IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 2, Ver. III (Mar-Apr. 2016), PP 93-104
www.iosrjournals.org

Detection of Sub-Lines of Various Orientations Using Back Propagation Networks

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

Abstract: *Various approaches can be taken to achieve vision for robot navigation using back-propagation artificial neural networks. An approach is presented in this paper which attempts to detect straight sub-lines from 8x8 pixel sized sub-images of pre-processed images, with the goal of further combining them at a later stage to detect longer lines from across the image (mimicking the hierarchical Hough transform which would find shorter line segments (analytically), and combine them into longer lines), or otherwise advance the result towards generating navigation information for a mobile robot.*

The approach proceeds by trying to find lines belonging to eight pre-defined categories from sub-images using eight back-propagation type artificial neural networks called the stage one networks. Results from these stage one networks would then be passed to other back propagation networks (or other means of further processing) in further stages set up to ultimately work out the direction the robot should move in. The stage one networks are the only ones presented here, and did reasonable well in recognising the target lines.

Keywords:artificial neural networks, back-propagation networks, image processing, line detection, robot navigation*

I. Introduction

Various approaches can be taken to achieve vision for robot navigation using artificial neural networks. A few of them are discussed later in *1.2 Related Work*. This paper presents a method to detect sub-lines using artificial neural networks, from images captured by the camera on a mobile robot, which can be further processed to determine how a mobile robot should proceed navigationally.

Artificial neural networks (ANNs) are often used to provide automated solutions on computing platforms, where an analytical solution does not exist, or is not ideal. They do not require a person (or persons) to analytically develop algorithms to provide solutions, and then implement as a program to generate the solution as would be done conventionally. Instead, a suitable ANN is set up which can take suitably pre-processed samples of the problem, and the correct corresponding solutions for the problem. Counter examples are also provided. The ANN, if successful, is able to automatically determine the correct features to use to map from problem to solution.

In working as they do, ANNs are thought of as learning in the way a biological brain often learns. Human children, for example, mostly learn to identify oranges by being shown examples of oranges, and counter examples like apples and lemons. This paper presents one of several investigations into replacing all or part of an existing analytical method for detecting sub-lines in a digital image, which uses the straight line Hough transform followed by a sub-line detection scheme, with an ANN, because the analytical method is not ideal, in this case, because it can take too much time [1]. The particular investigation presented attempts, as mentioned earlier, to find sub-lines that can be further processed to assist vision-based robot navigation.

The sub-line detection scheme this work somewhat mimics has been presented in [2] and [3]. [2] describes what the Hough transform is and how it has been applied to detect full lines from images captured by a mobile robot for self-navigation within an indoor corridor type environment. [3] presents a method to determine sub-lines of the lines detected from [2].

Before the scheme in [2] can be applied to an image, the image is pre-processed using a scheme detailed in [4] which converts the image to a 128x96 pixel sized binary image showing outlines of boundaries of major regions of the image. The same pre-processing is applied to images prior to application of the methods described in this paper.

Fig. 1 shows a sample image, fig. 2 shows a pre-processed version of the same image using methods from [4], and fig. 3 shows the full lines detected using the scheme from [2]. The lines detected are shown in colours other than black, and they are superimposed on the pre-processed version of the image shown in fig.2.

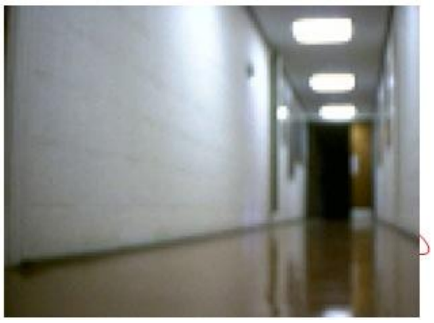
Figure 1: Sample captured image

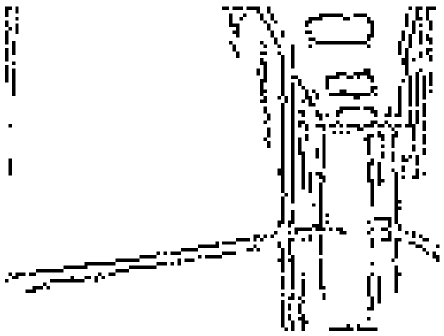
Figure 2: Pre-processed version of fig. 1 sample image

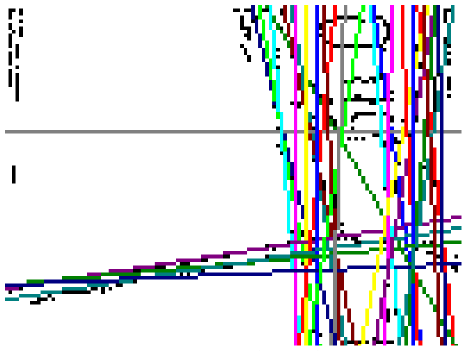
Figure 3: Full lines detected from fig. 1 sample image

Fig. 4 shows sub-lines found using the scheme described in [3]. The goal of the investigation of this paper is to see if ANNs can be trained to detect similar sub-lines.

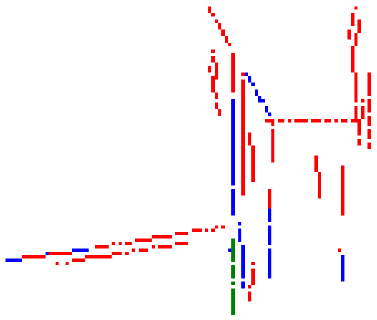
Figure 4: Sub-lines found from fig. 1 sample image

1.1 Back Propagation Networks

ANNs are computer implementations of mathematical models of biological brains. They are made up of artificial neurons, which are mathematical models of biological neurons. They have been described in detail in various publications including [5]. Many types of ANNs exist. The ones used in the investigation presented here are Back Propagation Networks (BPNs). BPNs are multi-layer ANNs whose training algorithm involves adjustment of weights within the network by determining errors in output when a sample input and output has been provided to a network under training, and then propagating the errors from the output back through the network, adjusting weights based on the errors. BPNs have also been described in detail in various publications including [5]. A BPN is illustrated in fig. 5.

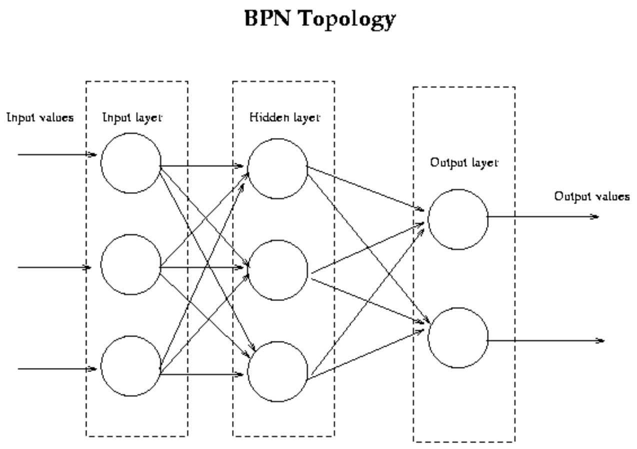

Figure 5: Neurons arranged in layers in a typical back propagation network(Source: [6])

The training method used for the BPN is summarised as follows:

```
backPropTraining
{
    initialise iterationCount to 0
    while numOfTrainedPatterns < NumInTraining
    {
        Initialise numOfTrainedPatterns to 0
        forall patternsInTraining, p
        {
            place p on the network
            do forward pass
            determine error for p
            do backward pass
            if error for p is less than threshold
                increase numOfTrainedPatterns by 1
        }
        increase iterationCount by 1

        if iterationCount == maxNumOfIterationsAllowed
            break
    }
if iterationCount < maxNumOfIterationsAllowed
    save network parameters
else
    declare that training failed
}//end backPropTraining
```

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

At the heart of the process are two loops, one nested in the other.

1.1.1 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 judged to have failed.

Various networks will be considered in the rest of this paper, different specific information about training parameters.

1.1.2 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, 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 all links coming into that node, and the value of the node that the particular link originates from. 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 1.2.1 Outer Loop.

A defining step of the training in the back-propagation method is the back pass. It involves propaging 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

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

where

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

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. 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 $p+1$).

1.2 Related Work

1.2.1 The Work of Chang and Others

The work of [8] discusses a system which enables navigation of a robot in unknown environments. Its navigation controller consists of three sub-controllers – the main controller, the avoidance neural network and the forward neural network. The controllers get input from infrared and ultrasonic sensors. The main controller checks whether or not the area ahead of the robot is safe and transfers control to the forward neural network or the avoidance neural network depending on what it finds.

The system is provided with an initial and a goal position and it works to get from one to the other while avoiding obstacles if necessary. [8] conclude that neural network navigation controllers are efficient, robust and fault-tolerant.

The input to their system, infrared and ultrasound data, are different from the visual data employed in the current work. However, their use of feed forward neural networks and their conclusions about them are of interest to the current work.

1.2.2 The Work of Inigo and Others

The work of [9] is closely related to the current work in the sense that they use a single camera as input to a system they have developed. The system consists of three modules each performing one of three tasks - maintaining alignment, obstacle recognition and determining location of the robot relative to a fixed point of reference.

The system aims at robot navigation using "qualitative navigation behaviour". The networks try to maintain the orientation of the robot while avoiding moving obstacles. The interaction of the three modules resulted in the robot moving in a zig-zag fashion.

A grading system they also developed compares the results of the modules with what a human referee decides is the correct response for the given situation. They report that the results for the system as a whole is better than the results for any of the modules individually.

Their use of neural networks to achieve navigation in a robot relates their work to the current one even though the way they use neural networks (for maintaining alignment, obstacle recognition and determining location) is different from the line detection approach of this work.

1.2.3 The Work of Dempsey and McVey

In an attempt to address the relatively slow processing speed associated with the (hardware) implementations of the Hough transform which has kept it from being widely used despite its importance as a robust and noise-resistant feature identification algorithm, [1] have proposed the use of ANNs-like circuitry to map from image space to parameter space and a modified Hopfield optimisation network to detect peaks in Hough transform parameter space (two important but time consuming steps in the Hough transform). They report tremendous improvement in processing time. The current work only considers software implementations of ANNs.

II. Sub-Lines Detection from Sub-Image

To prepare for the BPN to be used, the pre-processed 128 x 96 sized image obtained from the scheme of [4] is broken down to 8 x 8 sized sub-images. Fig.6 illustrates this.

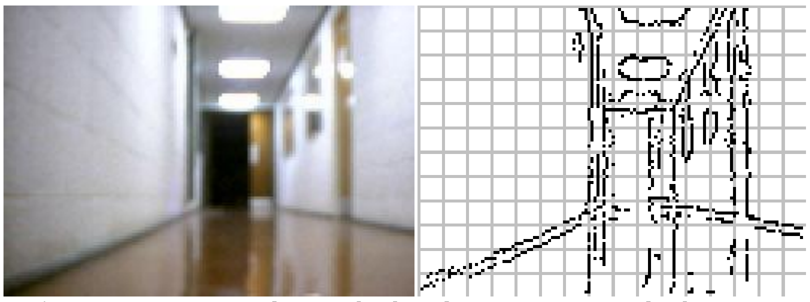

Figure 6: Pre-processed image broken down into 8x8 sized sub-images

Sub-images are labelled with identification codes illustrated in fig. 7. 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 7: Sub-image labelling order

The 64 pixels in the sub-images constituted input to the neural networks used. 8 separate neural networks are trained to determine if the sub-images contain lines from each of the 8 categories detailed in table 1.

Table 1: Lines Categorisation

Category ID	**Description**		**Minimum Θ**	**Maximum Θ**	**Range Size**	**Direction of Distance Measurement**
0	Vertical	|	-5 (or 175)	4	10	left to right
1	Vertical Backslash	\	5	24	20	bottom left to top right
2	Backlash	\	25	64	40	bottom left to top right
3	Horizontal Backslash	\	65	84	20	bottom left to top right
4	Horizontal	—	85	94	10	bottom to top
5	Horizontal slash	/	95	114	20	bottom right to top left
6	Slash	/	115	154	40	bottom right to top left
7	Vertical Slash	/	155	174 (or -6)	20	bottom right to top left

Output from each of these networks is a binary digit which indicates whether the sub-image contains a line of the category the network is trained to detect. These networks are described further in 2.1 Line Recognition and Categorisation from Sub-Images.

2.1 Training for Recognition of Lines in Categories from Sub-Images

8 networks were set up to recognise lines in each of the categories described earlier, from 8 x 8 sized sub-images extracted by breaking the image down. The networks therefore have 64 binary inputs. Each network has a single output which has a value of 1 if a line which falls into the corresponding category is detected in the input sub-image and 0 otherwise. The networks also have 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.

In the training for each network, further sub-images were included which contain clear instances of lines in the category. This was necessary because certain categories do not occur commonly in actual images so there were not enough of them to properly train the networks. The final training set contained 226 to 263 sub-images for the various categories. The target outputs were adjusted appropriately for each sub-image in the training set, when training for each category.

III. Results

Final testing was performed for each category with at least all the 192 sub-images from a complete image.The sub-sections which follow discuss the specific training information and testing results for each line category in more detail.

3.1 Vertical Category Lines Recognition

Lines in this category have θ values in the range -5° to 4°. How lines within this category appear in sub-images is illustrated in the fig. 8. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$. The figure shows various possible appearances of the lines in sub-images, and was used as a guide during training.

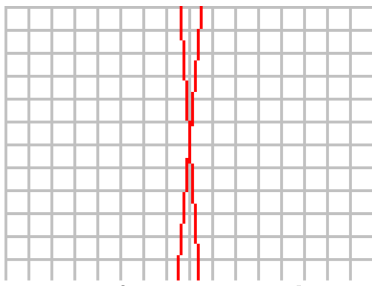

Figure 8: Possible appearances of extreme vertical category lines in sub-images

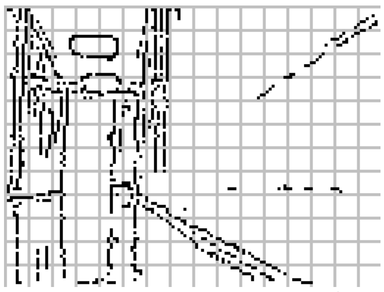
Figure 9: Broken down image used for testing

Fig.10 shows results from testing with image from fig. 9.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 position within the sub-image where the line was found. Information about the actual position or arrangement of pixels in the line, or whether more than one vertical line exists, is not obtained with this approach.

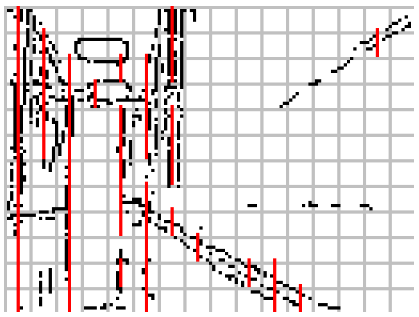
Figure 10: Results of test for vertical category

3.2 Vertical Backslash Category Lines Recognition

Lines in this category have θ values in the range 5° to 24°. How lines with these extreme θ values for this category appear in sub-images is illustrated in the fig. 9. For the purpose of this illustration, they are both drawn to pass through the centre of the image where $\rho = 0$.

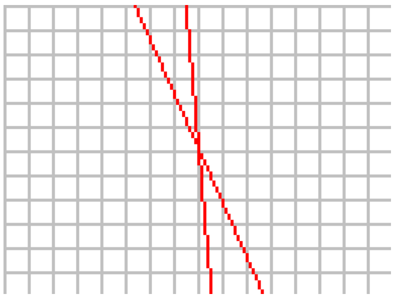
Figure 11: Possible appearances of extreme vertical backslash category lines in sub-images

Note that some lines in this category appear like vertical lines in within some sub-images.
Test results for the random image which is shown in fig. 9 are illustrated in fig.12.

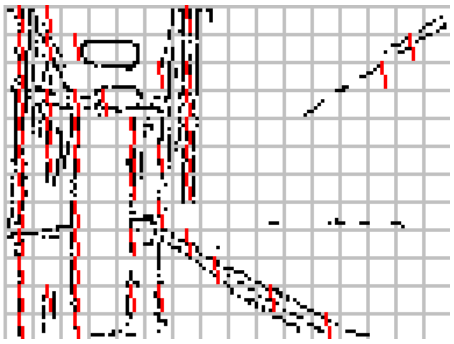
Figure 12: Results of test for vertical backslash category

3.3 Backslash Category Lines Recognition

Lines in this category have θ values in the range 25° to 64°. How lines with these extreme θ values for this category appear in sub-images is illustrated in fig.13. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

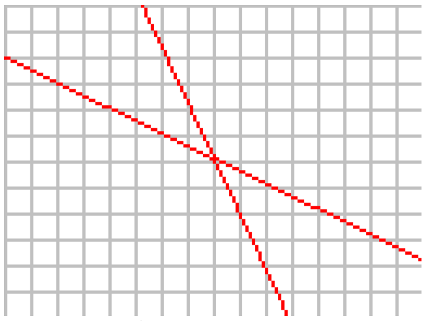

Figure 13: Possible appearances of extreme backslash category lines in sub-images

Test results for the image shown in fig. 9, are illustrated in fig.14.

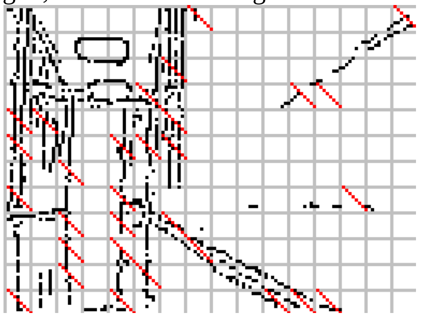

Figure 14: Results of test for backslash category

3.4 Horizontal Backslash Category Lines Recognition

Lines in this category have θ values in the range 65° to 84°. How lines with these extreme θ values for this category appear in sub-images is illustrated in fig.15. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

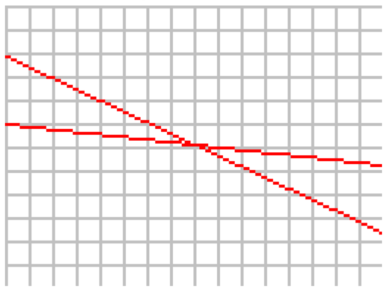

Figure 15: Possible appearances of extreme horizontal backslash category lines in sub-images

Test results for the usual random image, the one shown in fig. 9, are illustrated in fig.16.

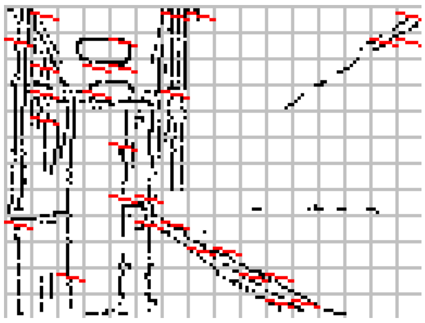

Figure 16: Results of test for horizontal backslash category

3.5 Horizontal Category Lines Recognition

Lines in this category have θ values in the range 85° to 94°. How lines with these extreme θ values for this category appear in sub-images is illustrated in fig.17. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

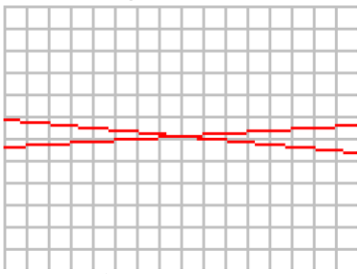

Figure 17: Possible appearances of extreme horizontal category lines in sub-images

Test results for the random image shown in fig. 9, are illustrated in fig.18.

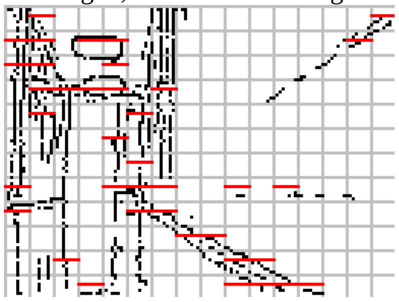

Figure 18: Results of test for backslash category

3.6 Horizontal Slash Category Lines Recognition

Lines in this category have θ values in the range 95° to 114°. Their appearance is similar to the appearance of horizontal backslash lines.

Results for test with the image in fig. 9 are illustrated in fig.19.

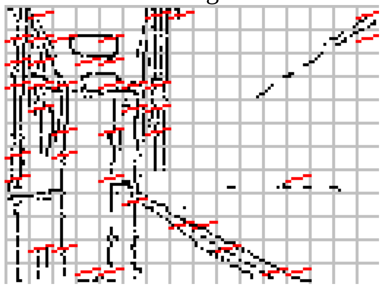

Figure 19: Results of test for horizontal slash category

3.7 Slash Category Lines Recognition

Lines in this category have θ values in the range 115° to 154°. Their appearance is similar to the appearance of backslash lines.

Results for test with the image in fig. 9 are illustrated in fig. 20.

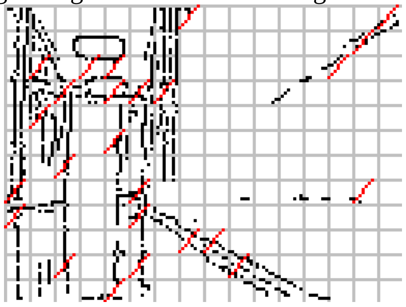

Figure 20: Results of test for slash category

3.8 Vertical Slash Category Lines Recognition

Lines in this category have θ values in the range 155° to 174°. Their appearance can be deduced from the appearance of lines in the vertical backslash category.

Results for test with the image in fig. 9 are illustrated in fig.20

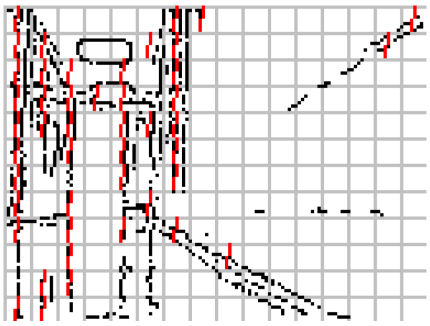

Figure 21: Results of test for vertical slash category

3.9 Aggregated Results

All the results from the various networks are put together in an 8-tuple vector for each sub-image when the networks have all run. Table 2 summarises the configuration for each vector.

Table 1: Configuration of results vector

Vector Position	**Meaning of Possible Value**	
	0	1
0	No vertical line found	Vertical line found
1	No vertical backslash line found	Vertical backslash line found
2	No backslash line found	Backslash line found
3	No horizontal backslash line found	Horizontal backslash line found
4	No horizontal line found	Horizontal line found
5	No horizontal slash line found	Horizontal slash line found
6	No slash line found	Slash line found
7	No vertical slash line found	Vertical slash line found

As an example, the vector 00011100 from a sub-image would mean that for the particular sub-image, a horizontal backslash line, a horizontal line, and a horizontal slash line were found.
An example from an actual sub-image is shown in figure 22. Fig.22a is a typical sub-image, and fig.22b is the results vector obtained when it is processed as described.

```
0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0
0 1 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 1 0 0 0                    1 1 0 0 0 0 0 1
```

Figure 22: Sample combined result of sub-lines detection in sub-image using ANNs

(a) Sample sub-image (b) Sample result vector for fig.22a sub-image
In this example, the result shows that there is a vertical line (the first element of the vector is 1), a vertical backslash line (the second element of the vector is 1) and a vertical slash line (the eight element of the vector is 1).

It is important to remember that although all the lines in the example appear to be vertical lines in the sub-image, they may in fact be parts of a vertical backslash line, or a vertical slash line as suggested by the results vector, and as can be seen by studying fig.9.

The combined results are illustrated graphically in fig.23 below. Note that some of the coloured lines in the fig. overlap.

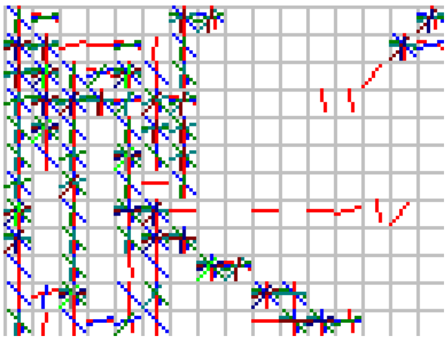
Figure 23: Results from line detecting networks for image in fig. 9

Different colours are used to represent the different lines found in each sub-image. The first line found in each sub-image is coloured red, the second one is blue, the third one is green, etc. Table 3 summarises all the colours corresponding to the orders in which the lines are found:

Table 3: Colours used to indicate different lines found in a sub-image

Line-Find Position	Colour	Sample Line
0	red	🟥
1	blue	🟦
2	green	🟩
3	teal	🟦
4	maroon	🟫
5	navy	🟦
6	lime	🟩
7	dark gray	⬛

Again it should be noted that the system developed cannot specify where in the sub-image a line was found, and all lines are shown centred in the sub-image in which they were found.

The full set of results for the image from fig. 9 is shown in table 4.

Table 4: All Results for Image in Fig 9

Sub-Image Number /Result Vector		Sub-Image Number /Result Vector		Sub-Image Number /Result Vector		Sub-Image Number /Result Vector	
0	1 1 0 0 0 0 0 1	48	1 1 0 0 0 1 0 1	96	1 1 0 0 0 1 0 1	144	1 1 0 0 0 0 0 0
1	0 1 0 1 1 1 0 0	49	1 1 0 1 1 1 0 1	97	0 0 0 0 0 0 0 0	145	0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0	50	1 1 0 0 1 1 1 1	98	1 0 1 0 0 1 1 0	146	1 1 1 0 0 0 0 1
3	0 0 0 0 0 0 0 0	51	1 1 0 1 1 0 0 1	99	0 0 0 0 0 0 0 0	147	0 0 0 0 0 0 0 0
4	0 0 0 0 0 0 0 0	52	0 0 0 0 1 1 1 1	100	1 1 0 0 0 0 0 1	148	0 0 1 0 0 0 0 0
5	0 0 0 0 0 0 0 0	53	1 0 1 0 0 1 1 0	101	0 0 0 0 1 0 0 0	149	1 0 0 0 0 0 0 0
6	1 1 0 1 0 1 0 1	54	0 1 0 1 1 1 1 1	102	1 1 0 0 0 0 0 1	150	0 0 0 0 0 0 0 0
7	0 0 1 1 0 1 1 1	55	0 0 0 0 0 0 0 0	103	0 0 0 0 0 0 0 0	151	1 1 1 1 1 1 1 0
8	0 0 0 0 0 0 0 0	56	0 0 0 0 0 0 0 0	104	0 0 0 0 0 0 0 0	152	0 0 0 1 1 1 1 1
9	0 0 0 0 0 0 0 0	57	0 0 0 0 0 0 0 0	105	0 0 0 0 0 0 0 0	153	0 0 0 0 0 0 0 0
10	0 0 0 0 0 0 0 0	58	0 0 0 0 0 0 0 0	106	0 0 0 0 0 0 0 0	154	0 0 0 0 0 0 0 0
11	0 0 0 0 0 0 0 0	59	0 0 1 0 0 0 0 0	107	0 0 0 0 0 0 0 0	155	0 0 0 0 0 0 0 0
12	0 0 0 0 0 0 0 0	60	0 0 1 0 0 0 0 0	108	0 0 0 0 0 0 0 0	156	0 0 0 0 0 0 0 0
13	0 0 0 0 0 0 0 0	61	0 0 0 0 0 0 0 0	109	0 0 0 0 0 0 0 0	157	0 0 0 0 0 0 0 0
14	0 0 0 0 0 0 0 0	62	0 0 0 0 0 0 0 0	110	0 0 0 0 0 0 0 0	158	0 0 0 0 0 0 0 0
15	0 0 1 1 1 1 1 1	63	0 0 0 0 0 0 0 0	111	0 0 0 0 0 0 0 0	159	0 0 0 0 0 0 0 0
16	1 1 0 1 1 1 0 1	64	1 1 1 0 0 0 0 1	112	1 1 1 0 1 1 1 1	160	1 1 0 0 0 0 0 0
17	1 1 0 0 1 1 0 1	65	1 1 1 1 1 1 1 1	113	0 0 0 0 0 0 0 0	161	0 1 0 0 0 1 0 1
18	0 1 0 0 0 1 0 0	66	1 1 0 0 0 0 0 1	114	1 1 0 0 0 0 0 1	162	1 1 1 1 1 1 1 1
19	0 0 0 0 1 0 0 0	67	0 0 0 0 0 0 0 0	115	0 0 0 0 0 0 0 0	163	0 0 0 0 0 0 0 0
20	0 0 0 1 1 1 0 0	68	1 1 0 0 0 0 0 1	116	1 0 1 1 1 1 0 0	164	1 1 0 0 0 0 0 0
21	0 0 0 0 0 0 0 1	69	1 0 0 0 1 1 0 1	117	1 1 0 1 1 0 1 1	165	1 1 1 0 0 0 1 0
22	1 0 0 0 0 0 0 1	70	1 1 1 0 0 1 0 1	118	0 0 0 0 1 0 0 0	166	0 0 0 0 0 0 0 0
23	0 0 0 0 0 0 0 0	71	0 0 0 0 0 0 0 0	119	0 0 0 0 0 0 0 0	167	0 0 0 0 0 0 0 0
24	0 0 0 0 0 0 0 0	72	0 0 0 0 0 0 0 0	120	0 0 0 0 0 0 0 0	168	0 0 0 0 0 0 0 0
25	0 0 0 0 0 0 0 0	73	0 0 0 0 0 0 0 0	121	0 0 0 0 1 0 0 0	169	1 1 0 1 1 1 1 0
26	0 0 0 0 0 0 0 0	74	0 0 0 0 0 0 0 0	122	0 0 0 0 0 0 0 0	170	1 0 0 1 1 0 0 0
27	0 0 0 0 0 0 0 0	75	0 0 0 0 0 0 0 0	123	0 0 0 0 1 0 0 0	171	0 0 0 0 0 0 0 0
28	0 0 0 0 0 0 0 0	76	0 0 0 0 0 0 0 0	124	0 0 0 0 0 1 0 0	172	0 0 0 0 0 0 0 0

29	0 0 0 0 0 0 0 0	77	0 0 0 0 0 0 0 0	125	0 0 1 0 0 0 0 0	173	0 0 0 0 0 0 0 0
30	1 1 0 1 1 0 1 1	78	0 0 0 0 0 0 0 0	126	0 0 0 0 0 0 1 0	174	0 0 0 0 0 0 0 0
31	0 0 0 1 0 1 0 0	79	0 0 0 0 0 0 0 0	127	0 0 0 0 0 0 0 0	175	0 0 0 0 0 0 0 0
32	1 1 0 0 1 1 0 1	80	1 1 1 0 0 0 0 1	128	1 1 0 1 1 0 1 1	176	1 1 1 0 0 0 0 1
33	1 0 0 1 1 1 1 1	81	1 1 0 0 0 0 0 0	129	0 0 0 0 0 0 0 0	177	0 0 0 0 0 0 0 1
34	1 0 0 0 0 0 0 1	82	1 1 0 0 0 1 0 1	130	1 1 1 0 0 0 0 1	178	1 1 0 0 0 0 0 0
35	0 0 0 1 0 1 1 0	83	0 0 0 0 0 0 0 0	131	0 0 0 0 0 0 0 0	179	0 0 0 0 1 1 0 0
36	1 0 0 1 1 1 1 1	84	1 1 1 1 1 1 1 1	132	1 1 1 0 0 0 0 1	180	0 0 1 0 0 1 1 0
37	1 1 0 0 0 0 0 0	85	1 1 1 0 0 0 0 0	133	1 0 0 1 1 1 1 0	181	1 0 0 0 0 0 0 0
38	1 1 1 0 0 0 0 1	86	1 1 1 0 0 1 0 1	134	1 1 1 1 1 0 0 1	182	0 0 0 0 0 0 0 0
39	0 0 0 0 0 0 0 0	87	0 0 0 0 0 0 0 0	135	0 0 0 0 0 0 0 0	183	0 0 0 0 0 0 0 0
40	0 0 0 0 0 0 0 0	88	0 0 0 0 0 0 0 0	136	0 0 0 0 0 0 0 0	184	0 0 0 0 0 0 0 0
41	0 0 0 0 0 0 0 0	89	0 0 0 0 0 0 0 0	137	0 0 0 0 0 0 0 0	185	0 0 0 0 1 0 0 0
42	0 0 0 0 0 0 0 0	90	0 0 0 0 0 0 0 0	138	0 0 0 0 0 0 0 0	186	1 0 1 1 1 0 0 0
43	0 0 0 0 0 0 0 0	91	0 0 0 0 0 0 0 0	139	0 0 0 0 0 0 0 0	187	1 1 1 1 1 1 0 0
44	0 0 0 0 0 0 0 0	92	0 0 0 0 0 0 0 0	140	0 0 0 0 0 0 0 0	188	0 0 1 0 1 1 0 0
45	0 1 0 0 0 0 1 0	93	0 0 0 0 0 0 0 0	141	0 0 0 0 0 0 0 0	189	0 0 0 0 0 0 0 0
46	0 0 0 0 0 0 0 0	94	0 0 0 0 0 0 0 0	142	0 0 0 0 0 0 0 0	190	0 0 0 0 0 0 0 0
47	0 0 0 0 0 0 0 0	95	0 0 0 0 0 0 0 0	143	0 0 0 0 0 0 0 0	191	0 0 0 0 0 0 0 0

IV. Conclusion

BPNs were trained to recognise different categories of sub-lines from sub-images. Results are mostly good. A lot of the poor results are due to confusion between similar line types, for example, between vertical-slash and slash lines, both of which would be approximately correct.

Results presented are not as specific as, say, using the analytical approaches presented in [2] and [3] which specify the pixels lines found come from, and their angles to, say, the vertical, correct to the nearest whole degree. Results from the current paper specify which sub-image a sub-line comes, a sub-image being 8 x 8 pixels in size, and which category it belongs to, a category spanning about 10 degrees. This is okay for the purpose of further processing for robot navigation such as in [10] which relies on the position of lines found relative to a reference point such as a vanishing point, but does not need to know exactly how far the line is from the reference point, and also combines lines of similar angles into the same categories used in this paper (presented in table 1). The detection of the vanishing point used in [10] was presented in [11] and relies on more accurate estimates for positions and orientation of lines than the method presented would provide. Using the methods in the combination of [2] and [3] would yield such accuracies. Future work would include looking into estimation of the vanishing point without needed the go the route of time-consuming analytical methods such as the combination of [2] and [3], if the idea of avoiding using them for line detection is to make sense.

Results have been used to try out other neural networks, and to apply other techniques to do further processing. One approach which uses the Hough transform to stitch together sub-lines found from sub-images as presented in this paper, along the lines of their categories, to get the full line they came from has been presented in [12]. Other further processing will be considered in the future.

References

[1]. G. L. Dempsey and E. S. McVey, “A Hough Transform System based on Neural Networks”, IEEE Transaction on Industrial Electronics, 39(6), pp. 522-528, 1992

[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. 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

[5]. G. K. Damaryam, A Back-Propagation Neural Network Detection of Vertical Sub-Lines, International Journal of Engineering and Science, 5(1), pp 22-29, 2016.

[6]. 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

[7]. J. Rogers, Object-Oriented Neural Networks in C++, (San Diego: Academic Press, 1996)

[8]. T. Chang, and J. Y. Hsu, “Neural networks for robot navigation control”, Proceedings of the International Symposium on Artificial Neural Networks, pp. 701-707, 1994.

[9]. R. M. Inigo and R. E. Torres, “Mobile robot navigation with vision based neural networks”, Mobile Robots IX. 2353, pp. 68-79, 1995

[10]. G. K. Damaryam and O. J. Abari, “Corridor and Doors Recognition from Detected Lines and Estimated Vanishing Point”, International Journal of Science and Research, 5(3), pp.204-211, 2016.

[11]. G. K. Damaryam, “One Point Perspective Vanishing Point Estimation for Mobile Robot Vision Based Navigation System”, International Journal of Research and Science, 5(2), pp 930-934, 2016.

[12]. G. K. Damaryam and A. S. Usman, “Hough Transform Consolidation of Back Propagation Network Line Estimates for Self-Navigation for a Mobile Robot” In Press: International Journal of Engineering and Science, 5(4), 2016.