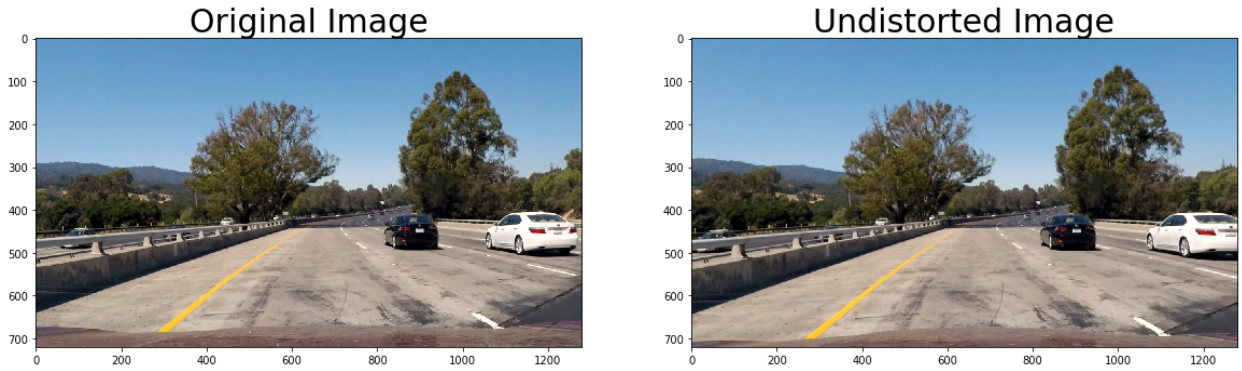
Lastly, I saved the camera calibration results for later use.

2. Pipeline

This section defines the pipeline for processing video.

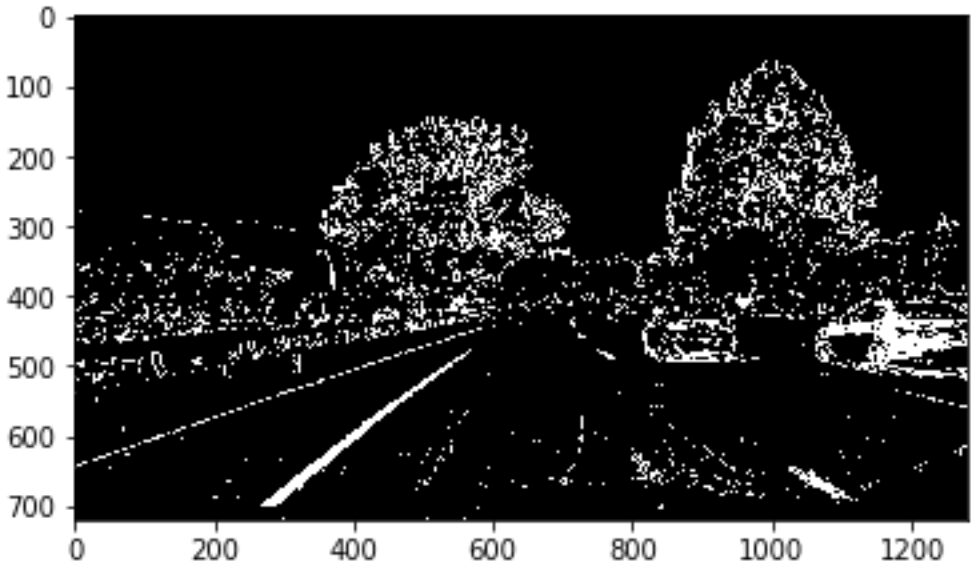
2.1. Check distortion correction

I defined the undistort_image() function to undistort an image using the camera calibration I have saved. Below are the results I obtained. The image can be successfully undistorted.



2.2. Create binary image

The second step is to create the binary image for an undistorted image. Function create_binary() uses the combination of x sobel gradient thresholds and saturation thresholds to get an binary image. The x-sobel gradient thresholds are (20, 100). The saturation thresholds are (170, 255). Apply this function to a test image “test1.jpg”, I obtained the below result. We can see the lane lines can be well highlighted.

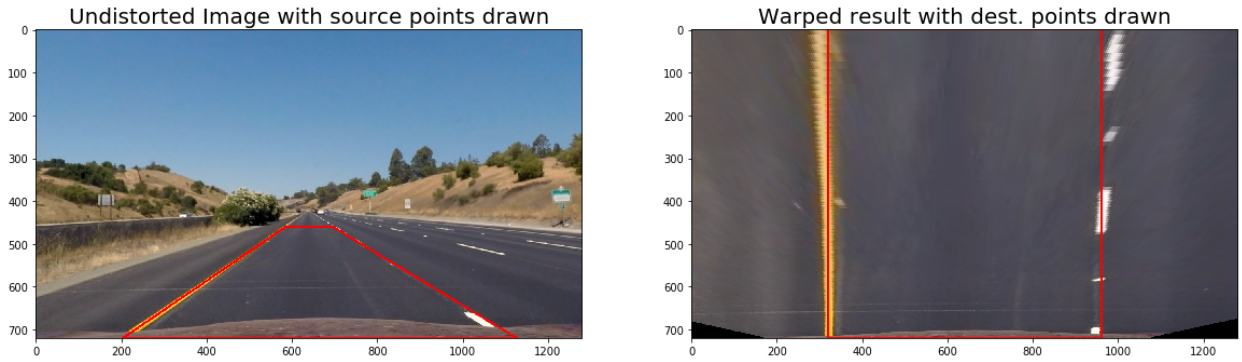


2.3. Perspective transformation

The third step is to transform the perspective of an undistorted binary image. The warper() function does this function. The warper() function takes image, source points (src) and destination points (dst) as inputs. I chose the hardcore the source an destination points, as shown below.

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



2.4. Identify lane line pixels and fit with a polynomial

For the first frame of video, I defined the find_lane_pixels_fit_poly() function to find pixels of left and right lane lines, and fit the detected pixels with a 2-degree polynomial. This function uses slide windows to find lane line pixels. First, I use histogram peaks as a starting point for determining where the lanes are. Then, use sliding windows moving upward in the image to determine where the lane lines go. Once the windows have found lane line pixels, I use np.polyfit() to fit a 2-degree polynomial regression for each lane. At last, this function returns the fitting results left_fit and right_fit.

For the subsequent frames of video, it is unnecessary to repeat using sliding windows because it is inefficient. I used the previous fitting results as starting searching regions to detect lane line pixels and again fit a polynomial for each lane. Function search_around_poly_refit() does these steps. To store and use previous fitting results, I defined function update_fit(). This function stores the last 7 fitting results and uses the average fitting results for the new video frame.

2.5. Calculate geometry

The geometry() function takes the fitting results as inputs and calculate radius curvature in the real word in meters. Also, the offset from the middle of vehicle to the middle of detected lane region is calculated. The geometry calculation results will be shown in the output video.

2.6. Draw detected lane region and geometry in the original image

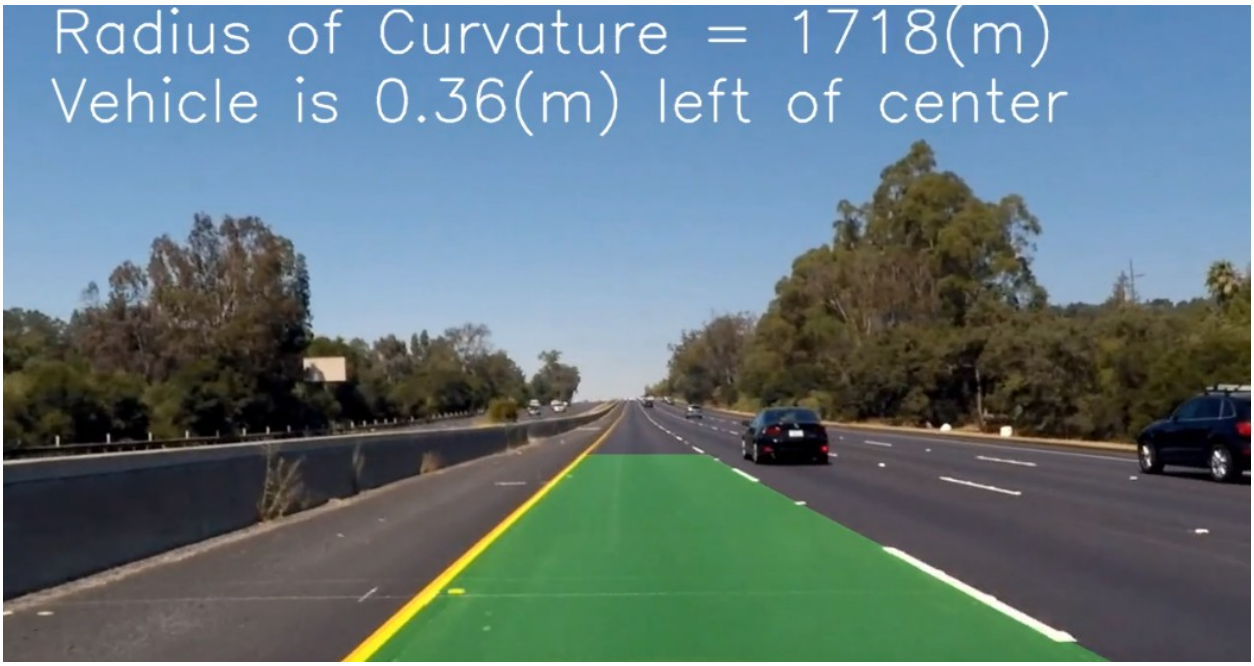
The color_lane_region() function does an inverse perspective transformation to the detected lane region and colors the regions green back to the original image.

2.7. Define pipeline using the above functions

The process_image() function is a pipeline for processing video. The input is a frame of video. The output is the frame with colored lane region and printed geometry.

3. Output

Apply the process_image() function to the project video. The output video is shown in the jupyter notebook, named “project_video_output.mp4”. Here is a screenshot of the output video.



4. Discussion

This algorithm has limitations for more complex driving scenarios. For example, if the vehicle is following a yellow or white vehicle (the same color as lane lines), then there would be many noises for lane line pixel detection. Thus, the lane line fitting results will be not very accurate.