Project 2: Lane Detection

ENPM 673

HAELEE KIM, ZHENGLIANG LIU

A. JAMES CLARK
SCHOOL OF ENGINEERING

Table of Contents

Introduction/Objective

This project's objective is to create a lane detection algorithm implemented for two sample videos of a car driving on the road.

Undistort Image

The calibration matrix of the camera needs to be applied to correct for any inaccuracies such as lens distortion. In order to do this, the opencv function cv2.undistort() is used. The function takes the current frame, the calibration matrix (which is given in the project), and the distortion matrix (also provided) to correct the video.

$$K = \begin{bmatrix} 1.15422732e+03 & 0 & 6.71627794e+02 \\ 0 & 1.14818221e+03 & 3.86046312e+02 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Dist.Matrix = [\text{-}2.42565104e\text{-}01 \;\; \text{-}4.77893070e\text{-}02 \;\; \text{-}1.31388084e\text{-}03 \;\; \text{-}8.79107779e\text{-}05 \;\; 2.20573263e\text{-}02]$$

Homography

Homography is a perspective transformation of a plane, that is, a projection of a plane from one camera into a different camera view, subject to change in the translation (position) and rotation (orientation) of the camera [1]. For lane detection, the homography is computed by manually extracting pixel coordinate points from a video frame of a straight road as a good approximation. By computing the homography, the video frames can be unwarped for further processing and accurate pixel detection and line fitting which will be discussed in the later sections.

Here are the steps for homography calculation:

1. Construct matrix A from the set of points found by feature detection.

$$\begin{bmatrix} x_1^{(w)} & y_1^{(w)} & 1 & 0 & 0 & 0 & -x_1^{(c)}x_1^{(w)} & -x_1^{(c)}y_1^{(w)} & -x_1^{(c)} \\ 0 & 0 & 0 & x_1^{(w)} & y_1^{(w)} & 1 & -y_1^{(c)}x_1^{(w)} & -y_1^{(c)}y_1^{(w)} & -y_1^{(c)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_4^{(w)} & y_4^{(w)} & 1 & 0 & 0 & 0 & -x_4^{(c)}x_4^{(w)} & -x_4^{(c)}y_4^{(w)} & -x_4^{(c)} \\ 0 & 0 & 0 & x_4^{(w)} & y_4^{(w)} & 1 & -y_4^{(c)}x_4^{(w)} & -y_4^{(c)}y_4^{(w)} & -y_4^{(c)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Where x(w) and y(w) are the world coordinates and x(c) and y(c) are the camera coordinates.

2. Decompose matrix A via SVD. The elements of homography, h, are obtained from the last column of V.

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T = \boldsymbol{U}\begin{bmatrix} \lambda_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_{99} \end{bmatrix}\begin{bmatrix} v_{11} & \dots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \dots & v_{99} \end{bmatrix}^T$$

3. Obtain vector h and normalize the vector by the last element.

$$\boldsymbol{h} = \frac{[v_{19}, \dots, v_{99}]^T}{v_{99}}.$$

4. Rearrange the elements of h into a 3 × 3 matrix H.

In this project, cv2.findhomography() and cv2.warpPerspective() functions were used to unwarp the video frames.

As shown in Fig. 1, the blue circles indicate the four points chosen to compute the homography.

```
cv2.circle(im,(320, 675),5,(200,0,0),2) #left bottom
cv2.circle(im,(1090, 675),5,(200,0,0),2) #right bottom
cv2.circle(im,(550, 490),5,(200,0,0),2) # left top
cv2.circle(im,(755, 490),5,(200,0,0),2) # right top
```

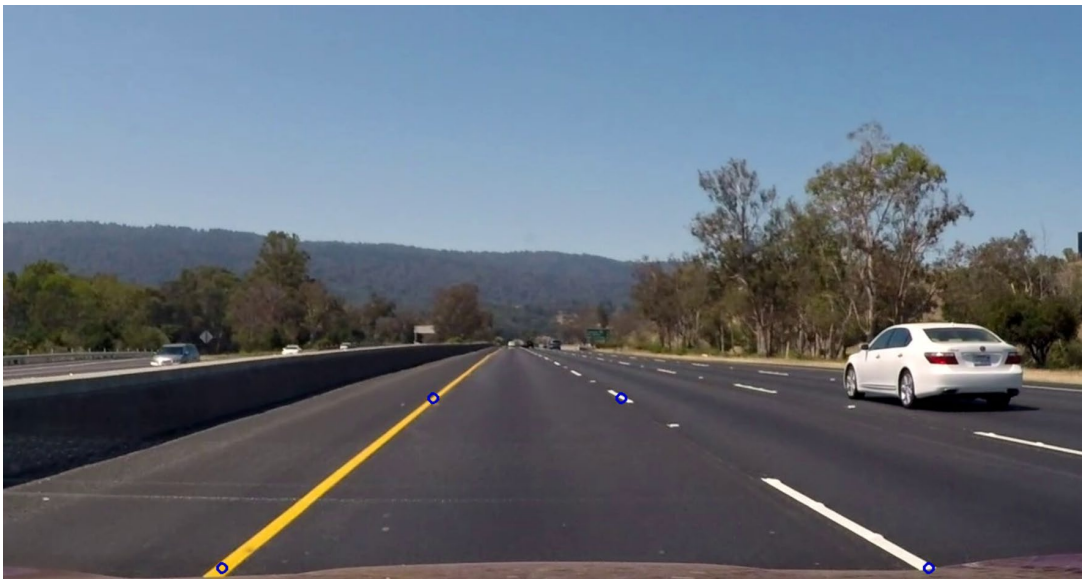

Fig.1: Points for homography

Here is the result for the unwarped image.

Fig.2: Unwarped frame

Filtering and Color Thresholding

Before color thresholding, a Gaussian filter with a kernel size of (5,5) is applied to smooth the image frame.

Further processing of the video frames needs to be done to have effective lane detection. The next step was to isolate only the lane pixels (solid yellow and dashed white lines) within the frames then converted to a grayscale image. After trial and error of trying out different color spaces such as RGB, HLS, LAB, etc., the HSV (hue, saturation, and value) color space was most effective in isolating the lane pixels. A trackbar method was implemented ranging the hue value from 0-179, saturation from 0-255, and value (also known as brightness) from 0-255 to get an idea for appropriate HSV thresholds, then fine tuned for the optimal low and high thresholds. A set of thresholds were found separately for the white and yellow then combined onto the frame. Fig. 3 shows the threshold values used.

```
hsv = cv2.cvtColor(color_frame_dst, cv2.COLOR_RGB2HSV)

white_lower = np.array([0,200,10])
white_upper = np.array([255,255,255])
white_mask = cv2.inRange(hsv,white_lower,white_upper)

yellow_hsv_low  = np.array([20, 75, 150])
yellow_hsv_high = np.array([130, 255, 255])
yellow_mask = cv2.inRange(hsv, yellow_hsv_low, yellow_hsv_high)

mask = cv2.bitwise_or(yellow_mask,white_mask)
masked_top = cv2.bitwise_and(color_frame_dst, color_frame_dst, mask = mask)
```

Fig.3: Mask values

Fig.4: Masked output

Hough Transform

The Hough transform is a method commonly used to detect lines by a voting scheme. The general process is as follows:

- Every point in the image space corresponds to an infinite number of lines that can pass through it each with a unique perpendicular distance and angle from the horizontal axis of the image.
- For every line in an image, a parameterized point is set in the Hough space
- The set of lines that plot points in the Hough space creates a sinusoidal function.
- The algorithm will give a “vote” for a set of parameters describing a line or curve. The more votes, the more evidence a corresponding curve is present in the image.

This detection method is robust especially for noisy images; however, this method is not very good for curved roads. This is why the histogram method introduced in the next section is used instead.

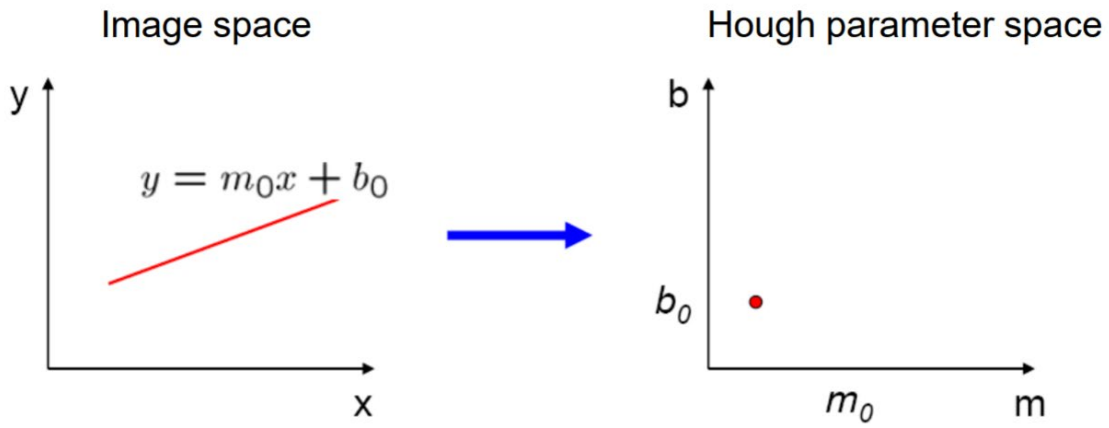

Fig. 5: Hough Transform[2]

Histogram

After processing the color thresholding, the frames can be converted to gray scale by using cv2.cvtColor(). Then np.sum() is used to sum up the white pixels within the lower half frame to create a histogram. The histogram is useful to provide information on the number of occurrences of an intensity value. If the occurrence value is large, this means there exists an abundance of detected pixels around that x-position. Based on the histogram, the x-axis coordinates of the peaks represent the x coordinates of the lines in the video. Those coordinates will be used in the future section for line fitting. The histogram for one of the video frames is shown below:

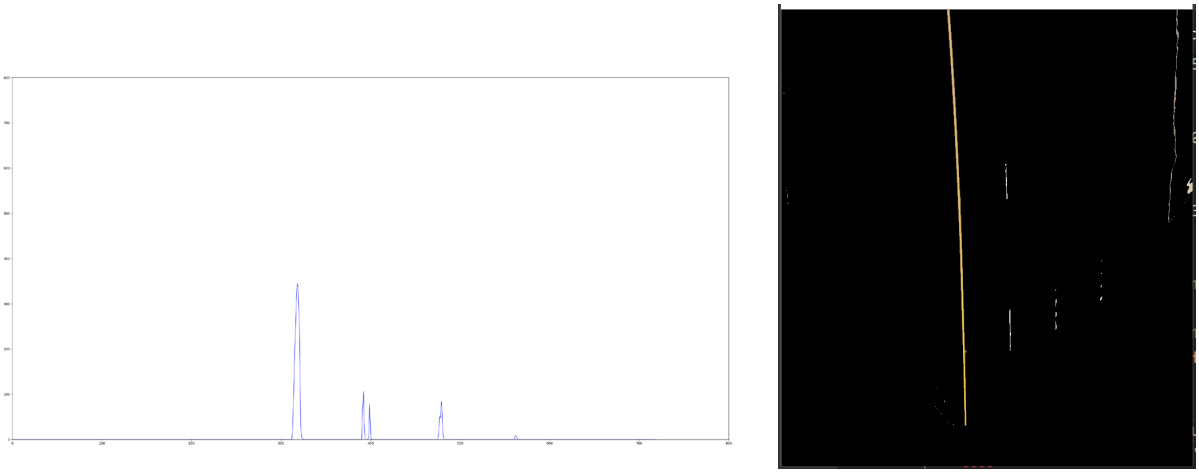

Fig.6: Histogram for lines

From the histogram, the peaks are extracted using a find_peaks function to locate the x-coordinates with the largest number of occurrences. After the peaks are found, an array of desired pixel (x,y) coordinates can be established.

Line Fitting

Using the histogram peaks, the high intensity pixels of the lanes can be detected then a polynomial fit can be applied to derive the coefficients of the lane's quadratic equation.

Here are the steps for line fitting in this project:

1. Find the x and y positions of all nonzero pixels in the image

2. Based on the peak positions, set up desired boundary regions to filter out the noise. The noise corresponds to the additional white pixels that are not in the desired region of interest.

3. Using the x and y coordinates of white pixels within the regions, apply np.polyfit to find the coefficients of the polynomial functions.

4. Using those coefficients, plot the curve which is the result of the line detection (Fig. #7).

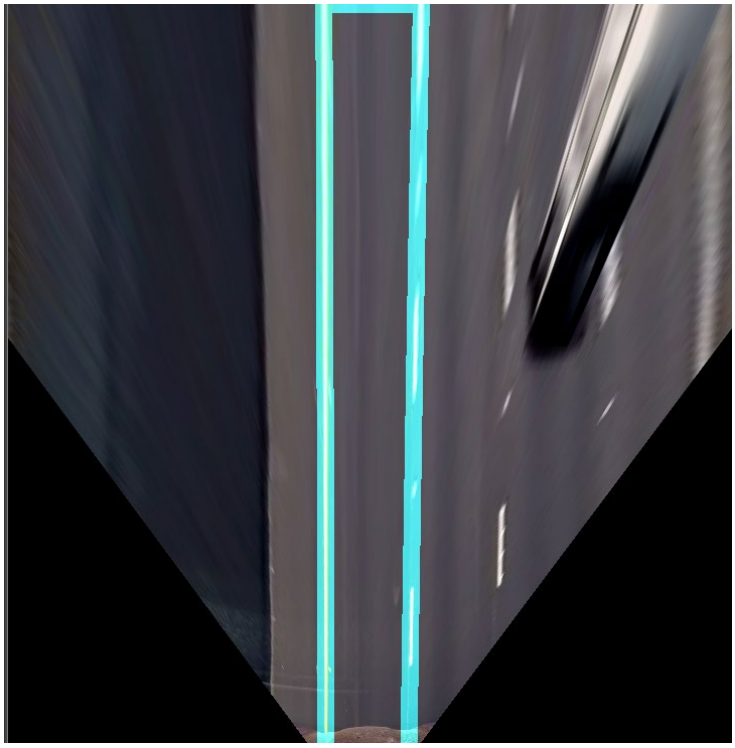

Fig.7: Fitted lines

Warp Frames Back into Original State

After the polynomial line fit has been applied, the cv2.warpPerspective() function can be used again to bring the frame back into the original form. Cv2.fillPoly() is used to color in the detected lane area. The results of both the project video and challenge video are shown in the Results section.

Fig.8: Final output

Turn Prediction

Radius of curvature is used for turn prediction, the formula is following:

$$\frac{(1+(\frac{dy}{dx})^2)^{\frac{3}{2}}}{|\frac{d^2y}{dx^2}|}$$

When the radius of curvature is large, it means the car is going straight. When the radius of curvature is negative, it means the car is turning left, and lastly, if the radius of curvature is positive means the car is turning right.

Final Results

Fig.8: Project video turn prediction 1

Fig.9: Project video turn prediction 2

Fig.10: Project video turn prediction 3

Fig.11: Challenge video turn prediction 1

Fig.12: Challenge video turn prediction 2

Application to Other Videos

The generated pipeline for this project is robust because it implements color thresholding by converting to the HSV space and uses the histogram method to detect intensity peaks in the data. However, if the colors, brightness, and saturation of another video is greatly different from the project/challenge video, the color and filter thresholding will need to be re-tuned. Other than that, as long as the yellow and white lane pixels can be detected depending on the thresholding tune, this pipeline will work for any similar video.

Challenges/Future Work

The most difficult aspect of the lane detection implementation was with the yellow and white pixel detection as well as isolating a good region to get a set of data to fit a curve onto. Improvements/future work for this project include other color space thresholding and other computer vision techniques for extracting lane pixel data arrays.

References

[1]. https://www.quora.com/What-is-a-homography-and-how-is-it-calculated

[2]. http://www.sci.utah.edu/~gerig/CS6640-F2012/Materials/CS6640-F2012-HoughTransform-I.pdf