Project 2 – Lane Detection

Submitted in partial fulfilment of the requirements for the course of

ENPM673 – Perception for Autonomous Systems

By

Adheesh Chatterjee

Andre Ferreira

Pablo Sanhueza

Course Faculty
Prof Cornelia Fermuller

A. James Clark
SCHOOL OF ENGINEERING

INTRODUCTION

The aim of this project is to apply different methods and mathematical concepts to detect lanes to mimic lane departure warning systems in self driving cars. The concepts learned in class will allow us to write a script that solves the problem of simple lane detection. This is accomplished with different methods explained and provided in the pipeline of the project guidelines. Below you will see explanations of what we did to accomplish such an arduous task.

PIPELINE

HSL

The goal of the project is to keep track of the lines and we first need to detect them. Since the line of the roads are always white or yellow we want to change the color space of the image to better identify these two colors.

For this to happen we changed the image from RGB to HSL. The new color space defines colors more naturally which allows us to choose a base color and choose a range of saturation. Using this approach, we created two different images – one which has a base color of white, and the other one which has the base color of yellow and for the last one we combine both images.

This approach made the detection of the two lines easier.

Original Frame

White HSL

Yellow HSL

Combined HSL

Homography

The next step of the project was to choose 4 points in the original video and create 4 correspondents points in the world frames. This is how we calculate the homography.

Homography is a non-singular, line preserving, projective mapping transformation that maps points from one frame to another frame. Such a perspective projection has 8 degrees of freedom, therefore we need atleast 4 correspondent points to calculate the homography matrix.

In the field of computer vision, homography relates two images of the same space. Calculating this relationship has many practical applications like image rectification, image registration, and computation of motion and localization of a desired object.

$$\boldsymbol{x}_{\boldsymbol{k}}^{(c)} = \lambda . \boldsymbol{H}_{\boldsymbol{c}}^{\boldsymbol{w}} \boldsymbol{x}_{\boldsymbol{k}}^{(w)}$$

Where x_k^c represents the point from the image coordinates, x_k^w represents the points in the image coordinates and H represents the 3x3 homography matrix that we want to find.

$$\boldsymbol{H}_{\boldsymbol{w}}^{\boldsymbol{c}} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{32} & h_{32} & h_{33} \end{bmatrix}$$

Since H_w^c has only 8 degrees of freedom we can fix the value of $h33\ as$ 1. Once we have the 8 points necessary to calculate the homography, we just need to solve the following system to find the values of the matrix.

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

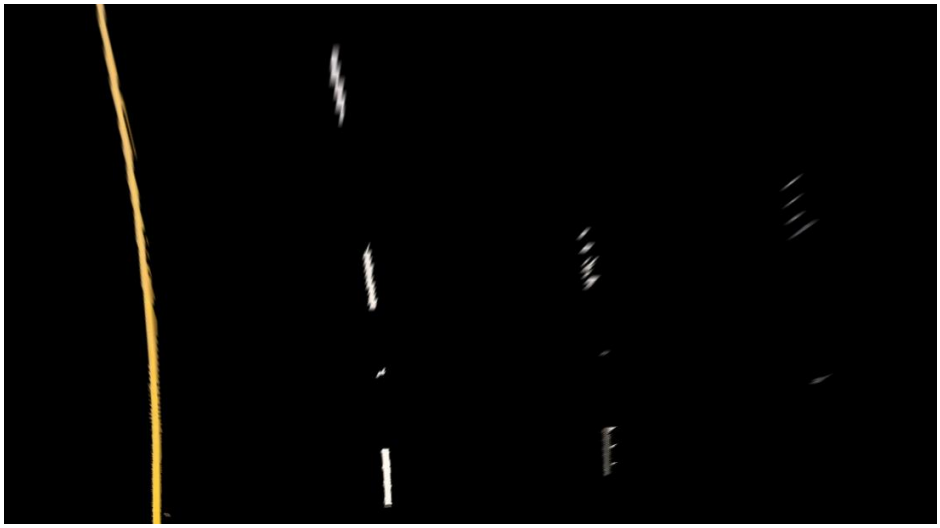
This system can be easily solved via singular value decomposition (SVD). Once the decomposition is made the values of the matrix can be obtained by the last column of the V. Since we require that the last element of V to be one we need to normalize with the value of this element

$$\boldsymbol{h} = \frac{\left[v_{19}, \ldots, v_{99}\right]^T}{v_{99}}$$

For this project, homography is used to unwarp the image of one lane in order to better detect the lanes and calculate the histogram to identify the desired lines.

The perspective matrix is found using the function cv2.getPerspective which will perform the calculations previously explained and will give us the perspective matrix to unwarp the image.

After applying cv2.getPerspectiveTransform we use the function cv2.warpPerspective that will apply the perspective calculated before.



Unwarped Combined HSL

Filtering and Edge Detection

The Gaussian and the Median filter are applied to remove noise and improve detection. Canny edge detection is done so we only grab prominent edges in the video.

The Gaussian Filter is a convolution operator that is used to `blur' images and remove detail and noise. It is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian curve.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

Where σ is the std deviation of the distribution

The median filter considers each pixel in the image in turn and looks at its nearby neighbours to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighbouring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighbourhood into numerical order and then replacing the pixel being considered with the middle pixel value.

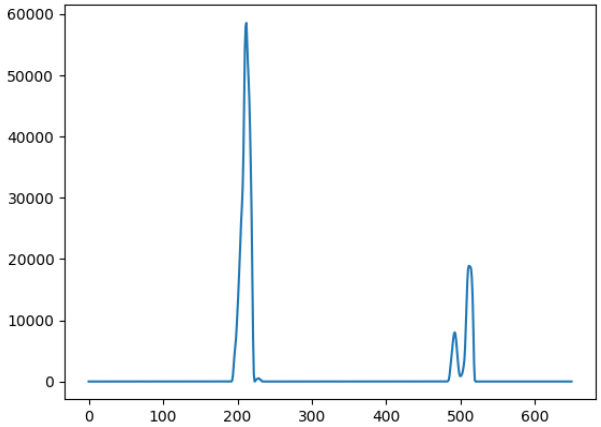
The Canny edge detector works in a multi-stage process. First of all the image is smoothed by Gaussian convolution. Then a simple first derivative operator is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges

and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as non-maximal suppression. The image is then displayed.

Histogram

In order to achieve the goal, one important middle step is to detect the location of the right and left lines of the lane where the car is located. To precisely detect the white and yellow lines we convert the image to grayscale and build the histogram of the image.

A histogram is a display of statistical information that uses rectangles to show the frequency of data items in successive numerical intervals of equal size. In the project case Histogram will count the number of white pixels in a determined location of our unwrapped image.



Since the image is in grayscale the locations with the peak amounts of white pixel can be identified as the lane of the roads. Since we have the location of the white pixels, we are going to separate them in two different arrays, one with the pixels of the right lane and other one with the pixels of the left lane.

Line Fitting

Now that we have all the line pixels separated by left line pixels and right line pixels, we needed to fit a curve that better represents the location of all points. This curve should match exactly the lines we are trying to identify. For fitting a curve between the points, we are using two functions mainly functions, np. polyfit and np.poly1d.

Np.polyfit use least square polynomial fit to calculate the coefficients that minimizes the squared error in the second order. Np.poly1d receive the coefficients generated by the function np.polyfit and creates the equation which represents the desired line fitting.

One problem that we had was that the size of the line was changing from frame to frame since for different values of x we had different values of y.

The solution to this problem was to change the way we create the functions of the curve. Since we wanted the values of Y to be always in a determined range, we created a function that receives Y and returns values of X.

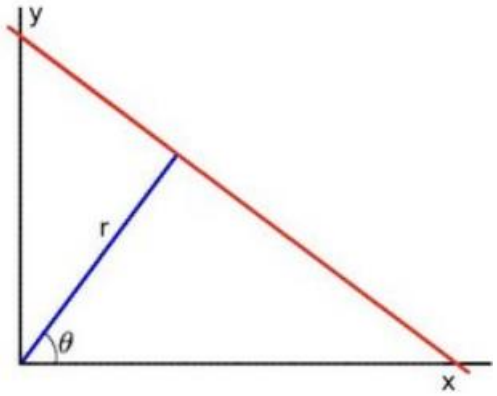
Polygon Fitting

A polygon is fit between the two lines of the lane generated. We find the top right and bottom right points for the right lane and, the top left and bottom left for the left lane. We also find averages of all the points for left and right lanes. We then use use cv2.fillPoly to fit a polygon between these 6 points. We could essentially do it with 4 points but to improve fitting, we use 6 points.

Hough Lines

The Hough transform is a method to detect straight lines. All lines have equations as $y = mx + c$, with a gradient m and y-intercept c. However, this would mean that m goes to infinity for vertical lines. So, we therefore construct a segment perpendicular to the line, leading to the origin. The line is represented by the length of that segment r, and the angle it makes with the x-axis θ. The line equation is now

$$xcos\theta + ysin\theta = r.$$



The Hough transform constructs a histogram array representing the parameter space (i.e. an M×N matrix, for M different values of the radius and N different values of θ). For each parameter combination, r and θ, we then find the number of non-zero pixels in the input image that would fall close to the corresponding line, and increment the array at position (r, θ) appropriately.

We can think of each non-zero pixel voting for potential line candidates. The local maxima in the resulting histogram indicates the parameters of the most probable lines.

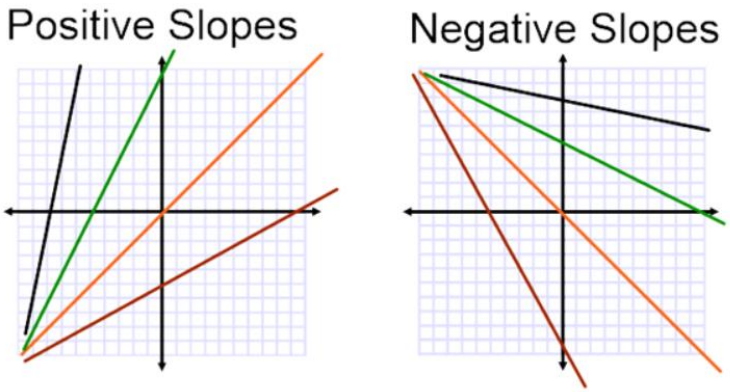
Another approach is the Progressive Probabilistic Hough Transform. It is based on the assumption that using a random subset of voting points give a good approximation to the actual result, and that lines can be extracted during the voting process by moving along connected components. This returns the beginning and end of each line segment.

We tried implementing Hough lines too, and we were able to get the hough lines for the lanes. The only issue we were facing was fitting the lines to a curve. Hence, we decided to change out pipeline to a histogram.

Turn Prediction

There were multiple ways to implement turn prediction, but we chose to do it with the Slope Method.

In this method, we simply take the left lane and find the slope. Since the road lanes will always be parallel to each other, we can only consider one lane. Basically, if one lane turns left, the other lane will also obviously only turn left.



Now we know that the slope for the left lane will be negative, while the slope of the right lane will be positive. We also know, from simple math, that straight lines will have a very high slope value in both positive and negative directions.

Now when we're coding it, since in images, the y starts from the top left and the x from the bottom right. Because of this the left lane has negative slope and the right lane has positive slope.

So, we restrict the value of the left slope and right slope as defined above and display them as 'turning left' and 'turning right'. We also display everything else as 'going straight'.

Generalization of Pipeline to Other Videos

So to summarize the pipeline –

- The video is opened and distorted using the Camera Matrix and Distortion Coefficients.
- The top part of the video is cropped out (as instructed) and the video is converted into the HSL scale.
- The homography is calculated and the road is warped out into the world frame.
- Blur and Edge Detection is applied to improve detection
- The histogram of the video and is found for every frame and the midpoint is calculated. Any peaks to the left and right of the midpoint are found to separate lines and we check intensities to get range for white and yellow.
- All the points in the left and the right lane are fit to a curve and they are then plotted onto the warped image.
- To improve fitting, averages for left and right are found. A polygon is fit between the two lanes with the points defined.
- The entire warp is then transformed back into the video.
- The slopes for the left lane is calculated and Turn Prediction is implemented.

Since we do not hard code anything in this video, this code should work for any video in which the lanes are at the least being detected.

So in retrospect, as long as the HSL mask holds and gives a good output of yellow and white lanes, we should be able to run this code for every video.

One other issue is presence of random disturbances in the video which can ruin the lane detection pipeline, to solve this we need to mask the video better, possibly account for the change in brightness and maybe introduce a sliding window to only get the lines.

INSTRUCTIONS FOR RUNNING CODE

Code 1 detects lanes for both project and challenge video.

Due to change in light intensity, the code is not able to account

for the diff threshold values for white and yellow

Due to this, we had create another code.

In Code 2, we have changed some parameters so it works perfectly for the project video

##########################

(OPTIONAL)

We also tried to solve this using hough lines.

and we were able to get the hough lines for the lanes.

The only issue we were facing was fitting the lines to a curve.

Hence, we decided to change out pipeline to a histogram.

#########################