Real-Time Vehicle License Plate Detection and Tracking Using Multilayer Back-Propagation Neural Networks with and without Hough Transform

by

Polat Utku Kayrak

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 10, 2015

Keywords: robotics, vision,
tracking, detection, security, intelligence, hough, neural

Approved by

Thaddeus Roppel, Chair, Associate Professor of Electrical and Computer Engineering
Prathima Agrawal, Professor of Electrical and Computer Engineering
Shiwen Mao, McWane Associate Professor of Electrical and Computer Engineering

Abstract

There are many applications of vehicle tag detection systems using image properties, which include image density, corner detection and blob analysis but using neural networks in real time for both detection and tracking is always challenging in terms of the computational load and the quality of the input images for training purposes. In this case, Hough transform is found to be useful to extract the square shaped vehicle tag from the scenario in order to improve the output of the neural network regardless of the different background scenarios.

In this study, first real time detection and tracking of a specific tag is examined using Multilayer Back-Propagation Neural Networks, its high computational load and background noise dependency, the effect of the quality of the input samples on the accuracy of detection and tracking are proved. Then in chapter two, a new approach, using Hough Transform in parallel with Neural Networks is simulated both in MATLAB and RoboRealm.

Acknowledgments

I would like to express my deep thankfulness to my advisor and chair of my committee, Dr.Thaddeus Roppel for being supportive and understanding during my graduate studies at Auburn University. I would also like to thank to Dr. Prathima Agrawal and Dr. Shiwen Mao for being my committee members and working with me towards the completion of my thesis. I am thankful for Dr. Hülya Kirkici for being supportive since I came to Auburn University and helping me through my graduate studies.

I am also grateful to my parents Sevgi Kayrak and Atakan Kayrak, my sister Defne Kayrak Talay and my brother-in-law Selcuk Talay for their lasting and unconditional support and love. It would be impossible for me complete my studies and graduate with an exceptional GPA without them.

I'm thankful for the presence of Lexi, who made me laugh even through the hardest times, wake me up every morning with puppy kisses and reminded me of unconditional love every single day.

Finally, I dedicate my thesis to my cousin, Volkan Durkal, the person I grew up with and who was a great example of being an honest, down to earth and hardworking person. You were more than a brother to me and you still are. Your lovely memories will always live with us for forever.

Table of Contents

List of Tables

List of Illustrations

List of Abbreviations

HT	Hough Transform
NN	Neural Networks
SHT	Standard Hough Transform
BPNN	Back Propagation Neural Network

1. INTRODUCTION AND MOTIVATIONS

In New York City, there are over 1000 cameras mounted on the police cars in order to read the license plates of the cars that pass through on the streets every day. As the cameras read the license plates, each and every letter and number are read separately, the unique number of the tag is put into the system for an automatic alert, to see if the number is wanted or the tag is expired. This method stores the every unique license plate number that appears within the sight of the camera and brings moral issues with it. First and foremost, it is not clear that how long these numbers are kept in the system. In addition, there are some discussions about the morality of reading every license plate passing on the streets, in other words holding the records of the each person's location. [23].The law regulations are insufficient to cover this issue and the law does not advance as fast as technology.

At this point, detecting the tag as a whole without reading the numbers and letters on it solve the problem and moral issues that come with it. If a wanted license plate or an expired tag is taught to a system as a whole pattern, without reading the numbers on it, then there would be no need to read each and every license plate number that pass through the streets. A tag on a vehicle can be extracted as a whole, put into Neural Network to decide if it's one of the desired or wanted tags.

In this study, first the wanted tags are trained with purely Back-Propagation Neural Networks and the trained NN is tested with different scenarios for its reliability. Then, Hough Transform is used to extract the tag from the scene and the extracted tag pixels, which are purely the image of the license tag, are used as the input of a newly trained NN, where its aimed to identify the tag as a whole input. The new algorithm is tested on MATLAB environment and different scenarios with different angles and lighting conditions. The algorithm is also tested with different salt&pepper noise densities to test limits of the algorithm.

2. LITERATURE REVIEW

2.1 GENERAL BACKGROUND ON NEURAL NETWORKS

Neural Networks are widely used for pattern and object recognition since 1960s. The main two learning algorithms for training adaptive elements are “Perception Rule” and “LMS Algorithm”. While the feed-forward neural networks does not exploit any feedbacks from the output and there is no closed circle system between the input and output, back-propagation neural networks trains itself and adjusts the weights of the neurons according to the feedback coming from the output. The first implementation of back-propagation neural networks was indicated in doctoral dissertation of Werbos in 1974 but the scientific community overlooked his work. In 1982 the back-propagation technique has been rediscovered by Parker and a report was published at MIT in 1985, by this way back-propagation Neural Networks became widely known. [2]

2.2 BACK PROPOGATION NEURAL NETWORKS

A multilayer back-propagation neural network works by adjusting the weights of the neurons each time during the training according to the result of the output. The weights of neurons are adjusted in such a way that, as the output of the network gets closer to the desired output, the calculated mean error decreases. As the mean error starts showing little to no improvement, the training stops. In other words, the weights in the back-propagation technique, the weights are adjusted in the direction opposite the instantaneous error gradient. [9]

The architecture of a back-propagation neural network simply consists of K number of rows where each row consists of processing neurons for the coming inputs, which are the outputs of the previous row. Each processing neuron’s output on the m_{th} row goes as an input to every unit on the $m+1_{th}$ row. Considering total K numbers of rows, rows starting from 2 until K-1 are called “hidden layers” since these rows are not connected to the outer world. The final row produces estimate outputs denoted by y^I and the correct output vector y. [31]

Each and every neuron in the network receives ‘error correction’ according to the feedback coming from the row above it. [31] The outline of the BPNN, taken from Nielsen’s study is shown in Figure 1.

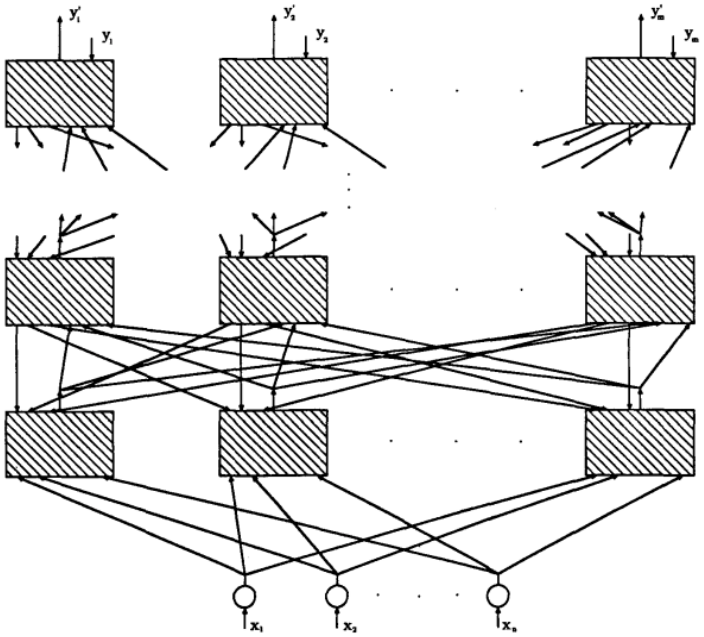

Figure 1. Outline of Back-Propagation Neural Networks

In Back-Propagation Neural Networks, if there are n predictions, then the *Mean Squared Error* can be defined in Equation 1;

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2. \qquad \text{Eq. 1}$$

Where $\ddot{Y}_1$ is defined as a vector of predictions and Y_1 is defined as the vector of real values.

Pattern recognition, self-adaptive control systems, pattern classification are the main fields that are used by BPNN. [8] It’s useful to utilize Neural Networks to read the license plates of vehicles in a still scenario, where it is necessary to print the tag number on a parking ticket. When it comes to detect and track the desired license plate in a dynamic real-time environment, the computational load and the dependency on the

quality of input images in the neural network play an important role in the detection and tracking performance.

2.3 HOUGH TRANSFORM

According to Nakashi, Hough Transform is the most convenient algorithm to for line extraction because its robustness. (1) Given an NxN binary edge image, straight lines are defined in Equation 2.

$$\rho = x \cos \Theta + y \sin \Theta \qquad \text{Eq 2.}$$

Where (x,y) is the measurement of position in X-Y coordinates, Θ ($0 \leq \Theta \leq \pi$) denotes the angle the normal line makes with x-axis and ρ ($-N \leq \rho \leq \sqrt{2}N$) is the normal distance from the origin to the line. In the standard HT, (x,y) denotes the coordinates of the points on the straight line, in other words the pixel space, while ρ and Θ are defined as the parameter space. The highest value in the parameter space represents the most likely straight line in the image domain. (5). The standard HT consists of three parts where the first one is calculating the parameter values, and then finding the local maxima that represents the line segments, and finally extracting the line segments. Every pixel in the image is visited once so the total number of pixels determines the complexity of the HT algorithm. (2) There are many other methods used to extract lines like Split-and-Merge algorithm, which that exploits the local structure instead of considering the global structure. One of the advantages of the Split-and-Merge algorithm is that, its computational load is less than HT but its resistance to noise is not as good as HT. (5) Since the noise resistance of HT delivers a great advantage over other algorithms, the line extraction for the desired license tag is succeeded via HT in this study.

There are many applications and studies that hold NN and HT together for pattern, shape and hand writing recognition. One of them is “Oriented Texture Classification Based on Self-Organizing Neural Network and Hough Transform” by A.N. Marana. [33] The technique proposed in this study is based on the straight-line information extracted from the images by the help of Hough Transform and the image

classification is run by Kohonen's model Neural Network. [33] Although this work used Kohonen's model NN to classify the pattern, it was a milestone for texture classification pattern. The diagram of the texture classification technique is shown below in Figure 2.

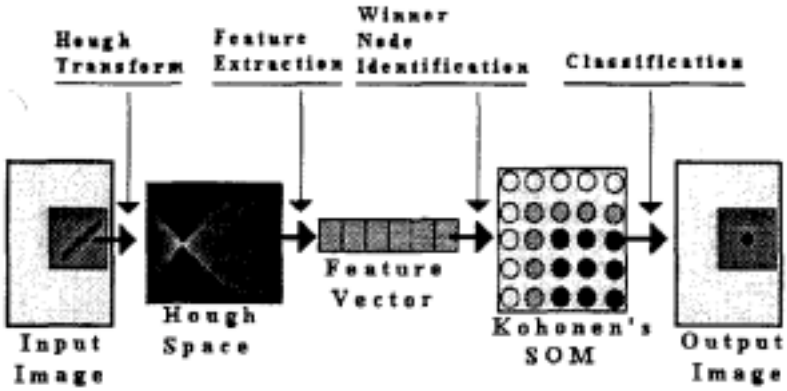

Figure 2. Diagram of texture classification method using HT and NN.

According to this technique, first HT is applied to detect the straight lines and then the feature vector is obtained by putting the columns of the sampled HT, which are arranged one after another. [33] Then the feature vector is put into Kohenen's Neural Network for classification. Another study about using both Neural Networks and Hough Transform is "A complete Shape Recognition System Using the Hough Transform and Neural Network" by C.K. Chan. This method is based on determining peaks and also followed by determining envelope vectors, which are extracted with HT. Then the extracted vectors are used as inputs for NN for recognition. The novelty of this study is that, finding the complementary features from Hough Space makes it easier to recognize the shapes. Secondly, feature vectors, which are extracted from Hough Space, make the data compact enough to be the input of a NN. [32]

2.4 CURRENT STATE OF ART IN LICENSE PLATE DETECTION

Current literature mostly offers the detection of license plates with feature extraction like Maximally Stable Extremal Region or symbol analysis. MSER technique has great advantages and it is an affine-invariant to scale transformation, rotation transformation and transformation of the viewpoint. Main advantages of this technique

are its robustness, repetition rate and discrimination. This method relies on length-width ratio of license plate and connected component determination. The left and right borders of license plate are determined by vertical projection. [38] This method only proposes the extraction of license plate on a car.

One of the most common way of reading license plates is using artificial neural networks by detecting number plate are and classification of the separated character regions. First and foremost, the region of interest has to be localized on order to apply this technique. After the extraction of region of interest, the areas, which contain the symbols, have to be extracted. After the extraction of the areas containing the symbols, separated symbols are classified by Neural Networks. [43] By this approach it's aimed to read the each and every number and letter on a license tag, in order to have the exact number on it. Localization and region of interest mapping of this technique is shown in Figure 3.

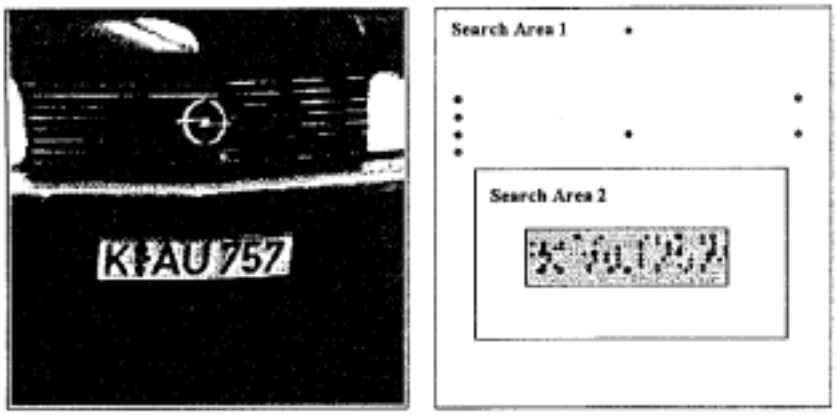

Figure 3. Localization of area of interest

3. METHODS

3.1 NEURAL NETWORK TRAINING CONFIGURATIONS

In this thesis, first a multi-layer back propagation neural network has been trained with 130 different images of the desired tag, which has the "43D98L0" numbers and letter on it. It's aimed to detect the desired license plate as a whole instead of reading all the numbers and letters on it separately. A multi-layer back propagation neural network with a single hidden layer with is chosen after many trials of experiments and it's found that the best result is obtained with 100 Neurons in the hidden layer. The use of more neurons in the hidden layer causes overflow in the network and decreases the accuracy of the detection.

Different pictures have been taken with the internal camera of a laptop, which produces a default resolution of 800x600 and Nikon D5100, which captures the scene with a default resolution of 1080x720. All the input training images has been rescaled to 128x128 pixels in MATLAB by the function *imresize*. Then the input training images are converted to grayscale by the function *rgb2gray*, where every pixel of the images is represented by the numbers 200s, by total of 16384 pixels with 419 training images. Each image is converted into a vector of 16384x1 by the MATLAB function *reshape*. Then all of the 419 vectors of input images are combined in a single matrix, which is 16384x419 and where it represents 419 scenarios each with 16384 properties for the purpose of neural network training. The first 251 scenarios include the desired license tag in it whereas the last 168 scenarios include different scenes of possible tag detection environments without tag in it. As the desired output of the neural network, it is aimed to get a result of 1 or a number closer to 1 when the tag is in the picture, otherwise it's aimed to get a number closer to 0. A target vector of 1x419 is created, where the first 251 rows are assigned as logical 1s and the last 168 rows are assigned as logical 0's. The MATLAB code for the reading the images from the source, converting them to grayscale and resizing to 128x128 is shown below.

%Creating Target for Neural Network Training

```
Target1=ones(1,251);

Target2=zeros(1,168);

Target=[ Target1 Target2]; %Concatenating Matrices Horizontally(Yatay);

Target_Input=logical(Target);


for i=1:419

 j=i+253;

D=imread(['DSC_0' num2str(j) '.JPG']);

D_re = imresize(D, [128 128]);

BW=rgb2gray(D_re);

BW=+BW;


%Turning matrices into vectors

[m,n]=size(BW);

BW=reshape(BW',m*n,1);

BW_Inputs(:,i)=BW;
```

```
end
```

The code shown above creates two matrices that are *Target_Input* and *BW_Inputs*. In order to train the neural network, *Neural Network Tool* of MATLAB is used.

After many trials and simulations, it is found that using 100 neurons in the hidden layer gives the best result and minimizes the error rate. It is useful to use some of the input images for validation and testing. Random images, which contain the 10% of the images, are assigned for validation and another random 10% percent of the images are assigned for testing. The rest 80% of the images are used for training purposes. The simple MATLAB code used to train the neural network is shown below.

```
inputs = BW_Inputs;

targets = Target_Input;

% Create a Pattern Recognition Network

hiddenLayerSize = 100;

net = patternnet(hiddenLayerSize);

% Setup Division of Data for Training, Validation, Testing

net.divideParam.trainRatio = 80/100;

net.divideParam.valRatio = 10/100;

net.divideParam.testRatio = 10/100;

 % Train the Network
```

```
[net,tr] = train(net,inputs,targets);

 % Test the Network

outputs = net(inputs);

errors = gsubtract(targets,outputs);

performance = perform(net,targets,outputs)
% View the Network

view(net)

% Plots

figure, plotperform(tr)

figure, plottrainstate(tr)
figure, plotconfusion(targets,outputs)

figure, ploterrhist(errors)
```

Advanced MATLAB code used to train the neural network is shown below.

```
inputs = BW_Inputs;

targets = Target_Input;

% Create a Pattern Recognition Network
```

```
hiddenLayerSize = 100;

net = patternnet(hiddenLayerSize);


% Choose Input and Output Pre/Post-Processing Functions

% For a list of all processing functions type: help nnprocess

net.inputs{1}.processFcns = {'removeconstantrows','mapminmax'};

net.outputs{2}.processFcns = {'removeconstantrows','mapminmax'};


% Setup Division of Data for Training, Validation, Testing

% For a list of all data division functions type: help nndivide

net.divideFcn = 'dividerand';  % Divide data randomly

net.divideMode = 'sample';  % Divide up every sample

net.divideParam.trainRatio = 80/100;

net.divideParam.valRatio = 10/100;

net.divideParam.testRatio = 10/100;


net.trainFcn = 'trainlm';  % Levenberg-Marquardt
```

```
net.performFcn = 'mse';  % Mean squared error


% Choose Plot Functions

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...'plotregression', 'plotfit'};


% Train the Network

[net,tr] = train(net,inputs,targets);



% Test the Network

outputs = net(inputs);

errors = gsubtract(targets,outputs);

performance = perform(net,targets,outputs)

% Recalculate Training, Validation and Test Performance

trainTargets = targets .* tr.trainMask{1};

valTargets = targets  .* tr.valMask{1};
```

```
testTargets = targets  .* tr.testMask{1};

trainPerformance = perform(net,trainTargets,outputs)

valPerformance = perform(net,valTargets,outputs)

testPerformance = perform(net,testTargets,outputs)


% View the Network

view(net)


% Plots

figure, plotperform(tr)

figure, plottrainstate(tr)

figure, plotconfusion(targets,outputs)

figure, ploterrhist(errors)
```

Once the Neural Network is trained, MATLAB creates a 1x1 Network called *net* where we can put the real time images from the dynamic into the network as inputs. For testing the Neural Network, a new class of MATLAB is created called *"NetworkCameraInput_Gray.m"*

Internal camera of a personal computer with 1.60 GHz Intel Core i7 CPU and 8.00 GB of total memory is used for the best performance. In order to import the

information from internal camera to MATLAB, *videoinput* function is utilized. Frame grab interval is defined by the function *vid.FrameGrabInterval* where one frame is grabbed in every 5 frames. The data that is acquired from the video is resized to 128x128 pixels with the function *imresize*. Then the resized image is converted into gray scale with the function *rgb2gray*. Then the rescaled and gray-scaled camera input image is converted into a vector of 16384x1 in order to have a proper input for neural network. It represents that it is only one input with 16384 different properties. If these 16384 properties match with category one in the trained network, which has the desired tag in it, then the network is desired to give an output close to number 1. Finally, all the network calculations are taken in a loop of 100 iterations, which implies that there is going to be 100 different snapshots and neural network calculations. The MATLAB code that is used to test the network and detect the desired tag is shown below.

```
%a = imaqhwinfo;

%%[camera_name, camera_id, format] = getCameraInfo(a);

% Capture the video frames using the videoinput function

% You have to replace the resolution & your installed adaptor name.

vid = videoinput('winvideo', 1);

% Set the properties of the video object

set(vid, 'FramesPerTrigger', Inf);

set(vid, 'ReturnedColorspace', 'rgb')

vid.FrameGrabInterval = 5;
```

```
start(vid)

for a=1:100

% Get the snapshot of the current frame

data = getsnapshot(vid);

%Resizing the camera input image to 128x128 and turning the image into

%binary - black & white

data_re = imresize(data, [128 128]);

%level_camera = graythresh(data_re);

%BW_Camera = im2bw(data_re,level_camera);

BW_Camera = rgb2gray(data_re);

BW_Camera =+BW_Camera; %turning them into doubles.


%Turning the camera input matrix into a vector 16384x1

[f,g]=size(BW_Camera);

BW_Camera=reshape(BW_Camera',f*g,1);

%BW_Camera_Vector=BW_Camera';        %to turn from 16384x1 to 1x16384
```

```
%BW_Camera_Vector;

BW_Camera_S=double(BW_Camera);

NetResult(:,a)= net_gray(BW_Camera_S) ;

NetResult;

imshow(data)

hold on
DisplayResult=text(40,100,num2str(NetResult));

set(DisplayResult,'Color','b','FontWeight','Bold','FontSize',12)

%{

if NetResult>0.8

BW_Camera_Binary=im2bw(data);

BW_filled = imfill(BW_Camera_Binary,'holes');

boundries = bwboundaries(BW_filled);

for z=1:numel(boundries)

plot(boundries{z}(:,2),boundries{z}(:,1),'g','LineWidth',3); %Putting boundries
end
end
```

```
%}

hold off

%}

end

stop(vid);

flushdata(vid);
```

In order to put a rectangle around the desired license tag in the scene, it is sufficient to remove the comment line above *"if NetResult>0.8"* and below *"plot(boundries)"*. If it is desired to put a bounding box around the license tag, the simulation will put a box around the rectangle shaped tag, when the returning result of the Neural Network is over 0.8.

3.2 APPLYING HOUGH TRANSFORM IN PARALLEL WITH BACK-PROPOGATION NEURAL NETWORK

License tags are mostly in the shape of rectangle and it's crucial to detect the lines in the scene in order to extract the tag.

For line extraction simulation with HT, a new Matlab file called *"HoughTransform.m"* has been created. In the simulation, first an example image is read with the command line *"imread"* then the image is resized to 500x500 pixels with the command line *"imresize"*. Then the resized image is converted into grayscale with *"rgb2gray"*. In order to apply HT on an image and extract the line segments, it is crucial to get the binary mapping of the image. Edge detection algorithms usually achieve it and 'canny' edge detection is chosen in this case. Finally to eliminate the background noise that is smaller than 50 pixels, the function *"bwareaopen"* is used. The original grayscale

image, edge detected image and the image that has reduced background noise properties are shown in Figure 5, Figure 6 and Figure 7 respectively in Chapter 5

In order to apply the HT, first we need to assign the values of Θ and ρ. After many trials, it is found that the optimum value for ρ is 0.5 and the optimum interval for Θ is (-90,89.5) degrees with 0.2 spacing. The next step to apply HT is finding the peak values in Hough transform matrix, which is obtained by using the *"houghpeaks"* function. The number of lines that is desired to be detected can be also implemented in the function. Since it is aimed to extract the four sides of a rectangle, the number of peaks that are desired to be extracted is 6. It is useful to assign 2 more extra Hough peaks in order to prevent error detections. Using the function *"houghlines"*, we can determine the straight lines on the image. The images that show the peak lines on the parameter space and the detected Hough Lines in order to extract the image are shown in Figure 8 and Figure 9 respectively in Chapter 5.

The MATLAB Code that is used to extract the rectangular shaped tag from the image is shown below.

```
D=imread('DSC_0415.JPG');

D_im=imresize(D, [500 500]);

D_gray=rgb2gray(D_im);

mshow(D_gray);

BW = edge(D_gray,'canny');

%BW = edge(D_inc,'roberts')

BW2=bwareaopen(BW,50); %to eleminate the background noise

figure,imshow(BW);
```

```
impixelinfo;

figure,imshow(BW2);

impixelinfo;
%Adjusting Image Intensity both intensity and RGB images

%J = imadjust(D_gray);

%figure,imshow(J)

%BW3=edge(J,'canny');

%figure, imshow(BW3);

K = imadjust(D_gray,[],[],0.011); %Changing the contrast(Brightening image) for better edge detection.

figure, imshow(K)

BW4=edge(K,'canny');

figure, imshow(BW4);

BW5=bwareaopen(BW4,200);

BW5(1,1:499)=zeros(1,499); %to eleminate the error on the top of screen

BW5(2,1:499)=zeros(1,499); %to eleminate thhu error on the top of screen
```

```
figure,imshow(BW5);

impixelinfo;

%RGB2 = imadjust(D,[.2 .3 0; .6 .7 1],[]);

%figure, imshow(D), figure, imshow(RGB2)

%Edge Tapering

%{

PSF = fspecial('gaussian',60,10);

edgesTapered  = edgetaper(D_gray,PSF);

figure, imshow(edgesTapered,[])

BW3=edge(edgesTapered,'canny');

BW4=bwareaopen(BW3,50);

figure, imshow(BW3);

figure, imshow(BW4);

%}

%Laplacian Edge Detection

%{
```

```
horizontalKernel = [-1,-1,-1;2,2,2;-1,-1,-1];

verticalKernel = [-1,2,-1;-1,2,-1;-1,2,-1];

diagUpKernel = [-1,-1,2;-1,2,-1;2,-1,-1];
        diagDownKernel = [2,-1,-1;-1,2,-1;-1,-1,2];

        horizontalBuilding = imfilter(D_gray, horizontalKernel);

        figure,imshow(horizontalBuilding);

        verticalBuilding = imfilter(D_gray, verticalKernel);

        figure,imshow(verticalBuilding);

        diagDownBuilding = imfilter(D_gray, diagDownKernel);

        figure,imshow(diagDownBuilding)

        diagUpBuilding = imfilter(D_gray, diagUpKernel);

        figure,imshow(diagUpBuilding);

        %}
        %Smoothing with Gaussian
        %{

        GaussFilter=fspecial('gaussian', [7,7],5);

        BW3=imfilter(D_im,GaussFilter,'symmetric','conv');
```

```
BW4=rgb2gray(BW3);

BW5=edge(BW4,'canny');

BW6=bwareaopen(BW5,50);
figure,imshow(BW5);

figure,imshow(BW6);
%}

%Corner Detection
%{

BW_Corner=corner(BW5);

imshow(BW5)

hold on

plot(BW_Corner(:,1),BW_Corner(:,2),'r*');

hold off

%}
%Sharperning of grayscale image
%{

sharpfilter=fspecial('unsharp');

sharp=imfilter(D_gray,sharpfilter,'replicate');
```

```
sharp2=imfilter(D_gray,sharpfilter);

figure, imshow(sharp);

figure, imshow(sharp2);

figure, imshow(D_gray);

BW3=edge(sharp,'canny');

figure, imshow(BW3);

BW4=edge(sharp2,'canny');

figure, imshow(BW4);
%}

%Hough Transform Calculations

[H,theta,rho] = hough(BW2,'RhoResolution',0.5,'Theta',-90:0.2:89.5);

%Display the transform using the imshow function.

figure, imshow(imadjust(mat2gray(H)),[],'XData',theta,'YData',rho,...

      'InitialMagnification','fit');

xlabel('\theta (degrees)'), ylabel('\rho');

axis on, axis normal, hold on;
```

```
colormap(hot)

%Find the peaks in the Hough transform matrix, H, using the houghpeaks function.

P = houghpeaks(H,10,'threshold',ceil(0.1*max(H(:))));

%Superimpose a plot on the image of the transform that identifies the peaks.

x = theta(P(:,2));

y = rho(P(:,1));

plot(x,y,'s','color','black');

%Find lines in the image using the houghlines function.

lines = houghlines(BW2,theta,rho,P,'FillGap',70,'MinLength',20);

%Create a plot that superimposes the lines on the original image.

figure, imshow(BW2), hold on

impixelinfo;

max_len = 0;

for k = 1:length(lines)

 xy = [lines(k).point1; lines(k).point2];
```

```
        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');

        %Plot beginnings and ends of lines

        plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
        plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

        % Determine the endpoints of the longest line segment

        len = norm(lines(k).point1 - lines(k).point2);

        if ( len > max_len)
max_len = len;

xy_long = xy;

end

end

% highlight the longest line segment

plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','red');

hold off;
```

In order to extract the detected tag in the picture, simple math is employed. The pixel locations of the 4 lines intersecting with each other are detected and then the area

that is covered between these lines is extracted. The new created image, which is basically the tag itself, is used as the input of the neural network, which is solely trained to detect the desired tag itself. This method is highly effective because according to Nguyen, each of the pixels in the edge image is considered for the determination of evidence of a line. (7) While this property brings a great advantage over other techniques, it also has a major drawback like detecting the false lines. As we can see in Figure 9 in Chapter 5, there is a false detected line in red color, which implies it's the longest line segment that has been detected. In order to test the effectiveness of the system, the algorithm is tested with the newly trained NN.

4. SIMULATION RESULTS OF BPNN

4.1 RESULTS OF NEURAL NETWORK TRAINING

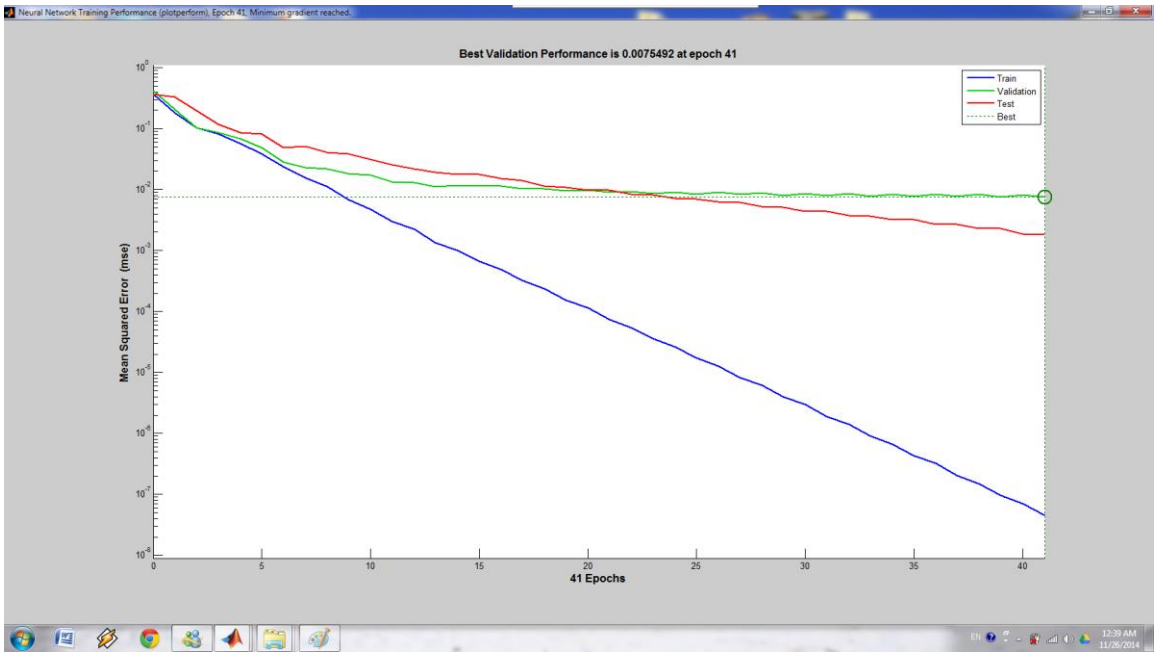

Figure 4. The performance plot of the neural network

As seen in Figure 4, the best validation performance is reached at epoch 41 and after 41 iterations; there is no significant improvement on the validation after that. Mean Square Error can be simply defined as the difference between the expected values and the true values.

According to network training result, the training stops when the Mean Square Error reaches *0.0075492* and no more significant improvement is observed after 41 iterations. The confusion matrices display a better insight about the training performance of the Neural Network. The confusion matrices of *training, validation* and *test* are shown below in Figure 5.

Figure 5. Confusion Matrices of Neural Network

According to Training, Validation and Test Matrices, the trained Neural Network works with 100% accuracy. As mentioned before, there are two types of output class that we want to achieve, the ones where the desired license tag is in the scenario and the other output class where the desired tag is not in the scenario. As we can see in the Training Confusion Matrix, 215 inputs are classified accurately into category one, where the tag is in the scene and 141 inputs are categorized accurately into category two, where the desired tag is not in the scene. There are no miscategorized inputs since the second row of the first column and first row of the second column is 0.0%. Also for the Validation Confusion Matrix, we can read that 26 validation inputs are accurately validated into category 1, whereas 16 validation inputs are accurately validated into category two. Finally in the Test Confusion Matrix, a total number of 21 input samples are used to test the trained neural network. 10 test samples are accurately categorized in the first target class and 11 test samples are accurately categorized in the second target class. These results produce All Confusion Matrix with 100% accuracy.

4.2 BACK-PROPOGATION NEURAL NETWORK SIMULATION RESULTS

Running the simulation in an environment where the training pictures were taken gives us nearly perfect results. The simulation is supposed to print the Neural Network output on the screen for each of the frames. An example of the simulation is shown below in Figure 6.

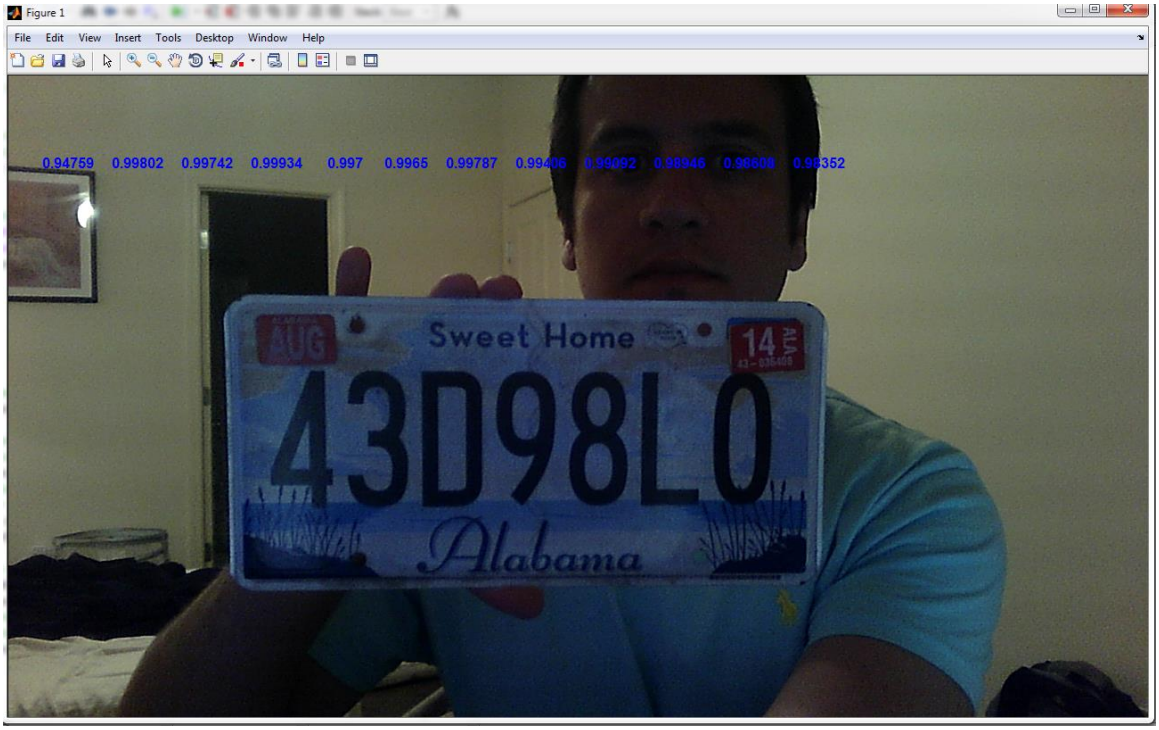

Figure 6. Simulation Result of BPNN

As we can see in Figure 6, NN gives the results around 0.98-.0.99, which indicates that the desired tag is within the limit of sights of the camera. However, when the same simulation is run with a different environment and background condition, the network does not give the same desired result when the tag is within the limit of sights of the camera. The reason for unsuccessful detection is that when the NN is trained, it detects the whole environment with the desired tag in it. In other words, not only the pixel pattern of the license tag but every single pixel in the picture serves for the purpose of detection of the tag. The NN output of a scene with different camera angle and lighting causes failure in the detection of desired tag since it is also heavily dependent on the intensity map of the scene. Figure 7 shown below is an example of failure of the detection of the tag, since the NN output is below 0.8 and has values close to 0 mostly.

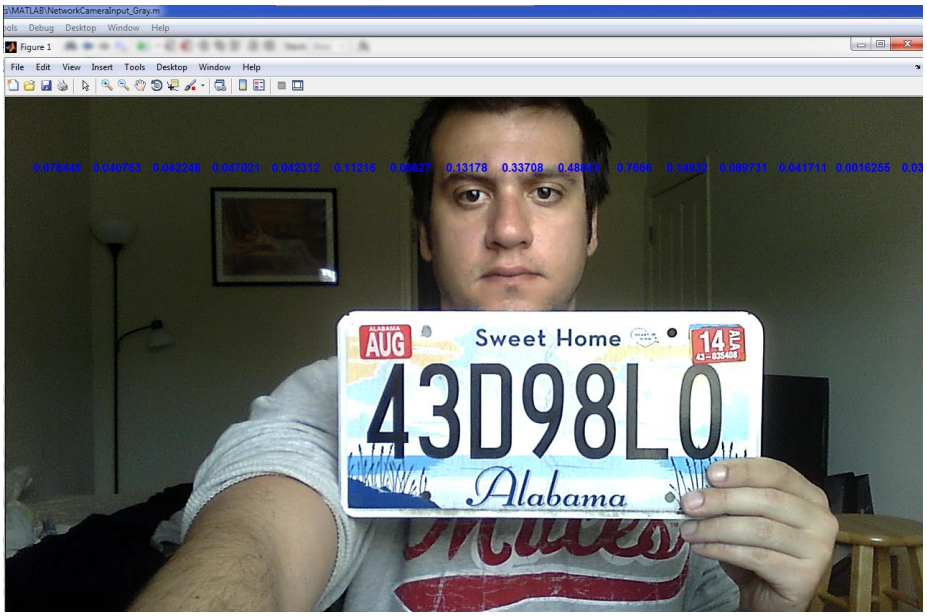

Figure 7. Simulation Result of BPNN with different lighting conditions

In order to have a proper detection of the desired tag regardless of the background, it's crucial to extract the tag from the given scene, and then use it as the input of a properly trained neural network.

5. SIMULATION RESULTS OF HOUGH TRANSFORM

5.1 SIMULATION OF HOUGH TRANSFORM IN PARALLEL WITH BACK-PROPAGATION NEURAL NETWORK

The original grayscale image, edge detected image and the image that has reduced background noise properties are shown in Figure 8, Figure 9 and Figure 10 respectively.

Figure 8. Gray Scale Transformation of Desired Tag

Figure 9. Canny Edge Detection of Gray Scale Image

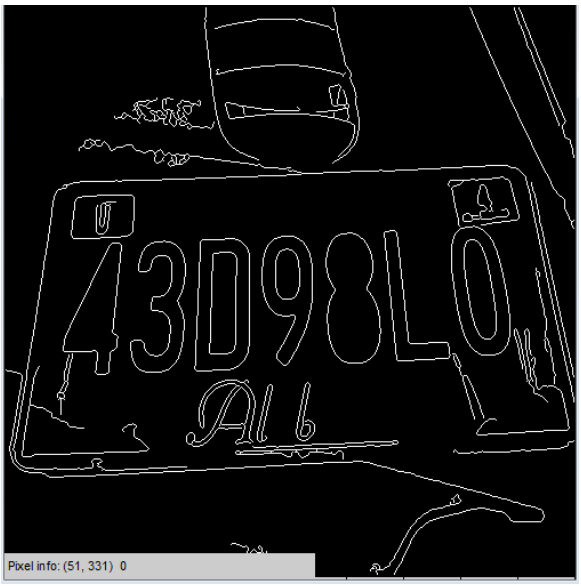

Figure 10. Canny Edge detection with background noise eliminated

In Figure 12 we see that there is a peak around -90 degrees and -400 ρ value. The other three peaks are grouped between -15 and 10 degrees -400 – 0 ρ values. The remaining two peaks are grouped around 89 degrees, between the ρ values of -400 and -100.

Figure 11. Hough Lines Covering 4 Sides of the Tag

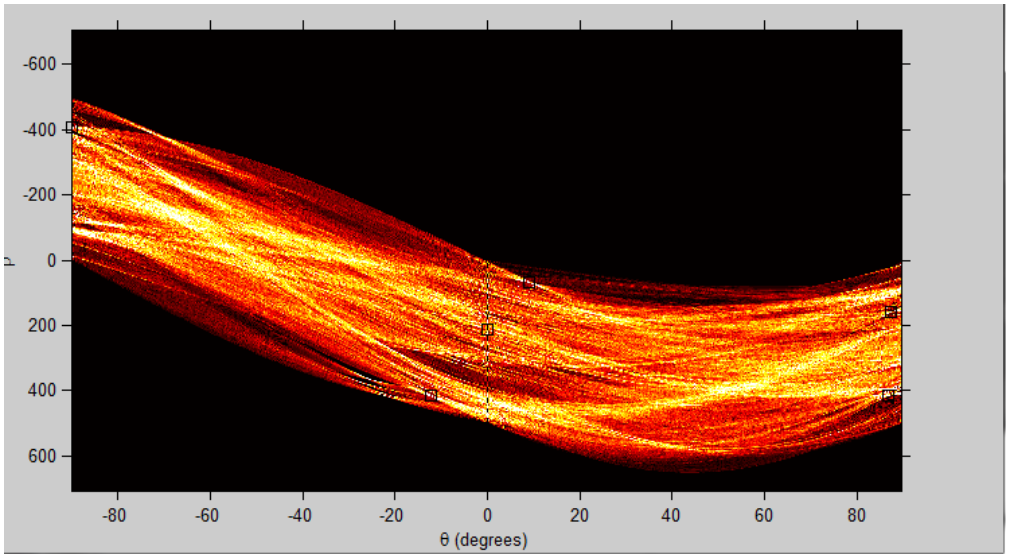

Figure 12. Hough Transformation of Figure 11

Figure 13 shows the effectiveness of the algorithm in by producing NN outputs that are close to 1. The first 6 frames of the simulation show that the output values of the NN vary between 0.99915 and 0.99958. The all output results for each of the 100 frames is shown in Table 1 below.

Figure 13. Simulation result of HT in Parallel with BPNN

In Table 1, the first row represents the first 10 outputs of the NN, where the second row represents the outputs from 11 to 20, with 10 rows in total; all 100 outputs can be read to prove the efficiency.

0.99958	0.99943	0.99915	0.99916	0.99939	0.99927	0.99920	0.99935	0.99973	0.99946
0.99922	0.99916	0.99965	0.99934	0.99987	0.99993	0.99958	0.99945	0.99949	0.99939
0.99984	0.99959	0.99933	0.99985	0.99994	0.99956	0.99912	0.99901	0.99952	0.99932
0.99946	0.99974	0.99927	0.99999	0.99957	0.99984	0.99922	0.99971	0.99943	0.99968
0.99955	0.99989	0.99931	0.99976	0.99922	0.99947	0.99973	0.99958	0.99944	0.99938
0.99966	0.99963	0.99959	0.99937	0.99949	0.99931	0.99989	0.99943	0.99968	0.99969
0.99911	0.99927	0.99914	0.99938	0.99919	0.99957	0.99944	0.99929	0.99912	0.99955
0.99964	0.99982	0.99931	0.99969	0.99922	0.99983	0.99981	0.99914	0.99911	0.99991
0.99978	0.99951	0.99982	0.99931	0.99990	0.99953	0.99921	0.99977	0.99915	0.99992
0.99958	0.99932	0.99971	0.99958	0.99933	0.99919	0.99915	0.99943	0.99973	0.99929

Table 1. Complete Neural Network Output of 100 frames

Now we can test the new algorithm with different background and lighting conditions to figure out its efficiency. After moving the input camera and changing the background conditions, it is found that the algorithm still works perfectly and results in outputs that are bigger than 0.8. Figure 14 represents the output of the NN in a different environment.

Figure 14. Simulation of HT with BPNN in a different environment

After testing the algorithm in a different environment to figure out its dependency on the background objects, it is found that a background object with straight lines or any other straight-line disruption can lead to misdetection of the desired rectangle in the image since the algorithm focuses on finding 6 peaks in total. In order to analyze the role of other straight-line segments in the current frame, the simulation is run especially in a scenario with straight-line segments in the background. The picture frames in the background are also shaped in rectangle and offers a more challenging simulation. As we can see in Figure 15, although the desired license tag is within sight of angle of the camera, the NN gives us the results closer to 0, which implies that the tag is not detected.

Figure 15. Simulation of HT with BPNN with background straight lines

In order to analyze the detected HT lines that might be the cause of the failure, the same simulation is run with the HT lines shown on the simulation itself. As it can be seen in Figure 16, the detected HT lines are not covering all the sides of tag itself thus leading the false area extraction as the input of NN.

Figure 16. Hough Lines imposed on Figure 15

A way to overcome the problem is increasing the number of peaks to be detected in the hopes of finding the four lines that are surrounding the sides of the license tag and extracting the correct are covering the entire tag. After setting the number of Hough Peaks to be detected as high as 10, the same simulation with the same background conditions is run. Although the number of Hough Lines detected is increased, In Figure 17 we can see that there are still misdetections that might lead to false area extraction as the input of NN. In order to tackle this drawback of HT, there are many methods proposed in the literature for an improved detection. In the Conclusion and Future work section; new ideas, works and literature reviews in order to improve the performance of Hough Transform will be studied excessively.

Figure 17. Hough Peaks with increased number of 10

In order to test the limits of the algorithm, salt&pepper noise is applied with increasing incremental. It is found that the detection method fails as the applied density has been increased to 0.2. The graphical representation of the salt&pepper density and the results are shown in Figure 18 below.

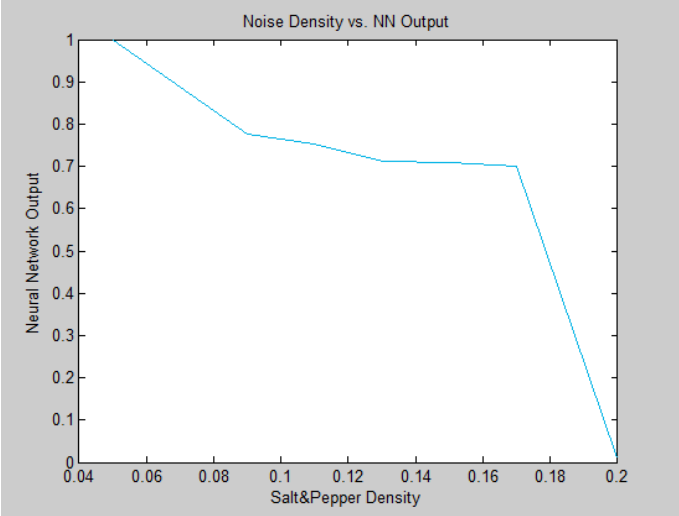

Figure 18. Salt&Pepper Density vs. BPNN Output

5.2 USING ROBOREALM FOR HOUGH TRANSFORM

MATLAB is a powerful computational tool to analyze and simulate the computer vision algorithms since it gives users the freedom to manipulate the codes and algorithm down to each pixel. On the other hand, RoboRealm is a user-friendly tool, which needs no coding and provides built-in controlling structure for many of Robotics structure. It is useful to make a comparison of the simulation results of MATLAB and RoboRealm to see if Hough Transform with the same parameters gives more improved results than MATLAB. The same captured scene with the faulty HT detection is simulated on RoboRealm. First, we need to apply edge detection in order to perform HT on the image. After loading the image on RoboRealm by using "*Open"* tab, under the "*Contents"* tab we can find *"Edges"* section for the Canny Edge detection.

Canny Edge detection is one of the most efficient ways to detect true edges. According to Gupta, unlike other edge detection methods, Canny Edge detection method

is highly efficient in finding the true weak edges. Other classical edge detection methods like Sobel and Prewitt tend to detect thick and inaccurate edges and to be more sensitive towards noise. Another method, called Laplacian of Gaussian, usually malfunctions at the corners, curves and there the gray level intensity varies. It is also weak at finding the orientation of the edge because Laplacian filter is not suitable for this purpose. (DGW-Canny)

On the other hand, Canny Edge Detection is an edge detection method that has the minimal edge detection concerns but usually has the drawbacks of computational load and complexity. It can also have the confusion at the corners since it exploits the Gaussian smoothing of the image. (DGW-Canny)

After choosing Canny Edge detection method in RoboRealm, Gaussian Theta is chosen as 0.8, Low Threshold and High Threshold are set as 0 and 34 respectively. Unlike MATLAB, where the threshold values are assigned automatically unless specified, we can adjust the values in RoboRealm to get the closest edge detection result with MATLAB. The values specified above are the ones that give the closest edge detection with the previous simulation.

After applying Canny Edge detection on image, it is convenient to apply HT on the binary image. *"Hough"* function in RoboRealm is a user-friendly function that is used to perform HT on binary images. Choosing the same values with the ones in MATLAB, a more accurate but not sufficient detection of the tag is achieved. Figure 19 below illustrates a more accurate detection of the tag as the input of NN.

Figure 19. Hough Lines on RoboRealm

Figure 19 clearly shows that although RoboRealm provides a better encapsulation of the license tag, there is still some misdetection. The Hough Line, which covers the right side of the license tag, is placed right next to the number “0”, detecting it as a straight line. Standard HT method is so useful to detect straight lines in a binary image but also needs improvement.

Figure 20 below provides a better illustration of misdetection of HT lines on the original image. It is interesting that MATLAB simulation provided Hough lines on the sides of the picture frames whereas RoboRealm is more successful and accurate for detecting the desired sides of the license tag. The reason of difference underlies in the binary images produced by both MATLAB and RoboRealm.

Figure 20. Hough Lines on the Original Image on RoboRealm

6. CONCLUSION AND DISCUSSIONS

In this study, the benefits of HT in object recognition, in our case a license tag, using BPNN is examined. First 251 different scenarios with the desired license tag in it used as a positive input to train the network, which are supposed to yield 1 in the output of NN. Then 168 different scenarios, without the desired license tag in it, are used to train the NN, which are supposed to yield 0 in the output. Providing the Confusion Matrix, which implies there is 0% error in testing and validation, proves the accuracy of the trained NN. After many simulations, it is observed that BPNN is only successful at recognizing the tag if the simulation is run in the same conditions with the trained pictures. A change in the background objects, or a change in the simulation environment, led to the misdetection of the desired object, the license tag. The importance of extracting the license tag without being dependent on the background objects and lighting conditions gains importance and HT is a useful tool to detect the straight lines in a binary image. With using HT, it is aimed to detect the four straight lines for each side the license tag. In the algorithm, if there are 4 Hough lines detected, intersecting each other with at least 80 and at most 100 degrees, then the area that has been bounded by these 4 lines are extracted and used as an input in a newly trained NN, where the solely the image of the license tag with different angles and lighting conditions are used. It has been proven that using HT in parallel with BPNN provides a great advantage over using BPNN in terms of detecting the license tag. Adapting HT minimizes different lighting conditions and background disruptions.

On the other hand, HT has downsides when there are other straight-line segments in the background. A simulation on MATLAB, where there were other picture frames in the background, has been run with the desired license tag in it. It has been observed that NN failed to produce outputs, which were supposed to be close to 1, since the desired tag was in the scene. In order to analyze the reason of the failure, Hough lines were printed on the image and it is found that other straight-line segments in the image led to the failure of extraction of the desired tag.

RoboRealm is also a very useful and user-friendly interface mainly used for the robotics vision and image processing. Its built-in robot controllers and low memory

consumption provide advantage over MATLAB. The same captured frame with HT failure has been run on RoboRealm this time to make a comparison with the previous simulation that has been run on MATLAB. It has been observed that RoboRealm is more successful in detecting the sides of the license tag and being resistant to background noise, which are mainly straight lines.

7. FUTURE WORK

Still the straight-line detection for the desired tag's sides is not perfect with HT on RoboRealm. There are many other techniques in the literature for improved HT, which can be applied to this study as a future work. One of them is using a new and improved approach of HT. [Nguyen]. According to Nguyen, three extensions to SHT provide better detection of straight-line segments. First one is the accumulation technique, second one is applying the local maxima rule and finally third one is the detection of line segments. The technique of accumulation consists of accumulating the pixel of the image only in the parameter space. [Nguyen] .We put the coordinates (x,y) of a pixel into Equation 2 and then ρ value, which will be referred as ρ_{exact} , is calculated.

If the location of a pixel is (x_i, y_i) , then the Equation 2 turns into

$$\rho_{exact} = x_i \cos\Theta + y_i \sin\Theta \qquad \text{Eq. 3}$$

Then we can consider there are two consecutive values in the ρ-axis, ρ_{low} and ρ_{high}, where $\rho_{low} \leq \rho_{exact} < \rho_{high}$. We can calculate the Hough transform for ρ_{high} and ρ_{low} as follows: [Nguyen2]

$$H(\rho_{low},\Theta)=H(p_{low},\Theta) + (\rho_{high}-\rho_{exact}) \qquad \text{Eq. 4}$$

$$H(\rho_{high},\Theta) = H(\rho_{high},\Theta) +(\rho_{exact}-\rho_{low}) \qquad \text{Eq. 5}$$

Next step to improve the HT is applying the local maxima rule. In Standard HT, the certain lines are selected only if their values are above a certain threshold. The problem with the SHT is that, the extracted lines might end up being too close to each other and other straight line-segments in the rest of the image might go unnoticed. Eliminating the unsatisfying lines by detecting the maximum value in the local area is the core of applying local maxima. Final step is the extraction of the accurate line segments. This method not only provides the detection of the straight lines but the line segments having accurate positions on the image. For future work, applying these extensions to extract the desired license tag from the scene to use it as the input of NN might provide a great advantage and lead to error free detection of the desired objects in parallel with BPNN.

REFERENCES

[1] A. Gupta, R. D. (2011, December). DOW-Canny: An Improvised Version Of Canny Edge Detector . *International Symposium on Intelligent Signal Processing and Communication Systems* , 6.

[2] B. Widrow, M. L. (1990). 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation . 28.

[3] Barinova, O., Lempitsky, V., & Kholi, P. (2012). On Detection of Multiple Object Instances Using Hough Transforms. *PIEEE Transactions on Pattern Analysis and Machine Intelligence* , 1773-1784.

[4] C. Gentsos, C. S. (2010). Real-Time Canny Edge Detection Parallel Implementation for FPGAs . *IEEE* , 4.

[5] C. Wang, F. W. (2009). A Knowledge-Based Strategy for Object Recognition and Reconstruction . *International Conference on Information Technology and Computer Science* , 387-391.

[6] D. Duan, M. Q. (2010). An Improved Hough Transform for Line Detection . *International Conference on Computer Applications and System Modeling* , 4.

[7] Datta, S., Khasnobish, A., Konar, A., & Tibarewala, D. (2013). Object shape and size recognition from tactile images. *International Conference on Control Communication and Computing* , 16-21.

[8] Diplaros, A., Gevers, T., & Patras, I. (2006). Combining color and shape information for illumination-viewpoint invariant object recognition. *IEEE Transactions on Image Processing* , 1-11.

[9] H. Koshimizu, K. M. (1989, April). Global Feature Extraction Using Efficient Hough Transform. *International Workshop on Industrial Applications of Machine Intelligence and Vision* .

[10] H. Zhao, G. Q. (2010). Improvement of Canny Algorithm Based on Pavement Edge Detection . *3rd International Congress on Image and Signal Processing* , 5.

[11] J. Lee, S. L. (2007). Object Recognition Architecture Using Distributed and Parallel Computing with Collaborator. *IEEE International Conference on Granular Computing* , 490.

[12] J. Oh, G. K. (2012). Low-Power, Real-Time Object-Recognition Processors for Mobile Vision Systems. *Micro, IEEE , 32* (6), 38-50.

[13] J.P.N Cruz, M. D. (2013). Object recognition and detection by shape and color pattern recognition utilizing Artificial Neural Networks. *International Conference of Information and Communication Technology* , 140-144.

[14] L. Chandrasekar, G. D. (2014). Implementation of Hough Transform for image processing applications. *Conference on Communications and Signal Processing (ICCSP), 2014 International* , 843-847.

[15] L. Liu, Z. C. (2009). An Improvement of Hough Transform for Building Feature map. *International Conference on Computational Intelligence and Software Engineering* , 1-4.

[16] M. Li, D. L. (2013). Vision-based robot navigation using parallel Hough transform. *Conference on Machine Learning and Cybernetics (ICMLC), 2013 International* , 1519-1526.

[17] M. Nakanishi, T. O. (1997). Real-time Extraction Using a Highly Parallel Hough Transform Board. *IEEE* , 4.

[18] N. Aggarwak, W. K. (2006). Line detection in images through regularized hough transform. *IEEE Transactions on Image Processing* , 582-591.

[19] Park, J. (2007). A new object recognition system for service robots in the smart environment. *International Conference on Control, Automation and Systems* , 1083-1087.

[20] S. Lee, H. S. (2010). Implementation of Lane Detection System using Optimized Hough Transform Circuit. *IEEE* .

[21] S. Maruno, T. I. (1993). Object recognition system using temporal pattern recognition networks with quantizer neuron chip. *International Joint Conference on Neural Networks , 2*, 1285-1288.

[22] Sato, N., & Hagiwara, M. (2000). A visual nervous network for moving object recognition. *IEEE International Conference on Systems, Man, and Cybernetics* , 1.

[23] Sidhu, R. (2014). Improved Canny Edge Detector in Various Color Spaces. *IEEE* , 6.

[24] T. Nguyen, X. P. (2008). An improvement of the Standard Hough Transform to detect line segments. *IEEE* , 6.

[25] T. Nyugen, X. P. (2008, July). A Test Framework for the Accuracy of Line Detection by Hough Transforms. *The IEEE International Conference on Industrial Informatics* , 6.

[26] X. Benxian, L. C. (2008). Moving object detection and recognition based on the frame difference algorithm and moment invariant features. *27th Chinese Control Conference* , 578-581.

[27] X. He, W. J. (2008, December). An Approach of Canny Edge Detection with Virtual Hexagonal Image Structure. *10th International Conference on Control, Automation, Robotics and Vision* , 4.

[28] X. Li, T. Z. (2014). A recognition method of plate shape defect based on RBF-BP neural network optimized by genetic algorithm. *The 26th Chinese Control and Decision Conference* , 3992-3996.

[29] X. Li, Z. L. (2010). The establishment of AHP-BP neural network model and its application. *2nd International Conference on Information Science and Engineering (ICISE)* , 4791-4794.

[30] Z. Xu, B. S. (2015). Accurate and Robust Line Segment Extraction Using Minimum Entropy With Hough Transform. *IEEE Transactions on Image Processing* , 813-822.