PROJECT REPORT

UDACITY

SELF-DRIVING CAR NANODEGREE

Project 2: Advanced Lane Detection

AUTHOR:
Thomas Teh

Date: February 9, 2019

1 Introduction

This project is part of the Self-Driving Car Nanodegree by Udacity and its objective is to build an advanced lane detection algorithm. The pipeline for the algorithm is as follow:

1. **Calibration and distortion correction:** Compute the camera calibration matrix and distortion coefficients using chessboard images, which we then use to correct the distortion of the raw images from the camera.
2. **Perspective transformation:** Select the region of interest in the raw images and apply a perspective transform (”birds-eye view”)
3. **Preprocessing and construct binary images:** Preprocessing the images and construct binary images using the gradients and color transforms.
4. **Lane detection:** Apply histogram and sliding window on the first frame of video. Subsequent frames are fitted and smoothed based on the first frame.
5. **Curvature estimation:** Determine the curvature of the lane and the vehicle center with respect to the lane center
6. **Overlay the detected lane:** Visual display of the lane boundaries and numerical curvature and vehicle position.

2 Pipeline

2.1 Calibration and Distortion Correction

We calibrate the camera using chessboard images provided. Using OpenCV function findChessBoardCorners(), we automatically find corners in an image of a chessboard pattern.

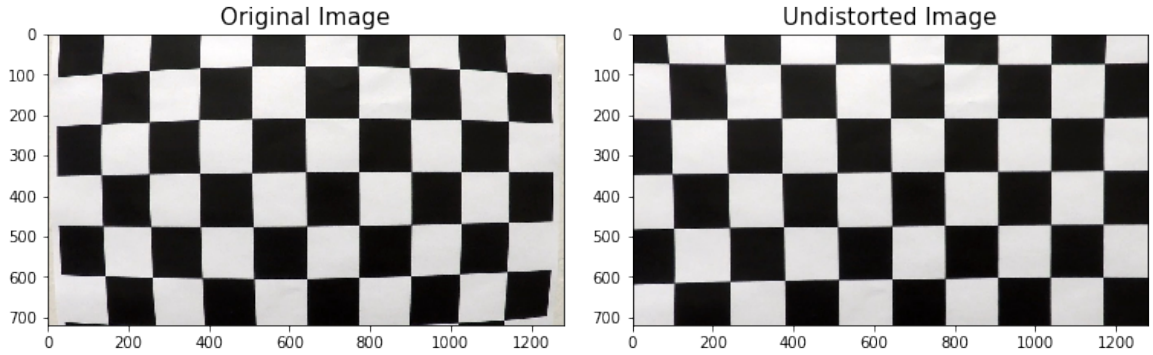

Figure 1: Camera distortion correction using chessboard images.

Then we calibrate the camera given the object points, image points and image shape using the function calibrateCamera(). Finally we use the undistort() function. The

sample source image and the undistorted image of the chessboard and raw image are in Figure 1 and 2 respectively. Detailed implementation is in Calibration.py.

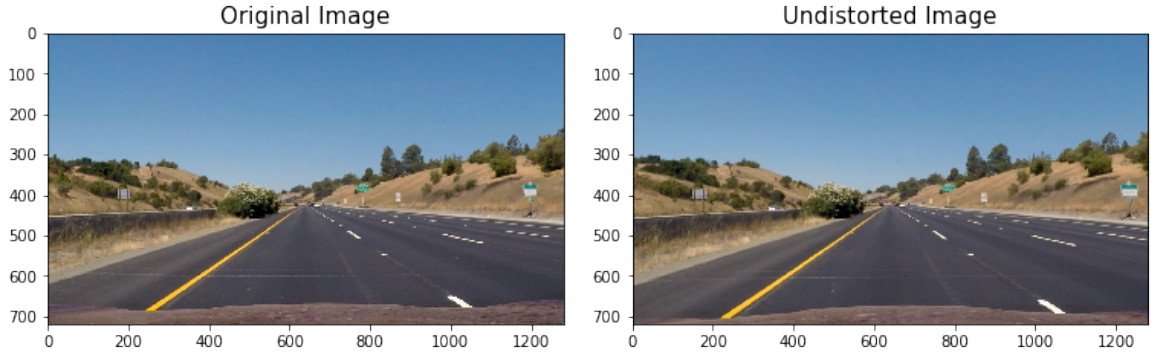

Figure 2: Raw images before and after distortion correction.

2.2 Perspective Transform

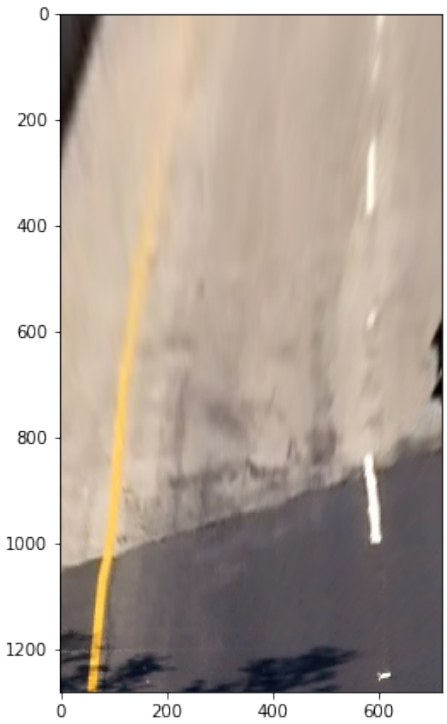

Figure 3: Image after applying perspective transform to a region of interest.

We applied OpenCV functions getPerspectiveTransform() and warpPerspective() to map the points in a given image to different image points with the bird's eye view perspective. The source points are manually determined and this would lead to a major weakness in our lane detection algorithm, which we will discuss later.

The image after the transform is shown in Figure 3. The detailed implementation is found in Warp.py

2.3 Preprocessing and Constructing Binary Images

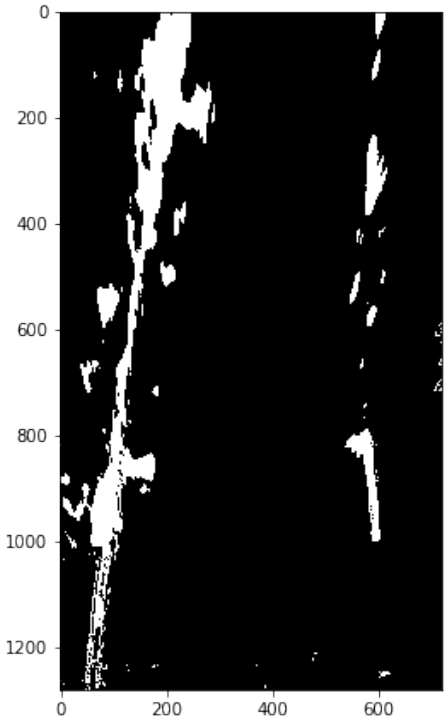

Figure 4: Binary image after applying the relevant thresholds to selected metrics.

We construct the binary images using a combination of two different methods - gradients and color transforms. Before we compute the gradient, the raw images is preprocessed using the following:

- Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied on each of the RGB channels of the raw image.
- Gaussian Smoothing is used to smoothed out the images.
- Lastly, we convert the the image from RGB to HLS and extract the S channel.

The gradients of the image's S channel, across the x and y axes, are then computed using OpenCV function Sobel(). We only normalize the gradients and take only the absolute values. While in the implementation, we computed the gradient magnitude and direction, however, it is found that constructing the binary images using those metrics are not great.

For the color transforms, we convert the raw BGR image into LUV and LAB channels respectively. We then extract the L and B channels respectively. Now, we are ready

to construct the binary images.

We apply the following thresholds:

Metrics	Lower Bound	Upper Bound
Gradient along x-axis	40	100
Gradient along y-axis	40	100
L channel	200	255
B channel	150	200

We retain pixels that meet falls within the thresholds for any of the above metrics in order to construct the binary image. A sample binary image in shown in Figure 4. Detailed implementation of the preprocessing and construction of the binary images can be found in Thresholding.py.

2.4 Lane Detection

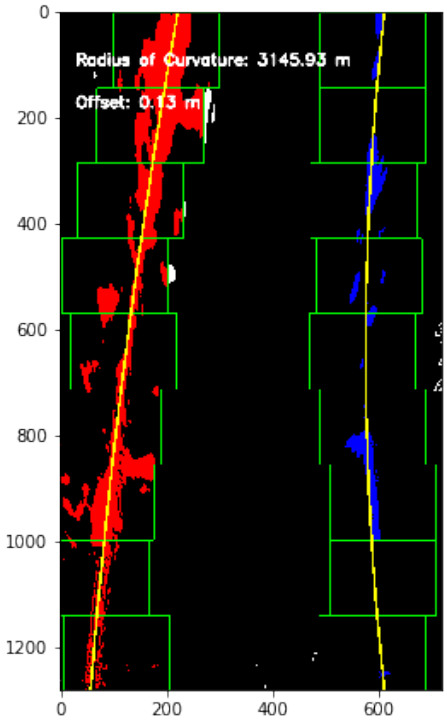

Figure 5: Detected Lanes.

The lane detection algorithm is divided into primary and secondary detection. Primary detection is used for single image or the first frame of the video. Secondary detection relies on the fact that we have already detected the lanes in the previous frames and reuse the polynomial functions again.

Primary Detection: We construct a histogram by adding the pixels at each column of the x-axis. If we have a line, it is more likely for us to get 1s in that region, hence we should have a peak. Since there are left and right lanes, we would expect the histogram to be bimodal (this assumption will be challenged later). Starting from the bottom of the peak in the histogram, we use a sliding window placed around the line centers to find and follow the lines up to the top of the frame. Once we have identify all the relevant pixels, we can then fit a quadratic line across the pixels.

Secondary Detection: Primary detection is computationally expensive. When the car is on the road, we need the lane detection to be as efficient and quick as possible. Hence, once we know where the lanes are in the prior frames, we can make use of that information.

We can start searching a customized regions formed by the two polynomial functions fitted in the previous frame. If we found there are more than 50 pixels, we would then refit the line; else we will use the previously fitted line.
A sample of the lane detected is shown in Figure 5.

2.5 Curvature Estimation

The formula for radius of curvature at any point x for the curve $y = f(x)$ is given by:

$$R = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}}{\left|\frac{d^2y}{dx^2}\right|}.$$

Since in our case, we are fitting in quadratic curve, hence the derivatives of the function can be found easily. Thus we can compute the radius of curvature easily. Once we obtain the curvature of the left and right boundary of the lane, we can then compute the lane center.

Assuming that the camera is mounted at the center of the car, hence the center of the car is just half the image size. We can then compute the difference between the car center and the lane center to know how much the car is drifting from the center.

2.6 Visualization

The output of the lane detection algorithms for the highway video and the challenge video can be found in

- `https://youtu.be/JSLnoQthI5M`
- `https://youtu.be/8MMegcRlbuU`
- `https://youtu.be/iPrrPCVpwKU`

3 Major Pitfalls and Potential Improvements

From the videos, we can see that the algorithm works well for roads with relatively few curves. However, once the car is on a winding road with a lot of turns, the algorithm fails. There are a few issues with the existing algorithm.

1. Secondary detection fails if the algorithm fails to detect the lanes at one of the frames. This can be solve by making the algorithm to do the primary detection whenever the curvature of the curves is below a certain level (in our case, 1000). The rationale is that when the radius of curvature is small, it implies a sharp turn or bend. It is best at the moment to run the primary detection when there are plenty of bends.

2. Perhaps the biggest issue is that the region of interest in the camera is dynamic. When the car is on a relatively straight road, the region of interest (i.e. where the lanes are), are relatively stable. However, as seen in the challenge video, on a window, the region of interest is less well-behaved. In extreme cases, the lanes actually start at the side of the image instead of from the bottom. If we do not overcome this particular issue, it will be difficult to ensure that the car stays in lane all the timee.

3.1 Can we shoot two birds with one stone?

One potential solution to a dynamic region of interest is image segmentation. While the car is on the road, the ability to know the surroundings objects can be vital. Hence, we also need an algorithm to implement image segmentation and recognition.

We can potentially implement image segmentation algorithm with a convolutional neural network. We can then implement the lane finding algorithm once we can detect the road dynamically in an image.