The International Journal Of Engineering And Science (IJES)
|| Volume || 5 || Issue || 1 || Pages || PP -22-29|| 2016 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

A Back-Propagation Neural Network for Detection of Vertical Sub-Lines

Gideon Kanji Damaryam
Department of Computer Science, Federal University, Lokoja, Nigeria.

----------**ABSTRACT**----------
This paper presents a back propagation artificial neural network for detection of vertical sub-lines from sub-images of images captured by a mobile robot for the purpose of self-navigation. The network is intended to be part of a system of networks for recognising various kinds of sub-lines, and the results from recognitions at this stage is intended to be further processed to complete a mobile-robot self-navigation system.

Back propagation networks are very widely used artificial neural networks. In this work they are used to recognise vertical sub-lines. Recognition of sub-lines has been achieved by analytic means. This paper presents part of the process of investigating the feasibility of replacing some parts, or all of such analytical methods, which can be very resource intensive, with back propagation networks.

Data for training and testing of the back propagation network is derived by partitioning pre-processed images into 8 pixel by 8 pixel sized sub-images, and using the individual pixels as input to the network. Training details are presented as well as sample test results.
Keywords: *artificial intelligence, artificial neural networks, back propagation network, line detection*

Date of Submission: 08January 2016 Date of Accepted: 18 January 2016

I. Introduction

This paper presents a back propagation artificial neural network for detection of vertical sub-lines from sub-images of images captured by a mobile robot for the purpose of auto-navigation. After capture, the images arepre-processed according to a scheme described in [1]. In a nut-shell, the pre-processing scheme converts captured images to gray-scale, resizes them to a 128 pixel by 96 pixel size, detects edges in them using the Sobel edge-detection operators, and thins them using a new method developed as part of this work. Fig. 1 show a sample re-sized image (a) and the same image after it has been pre-processed (b).

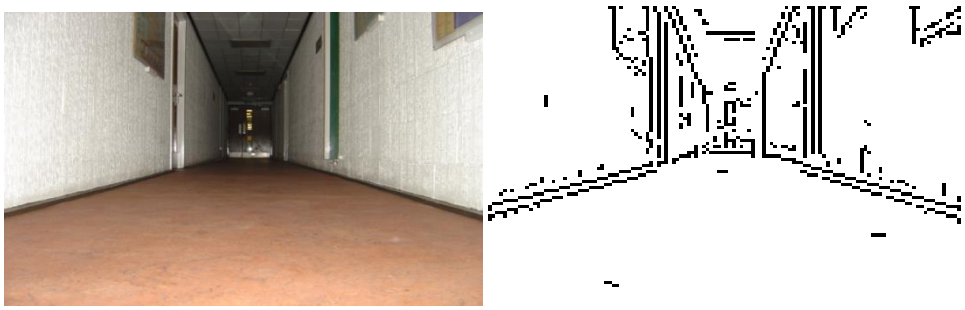

Figure 1 Sample resized image, and corresponding thinned image
(a) Sample resized image (b) Sample thinned image

While it is not necessary to pre-process images in this way in preparation for recognition of lines with an artificial neural network such as the back propagation network, this scheme was used as the work described in this paper was part of a bigger project to investigate the feasibility of hybrid Hough transform / artificial neural network vision systems. Use of the Hough transform to detect lines and subsequently find valid sub-lines have been described in [2] and [3] respectively. In this paper, a back propagation network is used to try to detect sub-lines also, with the broader goal of substituting some stages of the Hough transform system with the back-propagation. The hybrid system will be presented in a future paper. This pre-processing scheme used for the back propagation detection of sub-lines is the same one used for the Hough transform method for this reason.

Before going into details of the specific networks in this work, a quick overview of artificial neural networks culled from descriptions in [4], [5], and [6] follows.

1.1 Artificial Neural Networks

Artificial neural networks (ANNs) are commonly used in artificial intelligence systems to enable machines to learn to recognise and classify input patterns from a few representative samples. They are also called simulated neural networks (SNN) or just neural networks (NN).

There are many different types of ANNs. Typically, they consist of several artificial neurons which are mathematical models of biological neurons. Artificial neurons are 'connected' by some algorithm which governs the training of the network and processing of data in the network. They consist of mathematical models called artificial neurons designed to take in numeric input corresponding to some input pattern, process these and give outputs depending on input values in a manner imitating the functioning of biological neurons.

Artificial neurons and neural networks have been implemented in hardware as electronic circuits, often integrated on microchips, as well as software. Software implementations have the advantage of being much more flexible, and portable, and less expensive.A software implementation was used for this work. Hardware implementations can be advantageous in applications where very high speed of processing is an important issue as they are generally faster.

1.2 The Back Propagation Network

This work uses a type of ANNcalled the back-propagation network (BPN). BPNs are probably the most common types of artificial neural networks in use, and often the expression artificial neural networks in literature actually refers to back-propagation networks. This attitude is adopted in this work, so the expression artificial neural network will mean back-propagation artificial neural networks except when it is necessary to make a distinction.

Back-propagation networks consist of a number of artificial neurons organised into at least three layers. Fig. 2 illustrates a typical back propagation network.

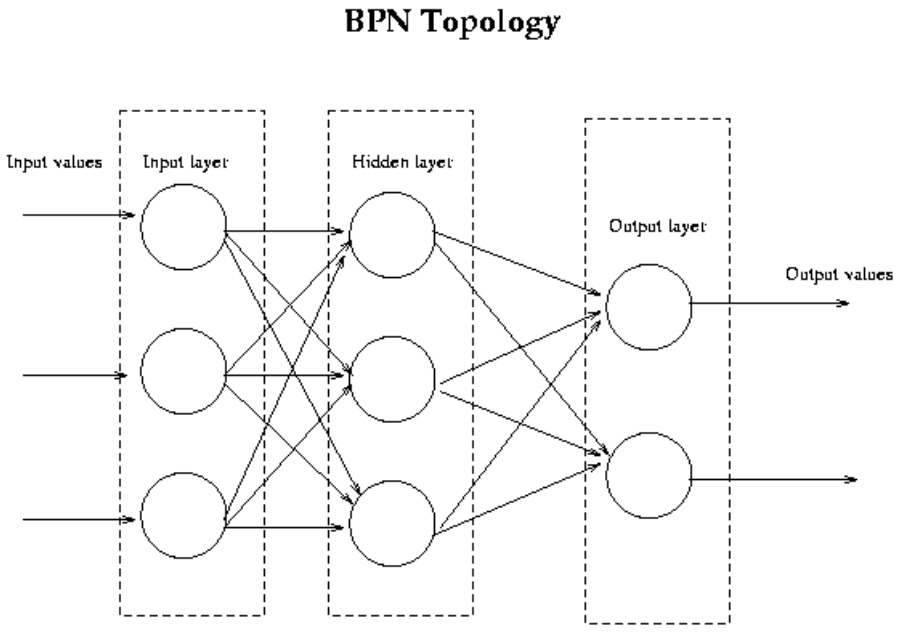

Figure 2 Neurons arranged in layers in a typical back propagation network
(Source: [7])

A layer referred to as the input layer has neurons equal in number to the number of inputs to the network and each neuron in this layer is fed with one of the inputs to the network when the network is in use. The input layer then passes whatever input it has received to one or more inner layers (also called hidden layers) applying weights to them, whose values would have been determined in the course of training of the network. Neurons in the inner layer sum the weighted inputs they receive to obtain a 'net', and then pass the value of 'net' through a squashing function which limits the range of possibilities of output from these neurons. This work employs the sigmoid function which limits output to a continuous range between 0 and 1, and is a commonly used squashing function for back propagation neural networks.

Before the network can be put to use, it needs to be trained. Training of a back propagation network is done by presenting a set of inputs and corresponding desired outputs to the network. This set is called the training set. A set of weights for the links in the network would have been chosen randomly initially. The inputs to the network are first received by the input neurons, which passes them on to the inner layers applying the current weights to them. The inner layers compute the weighted sums and apply the squashing function to obtain their outputs. The outputs to the current inner layer are passed on to the next inner layer if there is any, or to the output layer. The output layer performs the last round of information processing and calculates its output which is also the output of the network. The values of the calculated outputs are compared to the values of the desired or target outputs for that input pattern. The differences or errors are then propagated back through the network adjusting the weights of the links. The network type gets its name from this. The next input pattern in the training set is then passed to the network and the errors are again propagated back and the weights adjusted.

Two parameters, the learning rate, and the momentum term are sometimes introduced. The learning rate is introduced to alter the rate at which training is done, which may be helpful if, for example, the rate of learning is deemed to be so fast it could miss optimal solutions. The momentum term is sometimes introduced to "shake things up" a bit. It scatters the progress being made a little bit and could be helpful to prevent the training process to converge to a solution that may be optimal within a small locality, but not on the bigger scheme of things.

This process is repeated several times with all the input patterns in the training set. An epoch is then said to have been completed. Several epochs are usually done until the errors come below a predefined level. The final weights are then saved, and are used when the network is put to use.

It is necessary to have a maximum number of epochs allowable, so that attempts to train do no continue indefinitely. When this maximum number is reached, the process is stopped. Changes can be made such as alteration of the networks topology, or introduction and/or alteration of parameters like the learning rate and momentum term.

The use of BPNs, like ANNs in general, is growing rapidly. They are widely used in image processing for recognition of hand-written characters, faces, fingerprints, gaits, etc., and in visual search engines. They are also used for voice recognition, speech production, RADAR signature analysis and stock market prediction. BPNs are also becoming increasingly useful in robotics. Some robotics tasks where BPN are useful include processing of accelerometer system data (used for balance in two-legged robots), processing of voice commands, navigation, vision, etc.They are also used in manufacturing industries for control, and in business, for mortgage decisions, for example.

II. Application of BPNs in this Work

Recognition of vertical lines in this work involved breaking an image down to sub-images, and then determining whether the sub-image contains a vertical line using neural networks. This in a way mimics a stage of line recognition using what is called the hierarchical Hough transform([8]), and in another mimics detection of sub-lines as described in [4], after lines have been detected using the Hough transform as described in [2].

2.3 Data Generation

Data for training and exercising of the back-propagation network is derived from breaking down pre-processed images such as in Fig.3 into sub-images of size 8 pixels by 8 pixels. This yields a total of 192 sub-images – 16 across and 12 down the full image. This is illustrated in Fig.3 on the right.

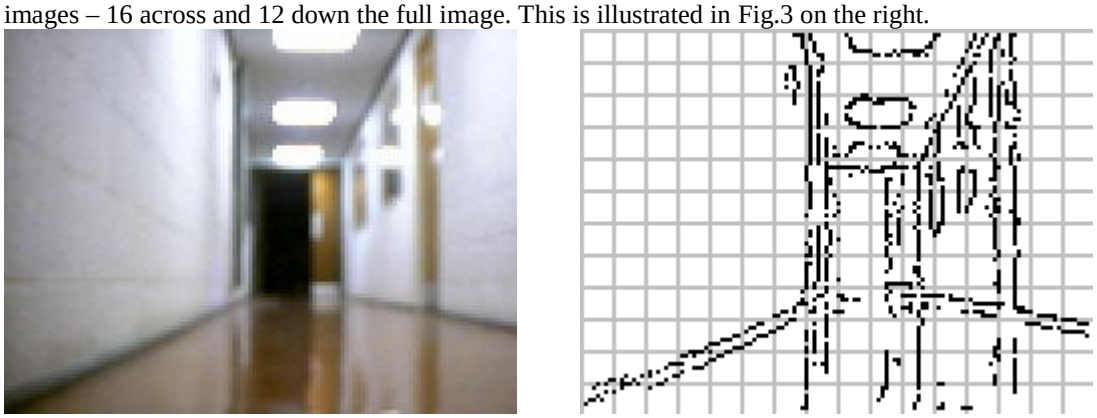

Figure 3 Pre-processed image broken down into 8x8 sized sub-images

Sub-images are labelled with identification codes illustrated in Fig.4. The sub-image at the top-left position is labelled 0. Subsequent sub-images going right are labelled with consecutive numbers until the end of the row. The labelling is continued on the next row from the left.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Figure 4 Sub-image labelling order

Each sub-image consists of 64 pixels and those constituted input to the neural network used to determine vertical lines. Output from the network is a binary digit which indicates whether the sub-image contains a vertical line or not.

2.2 Training for the Back-Propagation Network Used

1.2 The Back Propagation Network introduced the back propagation network, and how it has been used in this work. This section provides more details of this. Most of the implementation in this work is based on [9], and even further details can be found there.

The training method used is summarised as follows:

```
backPropTraining
{
        initialiseiterationCount to 0
        whilenumOfTrainedPatterns<NumInTraining
        {
                initialisenumOfTrainedPatterns to 0
                forallpatternsInTraining, p
                {
                        place p on the network
                        do forward pass
                        determine error for p
                        do backward pass
                        if error for p is less than threshold
                                increasenumOfTrainedPatterns by 1
                }
                IncreaseiterationCount by 1

                IfiterationCount == maxNumOfIterationsAllowed
                        break
        }

ifiterationCount<maxNumOfIterationsAllowed
        save network parameters
else
        declare that training failed
}//end backPropTraining
```

Inputs to the process include a training set, and a network set up with random weights for its links.

At the heart of the process are two loops, one nested in the other. They are detailed further in the sub-sections *2.2.1 Outer Loop*, and *2.2.2 Inner Loop* below.

2.2.1 Outer Loop

The outer loop, shown above as a while loop, runs until every pattern p, in the training set conforms to the training criteria, i.e., yields an error when passed through the current network, which is less than a pre-defined threshold. In other words, the outer loop runs until the number of patterns that have conformed, or have been trained, numOfTrainedPatterns, equals the total number of patterns NumInTraining. The loop also increments iterationCount by 1 each time it is run, and monitors it so it does not go beyond a predetermined threshold, maxNumOfIterationsAllowed. iterationCount is initialised to 0 before the outer loop starts, and if it does get to maxNumOfIterationsAllowed, the training process is halted and training is deemed to have failed.

2.2.2 Inner Loop

The inner loop passes individual patterns forward through the network, determines whether or not the error from the pass is lower than the error threshold, and update the count of trained patterns, numOfTrainedPatterns. It also does a back pass which adjusts the weights of the network to 'fit in' the current pattern better.The forward pass, as mentioned in *1.2 The Back Propagation Network,* uses the sigmoid function to assign values to nodes. This function is shown:

$$y = \frac{1}{1 + e^{-NET}} \quad . \quad . \quad . \quad (1)$$

where NET for a particular node is the sum of the product of the weight of a link coming into that node, and the value of the node that the particular link originates from, for all links. Value is determined for all nodes except for those in the input layer.

Errors are determined for output nodes by subtracting the actual outputs from the node from the target outputs, and for the pattern by summing the absolute value of all the errors of its output nodes. Patterns with errors exceeding a predefined threshold are counted using the variable numOfTrainedPatterns, as pointed out earlier in *2.1.1 Outer Loop*.

The most defining step of the training in the back-propagation method is the back pass. It involves propagating the error for the pattern back through the network by adjusting the weights on its links using what is known as the delta rule. By this rule, an adjustment, Δ_i, is worked out for each link i using

$$\Delta_i = \eta x_i \delta_i \quad . \quad . \quad . \quad (2)$$

where

$$\delta_i = y_i(1 - y_i)(d_i - y_i) \quad . \quad . \quad . \quad (3)$$

for the output neurons, and

$$\delta_p(q) = x_p(q)[1 - x_p(q)]\sum w_{p+1}(q,i)\delta_{p+1}(i) \quad . \quad . \quad . \quad (4)$$

for neuron q in hidden layer p .

η in(2) is the learning rate, introduced in *1.2 The Back Propagation Network*. y_i and d_i are actual and desired outputs respectively, in (3). For hidden layer neurons, $w_{p+1}(q,i)$ is the weight of the link ending in layer $p+1$ (the next layer from the current one, p), starting from node q in layer p and ending in node i (which is in layer $p+1$).

2.2.3 Recognition of Vertical Lines from Sub-Images

A network was set up to recognise vertical lines, from 8 pixel x 8 pixel sized sub-images extracted by breaking an image down. The network therefore has 64 binary inputs. It has a single output which has a value of 1 if a vertical line is detected in the input sub-image and 0 otherwise. The network also has 1 inner layer with 9 neurons.A training set was developed incrementally from sub-images taken from 5 randomly selected images. Training was performed, and the network was tested with a fresh random image. Sub-images which are not correctly identified are added to the training set, and the network is re-trained. This was done until further additions to the training set did not significantly improve the recognition rate in fresh random images. 216 sub-images were derived in this way.

For the purpose of this work, a vertical line is defined as a line in the range -5 degrees to 4 degrees to the vertical.

III. Sample Test Results

Summary of results from testing all 192 sub-images from two random images, test image 1 and test image 2, follows. (Both test images were not used for training.) Test image 1 is illustrated in Fig. 5(a), the pre-processed version of it in Fig. 5(b) and the 8x8 pixel sized sub-images derived from it in Fig. 5(c). Note that lines of the partitions have covered some data.

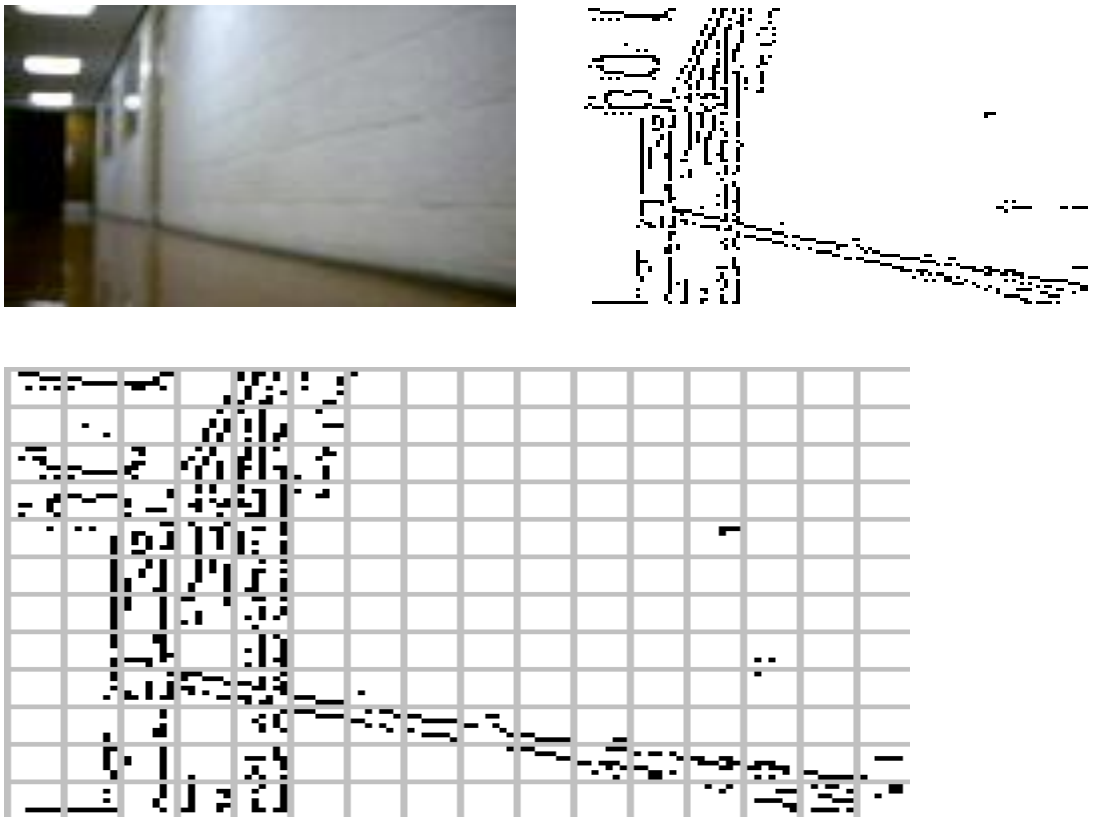

Figure 5 Test Image 1
(a) Captured image (b) Pre-processed version (c) Partitioned version showing sub-images

Table 1 summarizes the result for test image 1.

Table 1 Summary of Results for Test Image 1

	Neural Network Outcome		
Actual		Vertical Line	No Vertical Line
	Vertical Line	27	8
	No Vertical Line	5	152

Test image 2 sub-images are illustrated in Fig. 6.

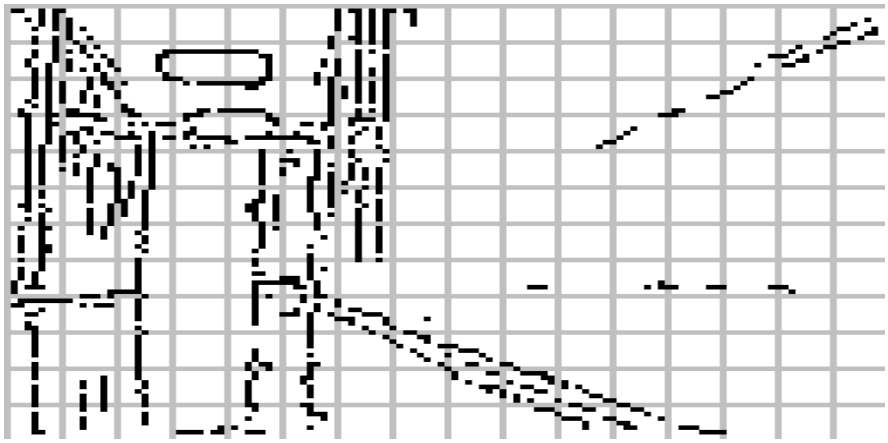

Figure 6 Test image 2 sub-images

Results for test image 2 are summarized in Table 2.

Table 2 Summary of Results for Test Image 2

	Neural Network Outcome		
Actual		Vertical Line	No Vertical Line
	Vertical Line	44	5
	No Vertical Line	13	130

Results of this test are illustrated in Fig. 7. Red lines in a sub-image indicate that a vertical line was found in the sub-image. Note that the red lines are drawn in the middle of the sub-image and do not indicate the actual positions within the sub-images where the lines were found. Information about the actual position or arrangement of pixels in the line, or whether more than one vertical line exists, is not obtained when using a BPN as described in this paper, unlike, say when using the Hough transform as described in [2] and [3].

Figure 7 Results of test for vertical category

Table 3. Some Measures of Performance of the Network

	Test Image 1	**Test Image 2**	**Average**
Accuracy	92.23	90.63	92.19
Sensitivity to Vertical Lines	77.14	89.8	83.47
Vertical Lines Positive Predictivity	84.38	77.19	80.785
Sensitivity to Non-Vertical Lines	96.83	90.91	93.875
Non-Vertical Lines Positive Predictivity	95.00	96.3	95.665

IV. Conclusion and Further Work

A back-propagation network was presented for detection of vertical sub-lines in a pre-processed images, by partitioning images into sub-images. Measures of performance when tested with two random images appear to be reasonably high – averagingpercentages in the eighties and nineties. Sensitivity to vertical lines is not very impressive, however, particularly for the first random image, and that is important.

Further work includes setting up of networks to detect other kinds of (horizontal and slanted) lines, and then going on to the next stage of processing the captured images by consolidating results from the vertical and other line types recognition systems, so they are useful for robot-self navigation, which is the ultimate goal of the work that is the basis for this paper.

V. Acknowledgement

This paper discusses work that was funded by the School of Engineering of the Robert Gordon University, Aberdeen in the United Kingdom, and was done in their lab using their robot and their building.

References

[1]. G. K. Damaryam and H. A. Mani,A Pre-processing Scheme for Line Detection with the Hough Transform for Mobile Robot Self-Navigation , In Press, *International Organisation for Scientific Research – Journal of Computer Engineering, 18(1)*, 2016

[2]. G. K. Damaryam,A Hough Transform Implementation for Line Detection for a Mobile Robot Self-Navigation System, *International Organisation for Scientific Research – Journal of Computer Engineering, 17(6)*, 2015

[3]. G. K. Damaryam, A method to determine end-points of straight lines detected using the Hough transform, *International Journal of Engineering Research and Applications, 6(1)*, 2016

[4]. G. Orr, N. Schraudolph and C. Fred, Neural Network Lecture Notes, Web Page, Accessed January 6, 2016, http://www.willamette.edu/~gorr/classes/cs449/brain.html
[5]. A. Jagadeesan, Real time evolutionary algorithms in robotic neural control systems, doctoral diss., Robert Gordon University, Aberdeen, United Kingdom, 2006.
[6]. Anonymous, Neural Networks, Web Page, Accessed January 6, 2016, http://www.en.wikipedia.org/wiki/Neural_network
[7]. P. Crochat and D. Franklin, Back-propagation Neural Network Tutorial, Web Page, Accessed June 25th, 2007, http://pcrochat.online.fr/webus/tutorial/BPN_tutorial4.html
[8]. C. Espinos and M. A. Perkowski,Hierarchical Hough Transform for Vision of the Psubot, Proc., *Northcon 90 Conference Record*, Portland, OR, 1991, 291-296
[9]. J. Rogers, Object-Oriented Neural Networks in C++, (San Diego: Academic Press, 1996)