An Investigation into Gesture Recognition in BingBee using Neural Nets in MATLAB®

Submitted in partial fulfilment of the requirements of the Degree Bachelor of Science (Honours) of Rhodes University

Primary Investigator: Ray Musvibe

Supervised by: Professor Peter Wentworth

RHODES UNIVERSITY
Where leaders learn

2nd November, 2008

Abstract

A lot of work has been done towards developing intelligent and natural interfaces between human beings and computer systems. Gesture recognition is a fast developing area in the field of Human Computer Interaction. Present day system developers not only aim to develop fully functional systems, but also aim for systems with more intuitive interfaces, allowing for easier user/system interaction. Gestures have provided interface designers with a good alternative to traditional users interface design approaches like the Command Line Interface (CLI) and the Graphical User Interface. In this regard, for interface designers to develop highly intuitive gesture based interfaces efficient and accurate gesture recognition approaches have to be employed and the most suitable implementation packages selected if need be.

Artificial Neural Networks are a rapidly growing area in the field of Artificial Intelligence. Whether neural nets are appropriate to the specific gesture recognition problem in BingBee shall also be discussed here. Initial conclusions are that neural nets will provide a robust gesture recognisor for BingBee. In addition, the reasons as to why MATLAB® is a suitable platform for prototyping this proposed neural net solution will also be discussed in this work. Again initial conclusions are that MATLAB® is a powerful tool for developing and testing a highly optimized neural network solution for gesture recognition in BingBee.

Acknowledgments

My greatest thanks goes to my Lord and Saviour, Jesus Christ, the bright morning star and the very author of life, without whom there would be no life to the world as we know it. I thank you Lord for the inspiration and the encouragement that you have given me over the past few months. I also thank you for always having a listening ear whenever I ran into trouble throughout this year, wether in my project or coursework. This work was only possible through your gracious and all conquering love.

Secondly, I thank my ever present, patient, hardworking and encouraging supervisor, Professor Peter Wentworth, for all the work that he put into this project. He was not only a great supervisor, but in many ways a great teacher, for through him, I learned so much. I hope this work brings much credit to you as supervisor; this work would not have been possible without your wisdom and guidance.

I also extend my heart-felt thanks to Professor Mike Burton and Dr Denis Riordan for their effort into the design and implementation of my gesture recognisor. This work would not have progressed without their influence.

To Brenda, dear, thank you for being there for me as you have been for so long now, though far, you are always by my side. May God almighty continue to bless you with much wisdom and love.

Also, to great friends Taka, Shange, Curt, Martin, Bwini and Snax, it just wouldn't have been the same without you. Going through the year with you was lots of fun.

To my family that supported me throughout my first year at "Rhodes", guys you have been great, you make all this worth while. May there be showers of blessings to you!

Lastly, but not least, I gratefully thank the financial support of Telkom SA, Bright Ideas Projects 39, Business Connexion, Verso Technologies, Comverse SA, Tellabs, Open Voice, Mars Technologies, Tellabs, Amatole, THRIP and Stortech, through the Centre of Excellence. I would also like to thank the trustees of the Andrew Mellon Foundation for granting me this opportunity to study at such a distinguished and highly respected institution as this.

Contents **Page**

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Problem statement

BingBee is a public information kiosk designed to stimulate children's education through entertainment. It can be set up in any secure building that has a window (for visual interaction with users). **Figure 1** shows an example of children playing with BingBee

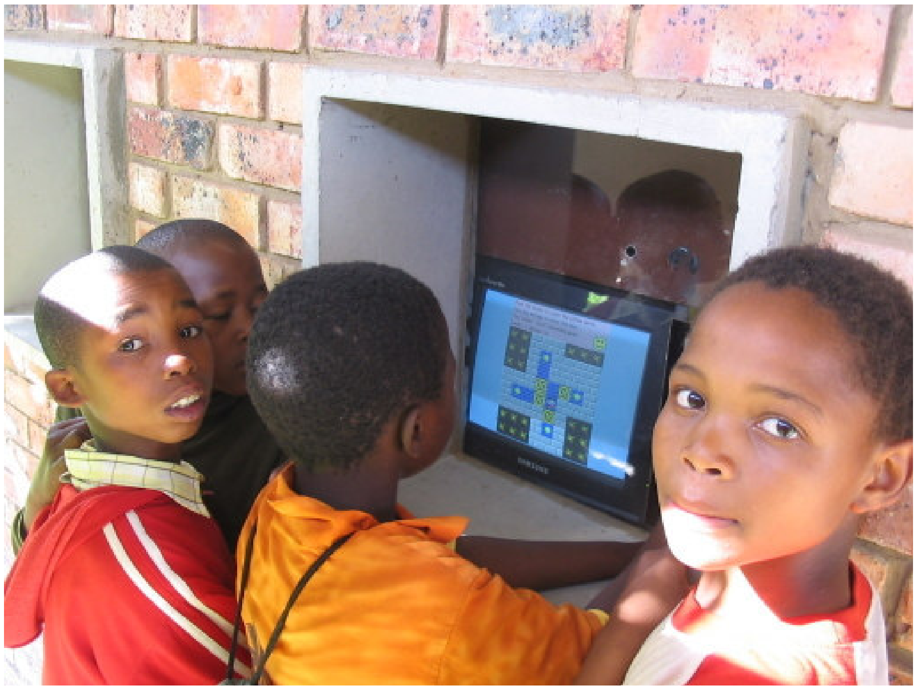


Figure 1 Children playing at the Kiosk

BingBee presently uses a pad on the window with key and mouse regions as a user interface. A web cam picks up the input. Image analysis is performed on the images captured to identify the input. Input is presently limited to these keys and mouse pad movements.

In order to improve on user interaction and control, we are proposing a set of ten primitive (2D) gestures that can be input through the touch pad, specifically targeting control of a 3D scene fly-though scenario.

A study by IBM has shown that an accuracy rating of below 97% in character and gesture recognition software such as Graphiti in PDA's and Palm Tops is generally viewed as inadequate by consumers. Most users have to change their writing styles for proper recognition in most systems. [37] We however anticipate that reaching this market standard may be difficult to achieve in BingBee due to observed noise levels and Pad sensitivity problems. The aim in terms of classification and accuracy, will thus be to come as close as possible to this market standard, thereby ensuring that user expectations are met and building a truly intuitive gesture interface.

1.2 The Ten Proposed gestures

The following table below shows the target gestures for implementation in BingBee. Please note that the arrow-heads are not part of the gesture – they simply exist to show that the line in the first case for instance, is drawn left-to-right, and the first circular gestures is in the clockwise direction.

→	↗	↓	↘	←
↑	↖	↙	↻	↺

Table 1 Showing the ten proposed gestures

1.3 Background

Gesture recognition is currently an area of intense study as developers attempt to make present day computers more human, and more intuitive.

This project presents some of the literature available on neural computing and gesture recognition in general. It examines previous work in related studies and will explore the various approaches available for gesture recognition. It will form the foundation for building an intuitive 2D gesture recognisor for BingBee.

I will conduct a survey on the many neural network architectures suitable for gesture recognition, particularly targeting architectures with low computational overhead and good gesture classification performance. These would be ideal for our target implementation in BingBee.

I will also conduct a brief survey into MATLAB® and a few other software packages available for developing neural nets in order to select the most suitable tool for developing our solution.

1.4 Motivation

It is expected of BingBee, as with most modern day or future computer system, to provide user friendly interfaces. Many games of the past decades had simple interface designs and could be manipulated quite easily with a keyboard or Joystick. A good example is of the 2D game Pac-Man that even today remains popular among gamers. Four arrow keys were adequate for complete control of Pac-Man. [38]

As computers have evolved over the past decades, so has computer gaming. Developments such as the continued introduction of more powerful CPU's and GPU's, have led to the creation of a multi-million dollar gaming industry were innovation is a critical survival trait. [39]

A more recent innovation on the gaming platform over the past few months have been the Nintendo Wii®, which in many respects has revolutionised Human Computer Interaction and the future of gaming interfaces through the Wii remote. [40] The Wii remote comes fitted with an accelerometer and optical sensors that allow user to manipulate objects on the screen.

Fig 2 The Nintendo Wii® remote

Another successful development in the gaming arena over the past few years was Sony's EyeToy®. The EyeToy uses a web cam and a microphone to allow user's to interact with it. [41]

Fig 3 A user interacting with a game using the Sony EyeToy's webcam

It is with such developments in mind that we seek to improve upon the interface design of BingBee. Due to the present interface design, consisting of a touch surface and key area, gesture recognition was a natural option. A gesture recognisor could easily be added in software without the need for specific hardware modifications.
With the current Xnasig group at Rhodes working on the implementation of 3D games in BingBee, the gesture interface should be welcome development.

1.5 Research goals

In order to achieve the overarching goal of developing an intuitive interface for BingBee, this thesis has the following research goals:

1. Conduct investigative studies into the most suitable approach for gesture recognition in BingBee.
2. Conduct investigative studies into the most suitable tool set for developing our solution for our target implementation in BingBee.

3. Develop an optimised solution for implementation in BingBee and integrate this solution into the interface design of BingBee.

1.6 Overview

This thesis begins by examining the BingBee interface design as well as looking into the various approaches to the gesture recognition problem in Chapter 2. Here I will also proceed to deliberate on the selected implementation method and architecture, namely a neural net solution. In chapter 3, I will examine available software packages for developing and simulation of neural nets. In Chapter 4, I will walk you through my neural net development process in MATLAB® and in Chapter 5 go over the C# implementation phase of the project. I provide my conclusions in Chapter 6 and also provide some insight into possible extensions.

Chapter 2: Related Work

2.1 BingBee [3]

BingBee presently uses a touch pad on the kiosk window with key and mouse regions as a user interface. A web cam picks up the input as distortions in the fabric and some image processing allows the software to map these onto corresponding keys or pad position information. [10]
The image below shows a user entering data on the pad;

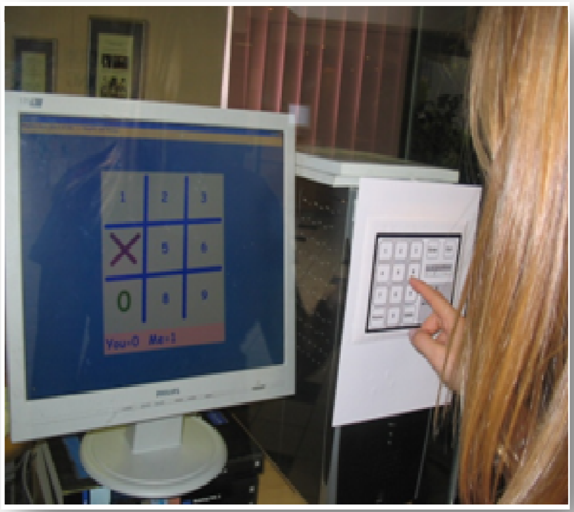


Fig 4 above shows a user entering data onto the pad. **Figure 5** below shows a single frame capturing user input as seen by BingBee's web cam.

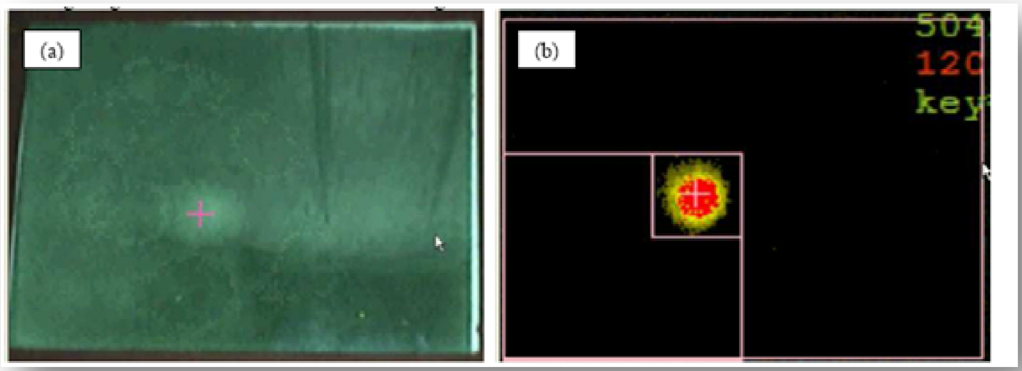


Fig 5 User input as captured by BingBee's web cam

Positioning information is extracted from web-cam frames, captured at slow-rate samples of between 5-25 frames per second. The position of the input is then determined using an image differencing technique that subtracts each image from a long-term historical average image. Thus we expect a typical Gesture would consist of a sequence of roughly between eight and sixteen (x, y) coordinate pairs. [10]

2.2 Gesture Recognition and Advances in Human Computer Interaction

A primary goal of Gesture Recognition research over the years has been to create systems which can identify specific human gestures and use them to convey information or for device control [6].

Gesture Recognition is also important for developing alternative human-computer interaction modalities [1]. It enables humans to interface with machines in a more natural way.

Cadoz [5] describes three distinct roles for gestures

- Semiotic- the communication of meaningful information
- Ergotic- for manipulating the environment.
- Epistemic- for discovering the environment through tactile experience

In our case of the BingBee Recogniser, only the Ergotic roles are relevant. Users only need to manipulate objects (Ergotic) through gestures.

Gesture Recognition can thus be defined as the process by which gestures made by the users are made known to the target system [17].

Gesture Recognition has been applied to augmented reality, sign language Recognition and Human Robot Interaction among others applications [20].

Microsoft recently unveiled its multi-touch functionality that is built into Windows 7.

Fig 6 Microsoft multi-touch functionality with Gesture Recognition

Gesture Recognition has even gone mobile, with some manufactures employing Gesture Recognition on their cellphones.

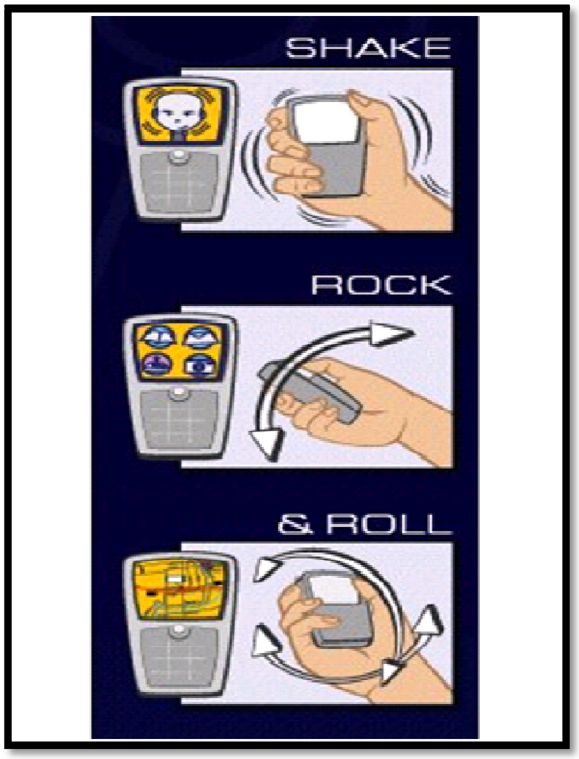


Fig 7 above shows the use of hand gestures in mobile device control

According to techeblog [29], Gesture Recognition is set to become the next major technology.

As more and more interface designers use gestures to complement traditional UI designs like GUI, Gesture Recognition seems the way to go in developing highly interactive, user friendly interfaces.

2.3 Online Recognition vs. Offline Recognition

In on-line recognition, characters are recognized as they are drawn whereas in off-line recognition, characters are first drawn 'on paper' and then optically scanned and represented as two-dimensional rasters [16].

Offline recognition has the major disadvantage of being unable to differentiate gestures that are only distinguished by the direction in which they are drawn, say an anticlockwise circle and a clockwise circle. . These would be classified as the same character or gesture, irrespective of which recognition method is used. In **Table 2** below, A and B would be classified as the same gesture using off-line recognition but as different gestures using on-line Gesture Recognition.

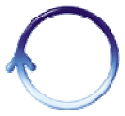
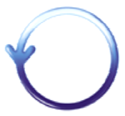
Gesture A	Gesture B
	

Table 2 Showing a clockwise circle gesture and an anticlockwise gesture

This significantly limits our set of input gestures. In this regard, online recognition becomes our preferred choice.

Guyon [9] used on-line character recognition to achieve very good results for the touch terminal that he built for character recognition. Guyon used a multi-layered feed-forward backpropagation network to identify the characters that he was working on. On-line character recognition uses the ordering or the sequence of points of characters for recognition. It can comfortably be adapted for Gesture Recognition in BingBee.

2.4 Approaches to Gesture Recognition

There are three common approaches to Gesture Recognition. These are Neural Nets, Hidden Markov Models and Dynamic Time Warping. I discuss these here.

2.4.1 Neural Computing

2.4.1.1Machine Learning

Neural networks together with Genetic Algorithms form a broad subfield of Artificial Intelligence called *machine learning*. Machine learning consists merely of techniques and algorithms that allow computers to 'learn'. [32].

Machine learning systems in part attempt to eliminate the need for human intuition in the analysis of data which can come in a variety of forms. Data can take the form of sound files, computer vision, financial data like stock market figures, DNA sequences, bitmap images, cheminformatics data and many other data forms.

Machine learning has thus been mainly applied to pattern classification as well as for prediction or forecasting based on data sets.

2.4.1.2 Background on Neural Nets

Neural nets represent an approach to Artificial Intelligence that attempts to model the human brain. Neurons are processing units that operate in parallel inside the human brain. There are an estimated 10 billion neurons in the human brain with about 60 trillion connections between these neurons. Each neuron receives inputs from other neurons in the form of tiny electrical signals and, likewise, it also outputs electrical signals to other neurons. These outputs are weighted in the sense that the neuron does not 'fire' any output unless a certain threshold/bias is reached. These weights can be altered through learning experiences; this is how the brain learns. The brain is therefore a network of neurons acting in parallel – a Neural Network. [4]

The diagram below shows a typical brain cell;

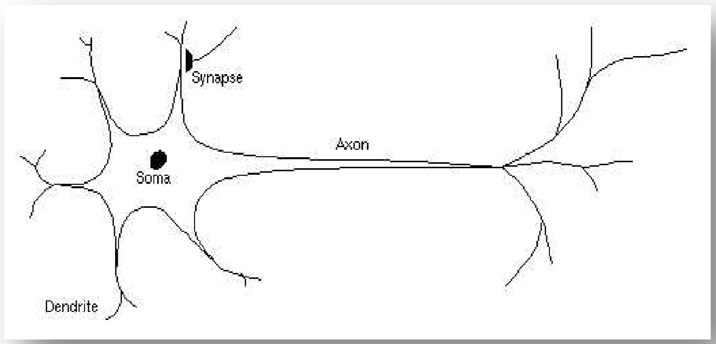


Fig 8 Showing a typical human brain cell

Similarly, an Artificial Neural Nets consists of artificial neurons, which are mathematical models of biological neurons. Instead of receiving electrical impulses like the biological neuron, an artificial neuron (called a perceptron), receives numerical values and also outputs a numerical value. [4]

The diagram below shows a representation of an artificial neuron.

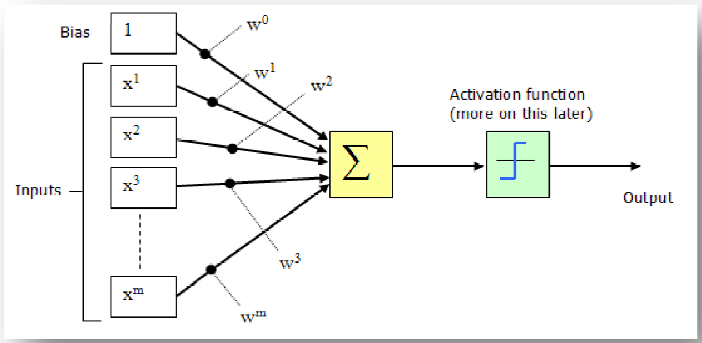


Fig 9 Showing an artificial neuron

The input into the perceptron consists of the numerical value multiplied by a weight plus a bias. The perceptron only fires an output when the total strength of the input signals exceeds a certain threshold. As in biological neural Networks, this output is fed to other perceptrons. [26].

The weighted input to a perceptron is acted upon by a function (the transfer function) and this will determine the activation or output. Common transfer functions used in Artificial Neural networks include the Hard Limiter, Log-Sigmoid and the Sign function. [8]

Fig 10 below shows a representation of a simple (feed-forward) Neural Network with four inputs, one hidden layer and four outputs. Neural networks learn by changing their weights. [4]

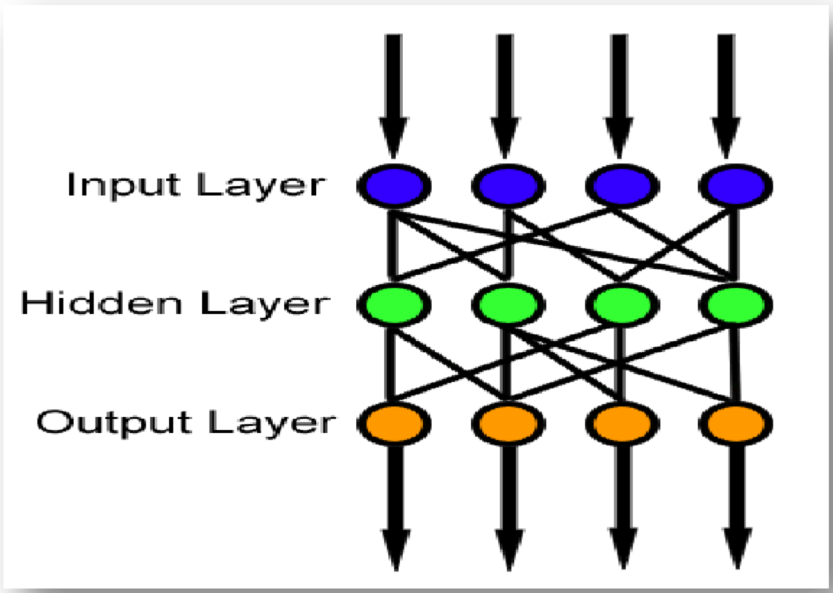


Fig 10 A simple feed-forward Neural Net

2.4.1.3 Types of Artificial Neural networks and their Applications

There are several types of Artificial Neural Networks, the different strengths and applications of these depend on their structure, dynamics and learning methods used. [14]

Neural networks have been particularly noted to be good at identifying patterns or trends in data, even where the smartest humans fail to identify any trends [4]. Artificial Neural networks have offered better performance compared to traditional methods in areas such as Virtual Reality, Optimisation, Pattern Detection, Data Mining and Signal Filtering.

2.4.1.3.1 Feed-forward Networks

These were the first types of Neural Networks to be devised. Information moves in one direction, from the input layer to the output layer and there are no cycles. A common learning technique for feed-forward networks is backpropagation; discussed earlier. A common application of feed-forward Neural networks is pattern recognition or classification. [26]

2.4.1.3.2 Recurrent Networks

These are fundamentally different to feed-forward networks in that movement through the net is bidirectional, not unidirectional as in feed-forward networks. Several recurrent architectures exist, such as the Elman Network and the Hopfield Network [42]. These can be trained using Backpropagation through Time (BPTT), Real-time Recurrent Training and by Genetic Algorithms. Recurrent Networks have been used in the problem of language acquisition [33], music composition and Time series prediction. These, however, are processor intensive.

2.4.1.3.3 Self-Organising maps

This is another type of Neural Network that uses a form of learning called Unsupervised Learning. Self-Organising maps produce a 'map' that seeks to preserve the topological properties of the input space. Applications of these include voice and handwriting recognition. [28]

2.4.1.3.4 Other Neural Network Architectures

Several other Neural Network Architectures exist. Examples of these include: [42]

- Spiking Neural Networks
- Time Delay Neural Networks
- Stochastic Neural Networks
- Modular Neural networks
- Cascading Neural Networks

General feed-forward nets are by far a natural choice for our Gesture Recognition case with BingBee. There is a lot of technical support on feed-forward nets as they have been around the longest period of time and they are well documented. They also have the added advantages of simplicity and low computational overhead, particularly during training.

2.4.1.4 Learning in Neural Nets

Neural Network Architectures generally fall into two broad categories; Supervised and Unsupervised Learning.

2.4.1.4 .1 Supervised Learning

The term ‘supervised' comes from the fact that the desired signals on individual output nodes are provided as part of the training. The goal of the machine in this case is to learn from the training set, so that it becomes able to produce the correct output given new inputs. [32]

It is usually performed with feed-forward nets where training patterns are composed of input and output vectors. A training cycle consists of the following steps.

a. An input vector is presented at the inputs together with a set of desired responses (targets), one for each node at the output layer.
b. A forward pass is done and the errors between the targets and actual responses for each node in the output layer are found. These are then used to adaptively make changes in the net depending on the prevailing learning rule. [18]

2.4.1.4 .1.1 The Backpropagation algorithm [8]

This is an algorithm used to train neural networks under supervised learning. It is mostly useful in feed-forward neural networks and requires that the desired output be known and transfer function be differentiable. Examples of suitable transfer functions include the sigmoid function, tanh and the log sigmoid function. To train a neural network using backpropagation, a training input pattern is propagated through the neural net from layer to layer until the output pattern is generated by the output layer. If the output pattern obtained is different from the desired output, an error is calculated and appropriate weight changes are made and propagated backwards through the entire network. The error is calculated by the equation.

$$\boldsymbol{e_k(p) = y_{d,k}(p) - y_k(p)}$$

Where $\boldsymbol{e_k(p)}$ is the error generated after P iterations on neuron k, $\boldsymbol{y_{d,k}(p)}$ is the desired output after P iterations on neuron k and $\boldsymbol{y_k(p)}$ is the actual output from neuron 'k' after P iterations.

The error is then used to generate the error gradient $(\boldsymbol{\partial_k(p)})$**.** The error gradient is a product of the error and the differential of the transfer function propagated backwards. In the case of the sigmoid transfer function, the error gradient for the output layer would be given by:

$$\boldsymbol{\partial_k(p) = y_k(p).[1 - y_k(p)].e_k(p)}$$

The weights are modified as the error is propagated back. The weights are modified by the equation:

$$\boldsymbol{\Delta w_{jk}(p) = \alpha.y_j(p).\partial_k(p)}$$

Where $\boldsymbol{\Delta w_{jk}(p)}$ is the weight change for the weight input to the 'k^{th}' neuron, inputs from the 'j^{th}' neuron of the previous layer (neuron j's input feeding into neuron k). Where $\boldsymbol{\alpha}$ is the *learning rate,* $\boldsymbol{y_j(p)}$ is the output from neuron 'j'. $\boldsymbol{\partial_k(p)}$ is the error gradient as calculated above.

The new weight for the neural net are thus calculated as:

$$\boldsymbol{w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)}$$

Where$\boldsymbol{w_{jk}(p+1)}$is the new weight for the 'k^{th}' neuron after P iterations and $\boldsymbol{w_{jk}(p)}$ is the previous weight while $\boldsymbol{\Delta w_{jk}(p)}$ is the weight change as calculated above. [8]

2.4.1.4 .2 Unsupervised Learning

In Unsupervised Learning, the machine simply receives inputs (say x1, x2…) but obtains no supervised target outputs. Unsupervised Learning has been used in finding patterns in the data that would otherwise be considered pure unstructured noise. Applications of Unsupervised Learning mainly centre on estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.[23]

Since we are not specifically looking for trends in data in our Gesture Recogniser, Unsupervised Learning automatically becomes irrelevant and Supervised Learning becomes a natural choice. We need to train the neural net with predetermined patterns and expected outputs, so that it can later recognize similar gestures.

2.4.1.5 Artificial Neural networks and Design Tradeoffs

The general guideline when it comes to developing neural nets is to develop a neural net with as few neurons as possible [4]. A small number of neurons have the advantages of better noise tolerance, low computational overhead and low complexity among others. Neural nets with large number of neurons tend to be more accurate but suffer from overfitting or overtraining, which means the net has learned to classify training data too well; hence it would find exact or very close matches, but may struggle with noisy data. The table below compares and contrasts the tradeoffs when moving from a low-count neural net to a net with many more neurons. [23]

Property	**General, Robust, Small size nets**	**Accurate, Brittle, Larger sized nets**
Complexity Resolution Level of accuracy	Low	High
Definition of the problem	Usually bad, but may be OK	Must be well-posed
Data coding	Dimensionality reduction	Many dimensions
Number of network units	Low	High
Data collection	Less data needed Sparse data Even distribution Noisy data tolerated	More data needed Dense data Uneven distribution No noise tolerated
Test criteria	Generalizes well to unseen data	Reaches required level of accuracy
Main problem in training	Inaccuracy	Overfitting

Table 3 Comparing design trade-offs in neural net architectures.

The preferred trend among neural net developers is to go for the more robust architecture. In our case a more robust net would also be a natural choice because we would expect the nature of the input in most cases to have a great deal of noise and variability, without generalisation our recogniser would surely struggle.

2.4.1.6 Optimising Neural Net Architectures

Optimising the training and recognition capability of a neural net has been achieved in two ways. [4]

1. Using Genetic Algorithms
2. Using a trial and error approach.

Genetic Algorithms are a natural choice if we are trying to optimize the performance of a neural net. The major problem in using this approach is correctly encoding the Neural Net into the Genetic Algorithm [22]. A simpler yet effective method is to conduct performance trials on the varied neural net architectures based on some performance metric, say noise tolerance, and then deduce conclusions from these. I would prefer conducting performance trials because of the simplicity of the approach, but most importantly, it has also been noted that Genetic Algorithms are not always correct. [4]

2.4.1.7 Advantages of Neural Computing

- Neural computing does not need to assume that an underlying data distribution exists as is usually the case in statistical modelling.
- Neural networks are applicable to multivariate non-linear problems. In our BingBee Recogniser we have two variables (x, y) .[2]
- Neural nets possess learning capabilities, [11]
- Fast computational ability[7]

2.4.1.8 Disadvantages of Neural Computing

- Minimizing overfitting in neural nets requires a great deal of computational effort.

- Individual relationships between input variables and the output variables are not developed by engineering judgment, meaning that the Neural Network model tends to be a black box or input/output table without analytical basis.
- The sample size has to be large.
- Neural networks are prone to overfitting [15]

2.4.2 Using Hidden Markov Models for Gesture Recognition

A Hidden Markov Model (HMM) is a statistical model in which the system being modelled is assumed to be a *Markov process* with unknown parameters. The challenge then is to determine the hidden parameters from the observable parameters. The extracted model parameters can then be used to perform further analysis, with particular applications to pattern recognition applications.

Hidden Markov Models are particularly known for their applications in temporal pattern recognition problems such as speech, handwriting, gesture recognition and musical score following.[13]

The HMM-based Gesture Recognition approach can be summarised as follows:

1. *Define the Gestures* – The Gestures must first of all be specified.
2. *Describe each Gesture in terms of an HMM* - A multi-dimensional HMM is employed to model each Gesture. A Gesture is described by a set of *N* distinct hidden states and *r* dimensional *M* distinct observable symbols.
3. *Collect training data* - With HMM-based approach, Gestures are specified through the training data. Input data is *preprocessed* before being used to train the HMM.
4. *Train the HMM's through training data*
5. *Evaluate Gestures with the trained model* - The trained model can be used to classify the incoming Gestures. [16]

The diagram below shows the gesture classification process using HMM.

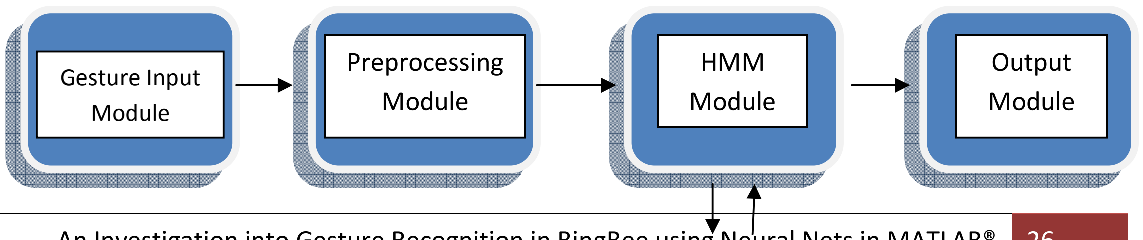


Fig 11 Gesture classification using HMM's

2.4.2.1 Advantages of HMM in Gesture Recognition

- HMM have efficient algorithms for learning and recognition.[12]

2.4.2.2 Disadvantages of HMM in Gesture Recognition

- many parameters need to be set
- Large amount of training examples are required
- There is difficulty for extension to large vocabularies [30]
- The number of states of the (threshold) model increases as the number of gesture models grows, which tends to cause a waste of running time. [12]

2.4.3 Dynamic Time Warping (DTW) and Gesture Recognition

The DTW is a template-based matching technique that can be applied to problems with temporal variabilities.

Although it has been successful in small vocabulary problems, the DTW often needs more templates for a range of variations. As DTW calculates variability information during the matching process, it needs more templates for representing spatial variabilities. Furthermore, it has no consideration for representing undefined patterns**.** [12]

It has mainly been used for off-line Gesture Recognition, not on-line Gesture Recognition, which makes it unsuitable for our recogniser in BingBee. [43] It would fail to distinguish similar gestures like the circular clockwise and anticlockwise gestures.

It is quite clear that the leading approaches to on-line Gesture Recognition are Hidden Markov Models and Neural Nets. Both methods have been vastly used in gesture classification and several hybrid architectures of these have been proposed to take advantage of their individual strengths.

They both suffer from a need for large training sets and thus tend to be processor intensive. Neural nets, however are processor intensive only during training, unlike HMM's which remain processor intensive even during the actual classification. This allows for computational speed-up in neural nets. There is always the remote chance of over fitting with neural nets but we can work around this through prudent network architecture selection. The ability of neural nets to generalise makes them a natural for our case in BingBee.

Even though neural nets are a 'black box' approach, they hide the underlying details and parameters, unlike HMM's that involve setting many parameters, and they thus offer a much simplified approach.

2.5 Proposed Neural Network Gesture Classifier

There are a set of ten proposed gestures. Users input the gestures using the touchpad. The web-cam captures the input at slow-rate samples of between 15-25 frames per second, and using an image differencing technique, the sequence of (x, y) coordinates representing the gesture is determined.

2.5.1 Preprocessing

This raw set of (x, y) coordinates will have to be preprocessed before it can be fed into the trained neural net for classification. One of the major limitations of neural nets is that they require a fixed number of inputs. Preprocessing must ensure that this condition is met. This means that a gesture with an inadequate number of inputs must not be passed onto the neural classifier or it must be 'enlarged' in an appropriate manner to meet the size requirement. A gesture that is too long must be sampled appropriately to fit the exact number of inputs in the neural classifier.

The resultant processed input can now be passed into the classifier. Yet further preprocessing can be performed. Preprocessing can also be used to extract further ‘meaning’ from the raw data and then passing the interpreted data onto the neural classifier. This has the general effect of improving the performance of neural nets [36]. In my approach, the *n* input sequence of (x, y) coordinates is preprocessed into an *n*-1 vector sequence, which is then passed into the trained neural net for classification. And yes, the general effect of this is improved gesture recognition performance as compared to using raw (x, y) coordinates. Scaling can also be introduced to improve performance. The table below outlines the *vectorisation* of points representing a gesture.

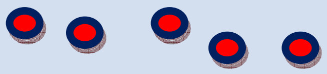
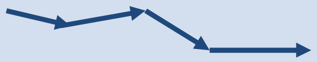
Gesture	***n* Corresponding (x,y) input coordinates**	***n-1* Vector form (each vector of form (δ(*x*), δ(y))**
		

Table 4 showing the *vectorisation* process

2.5.2 Classification

The set of input is passed through the trained Neural Network which classifies the gesture into one of several predefined classes that can be identified by the system.

Fig 12 below represents the classification process for system control

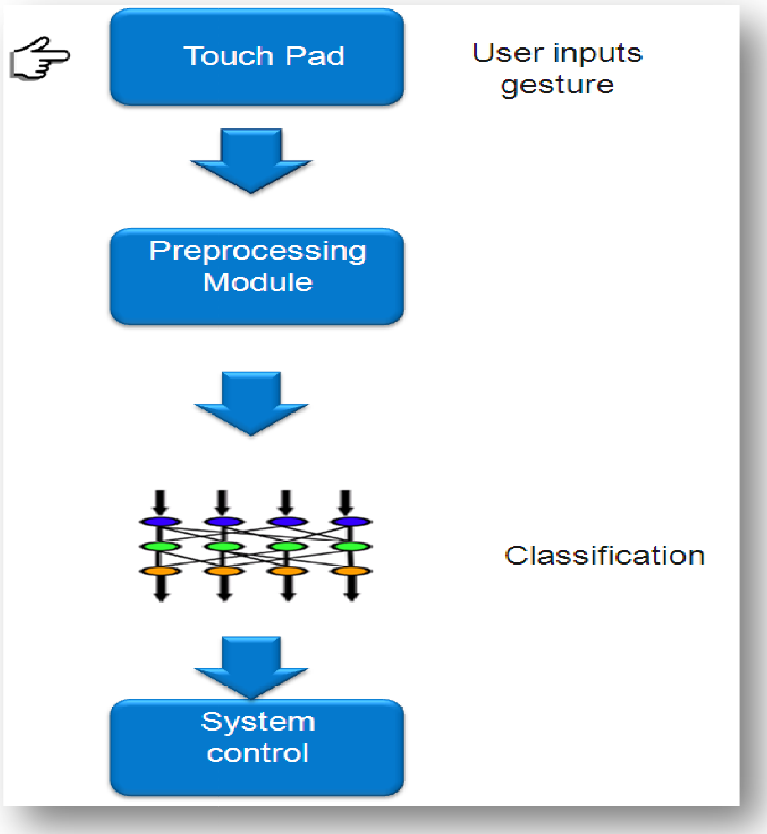


Fig 12 The classification process for system control

Chapter 3: Evaluation of Neural Network Development Package's

There is wide range of both proprietary and open source software products available to Neural Network designers. I will discuss a few alternatives.

3.1 MATLAB®

3.1.1 Fourth Generation Languages (4GL) and MATLAB®

Fourth Generation Languages are a class of programming languages that are usually designed with a particular application in mind. These follow on from Third Generation Languages but they offer higher abstraction. The major objective of these 4GL's is to reduce the amount of programming. 4GL's have been largely used for Rapid Application Development (RAD).
Examples of 4GL's include DataFlex®, WinDev® and MATLAB®.

MATLAB® is a 4GL that allows rapid application development for computer intensive tasks than traditional programming languages such as C, C++ and FORTRAN. MATLAB® specialises in application development in areas such as image processing, financial modelling and analysis as well as technical computing.

MATLAB® provides engineers, scientists, mathematicians and other professionals with companion 'Toolboxes' with relevant tools and a suitable application development environment [19]

3.1.2 Popularity of MATLAB®

Recent surveys show that MATLAB® is number 22 on a list of the most popular programming languages [31]. Perhaps a reason for its position among other programming languages is that it is a proprietary programming language and it is rather expensive [35].

3.1.3 The Neural Network Toolbox in MATLAB®

This is a Toolbox available in MATLAB® that provides users with tools for the design and implementation of Artificial Neural Networks.

The Neural Network Toolbox supports various Neural Network Architectures (supervised and unsupervised networks) and provides several Training and Learning functions. It also provides preprocessing and postprocessing functions such as *fixunkowns, removerows* and *mapminmax*.

Table 5 below shows a list of training functions and associated algorithms associated in the MATLAB® Neural Network Toolbox [34].

Function name	Algorithm
Trainb	Batch training with weight & bias learning rules
Trainbfg	BFGS quasi-Newton backpropagation
Trainbr	Bayesian regularization
Trainc	Cyclical order incremental training w/learning functions
Traincgb	Powell -Beale conjugate gradient backpropagation
Traincgf	Fletcher-Powell conjugate gradient backpropagation
Traincgp	Polak-Ribiere conjugate gradient backpropagation
Traingd	Gradient descent backpropagation
Traingdm	Gradient descent with momentum backpropagation
Traingda	Gradient descent with adaptive learning (lr) rate backpropagation
Traingdx	Gradient descent momentum & adaptive lr backpropagation
Trainlm	Levenberg-Marquardt backpropagation
Trainoss	One step secant backpropagation
Trainr	Random order incremental training w/learning functions
Trainrp	Resilient backpropagation
Trainscg	Scaled conjugate gradient backpropagation
Trains	Sequential order incremental training w/learning functions

Table 5 Lists training functions and related algorithms available in MATLAB®

3.1.4 Preprocessing and Postprocessing in MATLAB®

Preprocessing is when data is processed in such a way as to make it more tractable for analysis and design. Preprocessing Neural Network inputs and targets improves the efficiency of Neural Network training.

Postprocessing generally enables detailed analysis of network performance. The MATLAB® Neural Network Toolbox provides preprocessing and postprocessing functions that enable users to:

- Reduce the dimensions of the input vectors using principal component analysis
- Perform regression analysis between the network response and the corresponding targets
- Scale inputs and targets so that they fall in the default range [-1, 1], or any other user specified minimum or maximum range (using *mapminmax*).
- Normalize the mean and standard deviation of the training set
- Preprocessing and data division are built into the network creation process. [19]

3.1.5 Advantages of MATLAB®

- MATLAB® is quite easy to learn as a programming language. After only a few hours of training, a new MATLAB® user can start developing simulation tools.
- MATLAB® compilers can compile to C, C++ and binary code, allowing the use of different optimization options for high-speed executables.
- The open architecture allows for very rapid extension of the range of functionality of MATLAB® by developing and sharing new toolboxes. MATLAB® is available for a range of environments such as MS-Windows, Linux, Sun Solaris, Apollo, VAX, HP workstations, Gould, Apple Macintosh, and several other parallel machines. .[36]
- Ease of portability.

3.1.6 Disadvantages of MATLAB®

- Some simulations are simply too complex to program in MATLAB® as compared to third generation languages. This can lead to performance cuts.
- The ease with which new toolboxes are developed and then shared also has its drawbacks. For some application domains, there is currently a redundant choice of several overlapping toolboxes, some of which may be partially or entirely dysfunctional. The quality of the software in different toolboxes also varies dramatically.[36]

- MATLAB® is an interpreted language, therefore it tends to be slow compared to 3GL's.[21]

3.2 *STATISTICA*® Automated Neural Networks

***STATISTICA*®** Automated Neural Networks is a proprietary Neural Network software package available from StatSoft®**.** It offers the following features to developers.

- Integrated pre- and post-processing.
- A number of highly optimized training algorithms (including Conjugate Gradient Descent and BFGS).
- Support for combinations of networks and network architectures of practically unlimited sizes organized in network sets for forming ensembles.
- Comprehensive graphical and statistical feedback that facilitates interactive exploratory analyses.
- Full integration with the *STATISTICA*® system.[25]

3.3 SAS®

SAS® is another software package available to developers. The major advantage of neural nets in SAS is that the package is free, but you have to license SAS/Base software and preferably the SAS/OR, SAS/ETS, and/or SAS/STAT products. Neural Network Architectures supported in SAS are:

- Generalized linear model (GLIM) -, which is suitable when there is a linear relationship between the target and the inputs.
- Multilayer perceptron (MLR) - default, which is often the best architecture for prediction problems
- Radial basis function (RBF) [SAS, 2008]

SAS® is clearly outweighed by other available Neural Network Software packages like MATLAB® or STATISTICA® which offer more in terms of available architectures and training functions.

3.4 Third Generation Programming languages

Third generation programming languages like Java, Perl and C# have been used to develop tailor made Neural Network solutions. These have the added advantages of speed and flexibility over 4GL based implementations of Neural Networks like MATLAB®.

The major advantages offered by these mainly proprietary, 4GL Neural Network implementations like MATLAB®, is that they offer optimised development environments for training and simulation [4]. Apart from this, 4GL's are built along the RAD framework, allowing for quicker development and simulation.

Chapter 4: MATLAB® Implementation

4.1 Determining the Neural Net Structure

4.1.1 Deciding on the number of inputs for the neural net

In order to decide on the number of inputs for this project, I had to physically enter gestures on the TicklePad and examine the output at the varying frame rate. In general, for the central frame rate of 15 frames per second, the longest gestures were consistently the circle gestures, consisting of roughly 10 vectors on average while the other gestures typically consisted of between 2 and 10 vectors. So the important question was; how many vectors had to be entered until a clear pattern could be observed that could distinguish between the individual vectors. The closest similarity was observed between the diagonal gestures and the circular gestures. The circle gestures are by far the most complex of the available gesture set and it can be argued that exactly *one* vector can distinguish the other eight non-circular gestures. But the complexity of the circular gestures could not allow this; all the eight non-circular gestures can form part of a circular gesture.

To get around this problem, I physically conducted experiments on the TicklePad to see how many vectors would define a clear circular pattern or curvature, strongly taking into consideration the amount of noise on the Ticklepad. My findings were that four or more vectors could clearly distinguish between a curvature and a (roughly) straight line entered on the TicklePad. The minimum (four) was then taken for better interaction speeds and less computational overhead since these inputs would need to be propagated through the neural net.

4.1.2 The number of neurons in the output layer

The number on neurons in the output layer depends on how big your target set is and the nature of the output layer transfer function. In my problem case, I require only a target set of ten targets. My output layer transfer function being Logsig, whose output range falls between zero and one. This would mean that for a single Logsig transfer function; there is only the possibility of two targets, a zero or a one. So in order to cover my entire set of ten gestures, I would require a minimum of four Logsig outputs (2^4=16). This would also mean I will have 6 extra targets sets. In a typical classification scenario, if any of these six peripheral classes is produced, we assume that the input had large amounts of noise and therefore could not match any of the ten *real* target gestures.

Because of severe noise problems and conflicts in classification, this number of output layer neurons would have to later be changed as will be discussed in chapter 5.

4.1.3 Selection of transfer functions

Several training functions can be applied to a neural net, but these converge differently and have varying output ranges. Selection of the correct transfer functions is thus an important part of designing a solution specific optimised neural net structure. My first constraining factor here is that I am using backpropagation during training, meaning that I am only limited to differentiable transfer functions such as Tansig [4]. Another constraining factor was my decision to produce my output in the form of bits. It would make sense for me to have an activation function in the output layer that squeezes output between zero and one. For this reason, Logsig was a natural choice for the output layer as it produces *that* desired output.

For my first layer and hidden layer, I selected Tansig as my activation function because it allows for faster convergence during training and it is differentiable. [8]

4.1.4 Determining the number of neurons in the first layer and the hidden layer

There are two main approaches to determine an optimum neural net structure. Most experts in industry use genetic algorithms or a trial and error approach to find an optimised neural net structure for their projects. [8] The task of finding a correct neural net structure is a delicate one because a net with too few neurons will tend to misclassify, while one with too many neurons usually leads to overfitting as discussed earlier. I used a trial and error approached because it's a much simpler approach. [4] The number of neurons in the first layer and those in the hidden layer need to be decided upon in this regard, while those in the output layer will depend on the developers target set.

In my trial and error approach I started of with what most experts would call a large "neural net", with forty neurons in the first layer and forty in the hidden layer. The idea being to test how this net with perform on program generated test data, and then continually scale down the net to see if there is a drastic drop in classification, which would mark the end of the scaling down process. [4] All the scripts and tests were done in MATLAB®. I now walk you through the scaling down process, giving you screenshots of typical tests results and their interpretation.

4.1.4.1 Testing the [40.40.4] neural net

My initial performance test was with a neural net with forty first layer neurons, forty hidden layer neurons and four output layer neurons. To evaluate this net, I trained it over the same training set that I would use for the other, smaller nets and used the same evaluation function. To obtain comparable results, I performed the tests on the same gestures, *Line1* and *Circle1*. I will use only the results for the *Circle1* gesture since the results are quite the same as those for *Line1*.

Table 6 below shows *Line1* and *Circle1* gestures used for testing

Line1	Circle1

The MATLAB® script for the evaluation exercise is shown below;

```
%evaluation script for line1
count2=0;
n2=[];
pc2=[];
ns2=-1;
for q2=[0:1:100]
   ns2=ns2+1; % increasing noise levels
  for z=[1:1:100]%performs 100 recognition tests per noise level
   %circle1;
   line1;
   xl=length(P1);%This section of code samples the (x,y) coordinates      'evenly'
   yl=floor(xl/5);
   vl=P1([yl yl*2 3*yl 4*yl 5*yl]);
   v1l=P1([xl+yl xl+yl*2 xl+3*yl xl+4*yl xl+5*yl]);%works great
   data14=[vl' v1l'];
   cae2=[];
   ae2=2*ns2*rand(5,1)-ns2;%generates random #'s between +/-(ns)/2
   be2=2*ns2*rand(5,1)-ns2;
   cae2=[ae2 be2];%5*2 column of random numbers between +/-(ns)/2
   data14=data14+cae2;%add random vector with increasing magnitude
   data14=data14';
   rel=[];%holds vectors
   lel=length(data14);
   kel=1;
   for kel=1:lel-1
      wel=data14(:,kel);
      sel=data14(:,kel+1);
      tel=sel-wel;%subtracting consecutive columns to obtain a vector
      rel=[rel tel];%augment the matrix
      kel=kel+1;
   end
   rel=rel(:);
   data14=rel/100;
   load Neurons.mat
   sol2=(round(sim(net2,data14)));
   if sol2==t1
      count2=count2+1;
   end
   end
   %count=count*10;
   pc2=[pc2 count2];
   n2=[n2 q2];
   count2=0;
end
close all
plot(n2,pc2)
xlabel('Arbitrary Noise Levels ')
ylabel('Percentage Recognition')
title('Ploting Noise levels with recognition for line1 gesture using    [25.25.4]')
```

The following output was produced after running the script.

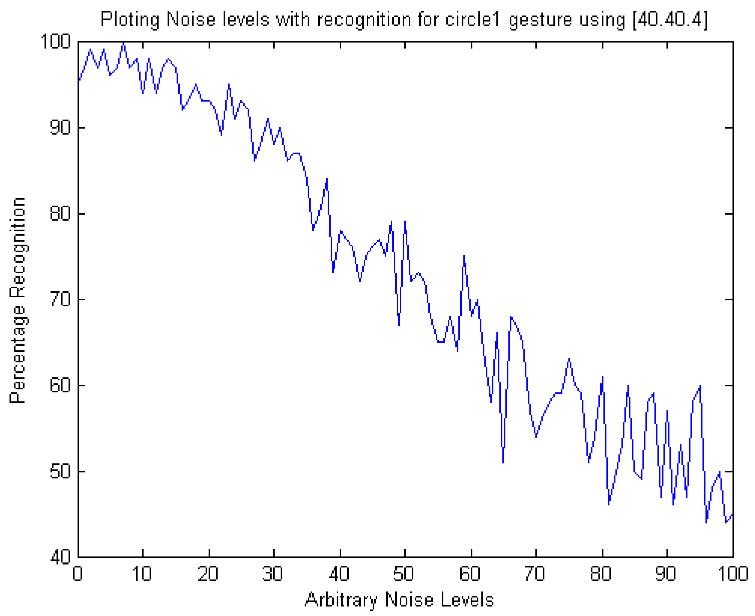


Fig 13 Showing results for testing a [40.40.4] net

4.1.4.1.1 Interpretation

The graph shows the gradual decline in classification as noise levels increase, this is to be expected. An important thing here is that this neural net fails to completely recognise all the gestures when there is absolutely no noise. This can be attributed to the overfitting problem discussed earlier. I will set this as a benchmark in my tests to ascertain an optimum neural net structure.

4.1.4.2 Testing the [30.30.4] neural net

I then proceeded to test a trained [30.30.4] net to see if there would be any significant drop in performance as compared to the [40.40.4] net. I used the same evaluation script used above, in the exact same conditions.

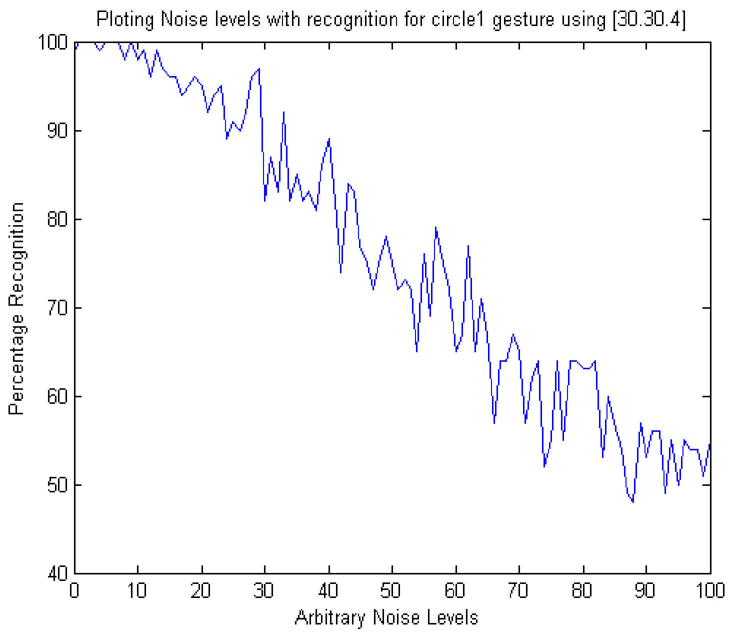


Fig 14 Showing performance results for testing a [30.30.4] net

4.1.4.2.1 Interpretation

Briefly inspecting the first performance graph **(Fig 13)** and **Fig 14** above, the smaller net, [30.30.4], appears to have outperformed the larger net. As noted previously, a neural net with too many neurons suffers from overfitting, which is when a neural net does not classify as expected when small amounts of noise is added, a well sized net would be able to generalise correctly and produce the desired output. Observe also that the net achieves 100% recognition when no noise is applied; it thereby meets my previously set benchmark.

4.1.4.3 Testing a [25.25.4] neural net

Results for testing a trained [25.25.4] net, over the same conditions are shown below

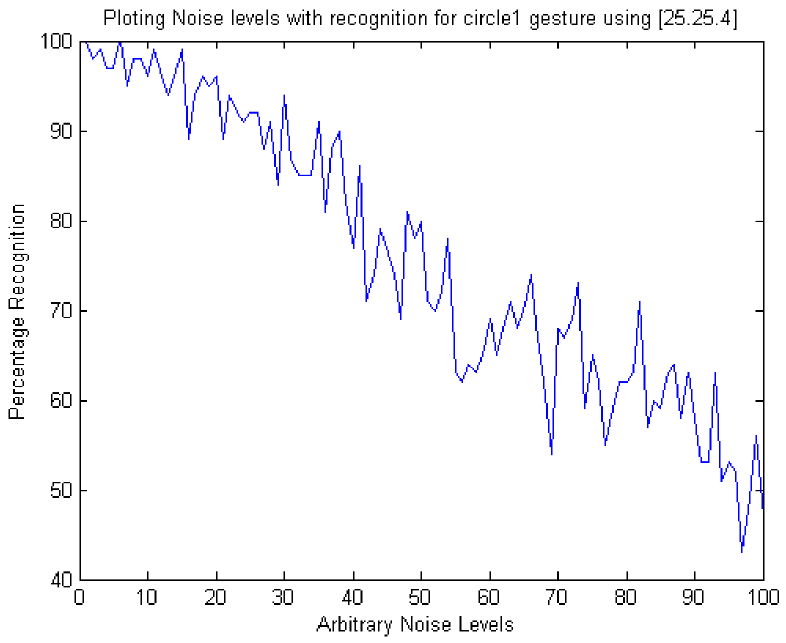


Fig 15 Showing performance test results for a [25.25.4] net

4.1.4.3.1 Interpretation

There is no *significant* drop in performance observed in this net and it meets my previously set benchmark of achieving 100% recognition at 0 % noise levels. This will *pass* as a suitable neural net structure.

4.1.4.4 Testing a [25.20.4] neural net

The result of this test is shown below in the graph.

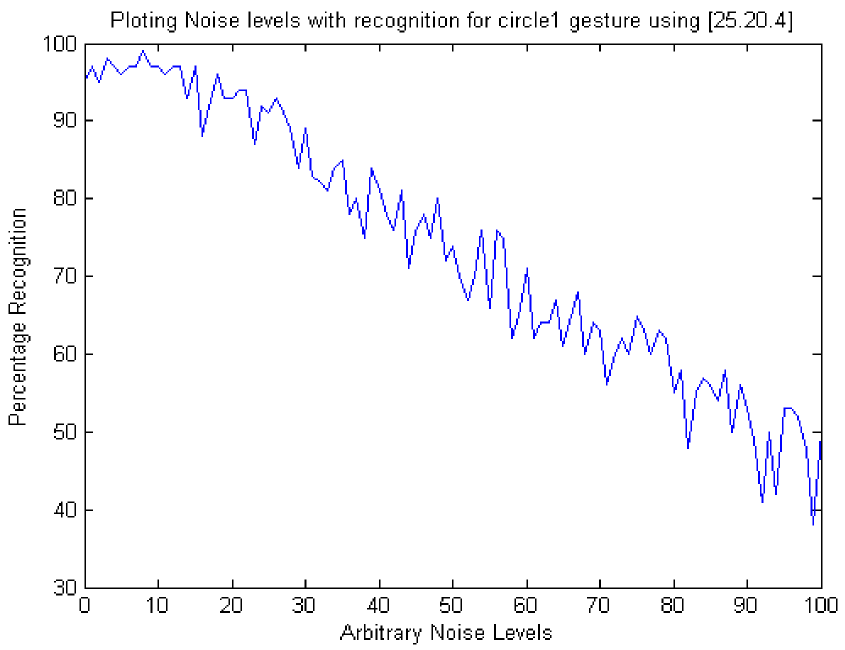


Fig 16 above shows the results for testing a trained [25.20.4] neural net

4.1.4.4.1 Interpretation

You may observe that though there isn't much difference between this graph and the previous one for the [25.25.4] net, it fails my benchmark test of 100% recognition for zero noise level. Because of this reason, I conclude at this point that this [25.20.4] net will be inadequate for my classification process.

4.1.4.4.5 Final conclusions

Since the [25.20.4] net fails my benchmark test, I will not proceed any further with these tests as it is almost certain that subsequent test will produce even poor performance. I then conclude that my optimum neural net structure for my classification problem is [25.25.4].

4.2 Selection of MATLAB® Training algorithm

In **Table 5** of **Section 3.1.3,** I outline the numerous training functions available in MATLAB®. For training my neural net for gesture recognition, I used the MATLAB® function *trainscg*. The reason behind this decision was simply the amount of time that was to be consumed during training. *Trainscg* allows for faster convergence (to targets) during training, especially when dealing with input in the range {0<*x*<1} [4]. In my problem case, as with many neural net problem cases, developers usually deal with large training sets; selection of a fast training algorithm will always save the developer much time.

4.3 Training for gesture classification

As mentioned, the most suitable training approach for our gesture classification problem in BingBee is supervised training via Backprogation. This is when a neural net is iteratively trained against a given set of targets, when particular inputs are fed through. The net must produce these same targets when presented with more or less of the same input sequence (noise tolerance). The diagram below outlines the training process employed in this work.

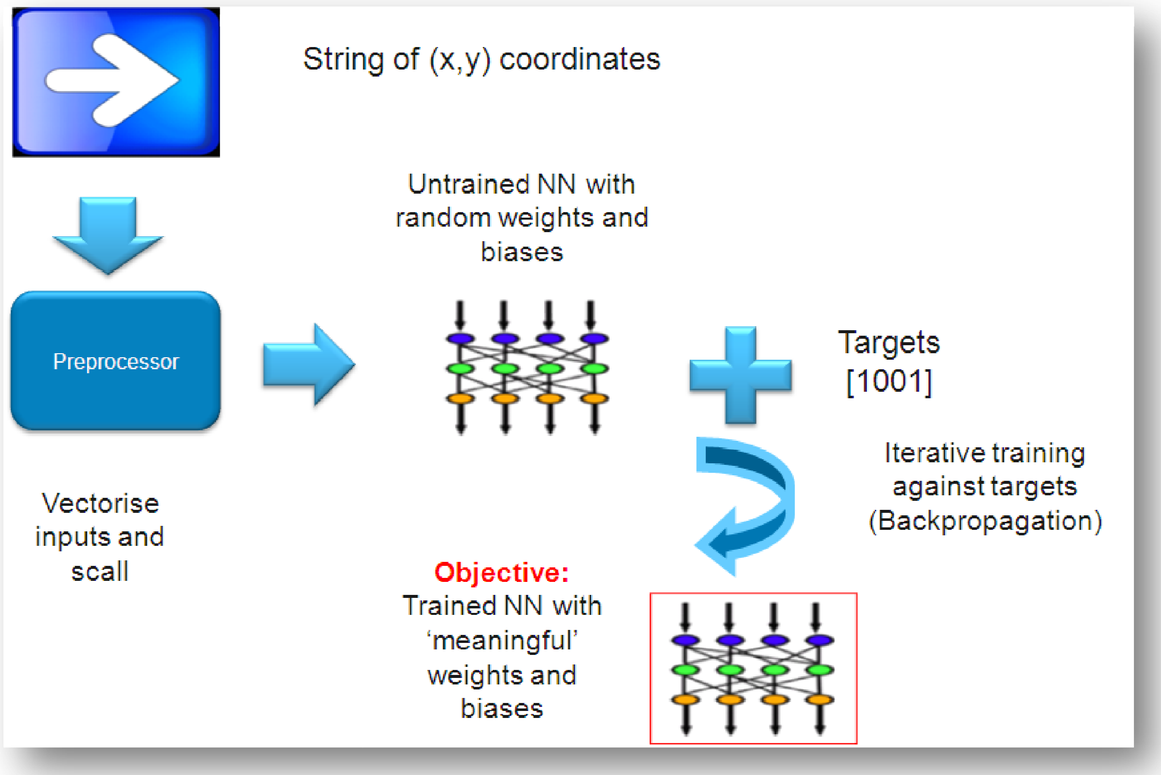

String of (x,y) coordinates
Untrained NN with random weights and biases
Preprocessor
Targets [1001]
Iterative training against targets (Backpropagation)
Vectorise inputs and scall
Objective:
Trained NN with 'meaningful' weights and biases

Fig 17

4.3.1 Inputs

Users are able to input gestures into the system through the touch surface. These gestures are captured as (x, y) coordinates. But for training purposes, it would be impractical to input training gestures manually because large training sets are required [15]. For this reason, I wrote ten MATLAB® scripts that model the ten gestures, each generating (x, y) coordinates that capture the essence of the gesture. I coded the scripts such that each time a particular script runs, it produces the same gesture, but with a small, random amount of noise and to model the type of (noisy) input obtained from the BingBee touchpad as much as possible. MATLAB® scripts that generate the proposed gestures can be found in Appendix A1.

4.3.2 Preprocessing

One of the major limitations of neural computing is that neural nets deal with fixed size input. For this reason, preprocessing is usually a necessary step before passing data into the neural net. The preprocessing module employed during training and simulation will perform this operation and transform the input sequence of (x, y) coordinates into a vector sequence. It is necessary to transform to coordinates into a vector sequence because it has been noted that neural nets are more likely to pick up a pattern when dealing with vectors [36]. Transformation of (x, y) coordinates into vector sequences is already being done on the TicklePad in BingBee and hence will not be part of my final implementation in BingBee.

The MATLAB® script used to preprocess the inputs can be found in Appendix A2.

4.3.3 Training a Neural Net via Backpropagation

As mentioned earlier, neural nets store their knowledge as weights and thresholds. An untrained neural net has random amounts of weights and thresholds (biases), *which have no meaning*. [8]Through supervised learning and backpropagation training against the set target, the weights and biases are altered to *have meaning* in our classification problem. This training process is repeated for all the ten gestures against their targets. The end result is a trained neural net, capable of distinguishing one gesture from another.

The MATLAB® script used for training can be found in Appendix A3.

Chapter 5: C# Implementation and BingBee Integration

5.1 BingBee's Interface design

BingBee's interface is primarily designed to capture touch-surface motion using a webcam at 15 frames per second. Any motion on the touch surface generates TickleMessages that are fed into a queue. The software implementation allows for separation of input from the KeyPad area and from the TicklePad. Key-area inputs are mapped to their corresponding keys using a configuration file while TicklePad inputs can be used for mouse motion control or for gesture recognition.

Since I'm not interested in the key area, I'll concentrate my discussion on the TicklePad. Interacting with the TicklePad/TouchPad generates *TickleMessage's*. These TickleMessage's from the TicklePad are of types KeyDown, KeyUp and KeyMoved. Each name depicts the type of activity being captured on the TicklePad. In a normal sequence of events, we would expect a KeyUp or KeyMoved TickleMessage only after an initial KeyDown action.

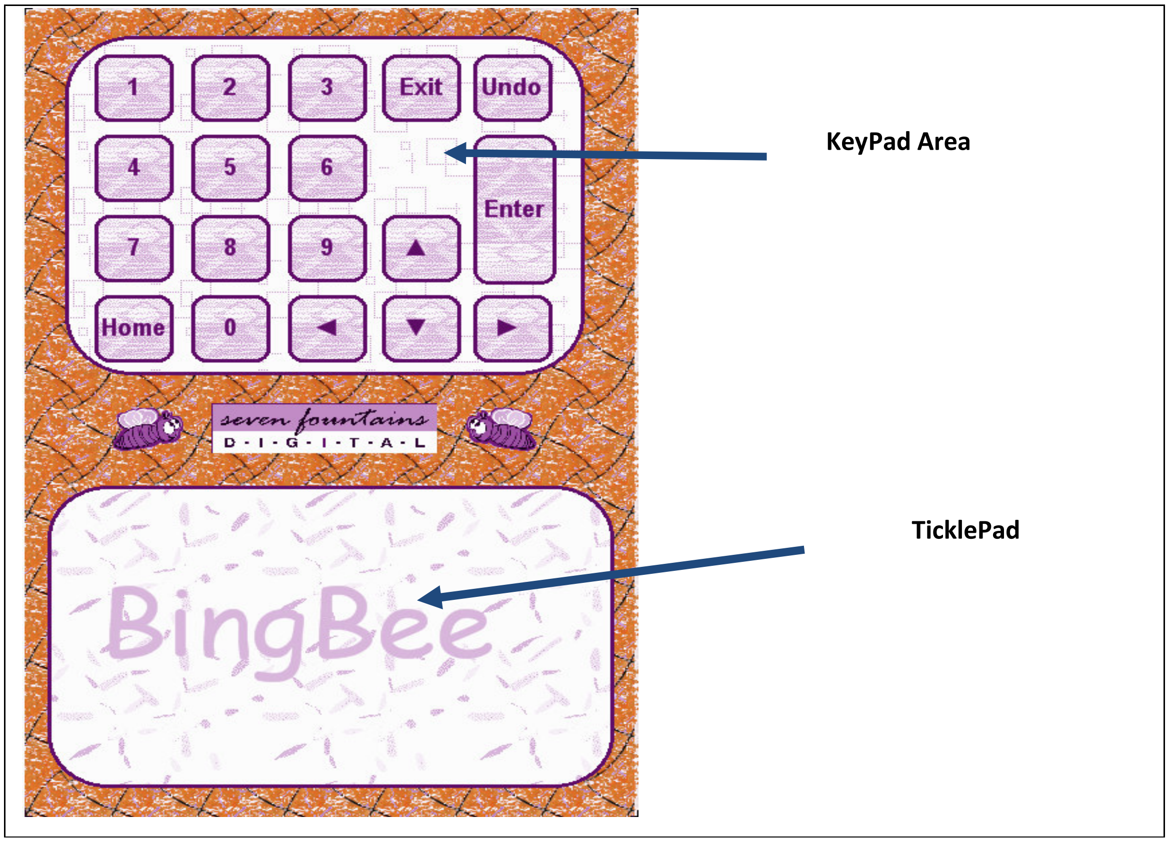


Fig 18 Showing BingBee's TicklePad ant KeyPad areas

5.1.2 Preprocessing of user inputs in BingBee

In BingBee's interface design, KeyMoved TickleMessages are generated each time motion is detected on the Ticklepad, as such; these will be used for gesture classification as they capture the essence of a gesture.

In the neural net training phase of this project, I generated *n* (x, y) coordinates corresponding to the respective gesture. These were converted to an *n-1* vector sequence that was scaled down.

The already defined *Dx* and *Dy* property of the KeyMoved TickleMessage represent the change in x and the change in y from the previous TickleMessage generated off the TicklePad. These will be ideal for my neural classier which uses vector input for gesture classification. There will be no need to Vectorise my input as it will be in vector format already. I only need to capture the *Dx* and *Dy* values and scale them down before feeding the result into my neural classifier. As pointed out in previous discussions, one major disadvantage of neural nets is that they require a fixed sized input. For our target case in our BingBee implementation, we shall use four vector pairs (four *Dx* values and four *Dy* values), meaning eight (scaled) inputs will be fed into the neural net. This scenario implies that KeyMoved TickleMessage's will be queued until they meet the required number (eight), before we can classify. Failure to meet this threshold will mean we have insufficient inputs for classification and the gesture will have to be re-entered.

5.2 Implementing a neural net in C#

There are no C# library's that provide API's for the creation, training or simulation of neural nets. I coded a Neuron class (found at Appendix B1) that defines the class variables and contains the Neuron constructor. It also defines the public methods for interacting with the Neuron object. The code snippet below shows the class variables and two constructors.

```
public class Neuron
    {
        #region PROTECTED FIELDS (State variables)
        protected double[] w;
        protected double[] input;
        protected double threshold;
        protected int N;
        protected ActivationFunction f = null;
        protected double o;
        #endregion

        public Neuron(double thresh, ActivationFunction af, int Ni)
        {
            w = new double[Ni];
            f = af;
            threshold = thresh;
            N = Ni;
        }
        public Neuron()
        {
            w = new double[8];
            f = new Tansigmoid();
            threshold = 0;
        }
```

I also defined an interface, ActivationFunction (found in Appendix B2) that defines the necessary activation functions Logsig and Tansig. The following code snippet defines the interface and public methods for the LogSigmoid (Logsig) activation function.

```
public interface ActivationFunction
    {
        double Output(double x);
    }
    #region Logsigmoid

    //for the output layer
    [Serializable]
    public class SigmoidActivationFunction : ActivationFunction
    {

        // Get the name of the activation function

        public string Name
        {
            get { return "Log Sigmoid"; }
        }

        public double Output(double x)
        {
            return (double)(1 / (1 + Math.Exp(-x)));
        }
```

A third class, Classifier (Appendix B3), calls the Neuron constructor and implements the activation function interface. Scaling the input and subsequent classification is also done here. The weights and thresholds for the neurons are imported from the trained neural prototype in MATLAB®. An alternative to this approach was to develop and train the neural based recognisor in C#.

The reason why we chose to prototype our solution in MATLAB® was because MATLAB®'s training functions have been developed and optimised over time. According to [Burton, 2008], it is difficult to develop and train your own neural network that will perform at par with neural solutions developed in MATLAB®. [8]

5.2.1 Extracting weights and biases from MATLAB® for BingBee integration

The following MATLAB® commands assign the variable *g* to hold the first layer biases (thresholds) of the first layer, and then write *g* to a Microsoft Excel file (import).

```
>>g = net2.b{1};

>>xlswrite ('import', g);
```

The biases obtained from the Excel file can then be hard coded into the respective first layer neuron through the neuron constructor. I placed the (25 first layer) thresholds in an array as follows.

```
double[] thresh1 = new double[25] { 1.303793107, 1.688246453, -1.243791373, -1.37607394, -0.794420995, -0.371868635, 0.127802655, 0.374340852, 0.560241243, 0.108020291, 0.302797518, -0.166342412, 0.121101233, -0.015519909, 2.506639247, -0.036533335, -0.504898662, 0.139638468, 1.0364786, 1.294939702, -1.19264938, 0.7718653, 1.427070406, -1.863300612, -1.791804725};//25 threshold values for layer 1
```

The same process is repeated for the respective layers and for the weights. The following code snippet instantiates the three layers, each consisting of the assigned number of neurons.

```
Neuron[] Layer1 = new Neuron[25];//the first layer neurons
Neuron[] Layer2 = new Neuron[25];//the 2nd layer neurons
Neuron[] Layer3 = new Neuron[4];//the output layer neurons
```

The following code snippet, part of the Classifier class (found in Appendix B3), takes in an array of inputs (scaled vectors), passes it through the neural net layers and returns an array as output. This method handles all the neural calculations.

```
public int[] Outputs(double[] a)
        {
       double[] OutputLayer1 = new double[25];
       double[] OutputLayer2 = new double[25];
       int[] OutputLayer3 = new int[4];
       for (int k = 0; k < 25; k++)
       {
          OutputLayer1[k] = Layer1[k].ComputeOutput(a);
       }
       for (int k = 0; k < 25; k++)
       {
          OutputLayer2[k] = Layer2[k].ComputeOutput(OutputLayer1);
       }
       for (int k = 0; k < 4; k++)
       {
         if (Layer3[k].ComputeOutput(OutputLayer2) >= 0.5) OutputLayer3[k] =1;//Post Processing...
       else OutputLayer3[k] = 0;
       }
       return OutputLayer3; }
```

5.3 Evaluating the gesture interface

After successfully transferring the weights and biases from MATLAB® into my C# implementation, it was then time to test and see whether the gesture recognisor would function as expected in BingBee.

As mentioned, the gesture interface is specifically to target control in a 3D fly-through scenario. The lists of games installed on BingBee at present are as follows;

Game	2D game	3D game
Tetris	✓	
Xox	✓	
Minesweeper	✓	
Slidinggame	✓	
Rubliccolors	✓	
Rubicgame	✓	

Boxworld	✓	
Reversi	✓	
Dropblock		✓
Timestables	✓	
Sketchart	✓	
Sudoku	✓	
FreeDraw	✓	
WireArt	✓	
MemoryPairs	✓	
MemorySequence	✓	

Table 7 Showing the list of games currently installed on BingBee.

After careful consideration, Tetris was identified for implementing and initial testing of the gesture interface to evaluate its capabilities and performance. The ultimate goal though is that the interface will be used by 3D games as more 3D games are added.

5.3.1 Integrating the gesture recognisor with Tetris

Tetris is a simple 2D game in which random shapes are presented to the player and the task is to arrange the shapes as neatly as possible as more and more shapes are added to the game environment. Buy 'completely filling a row', a user gets the bonus of having that entire row removed, thereby lowering the structure and thereby buying the player more game time and earning him/her more points.

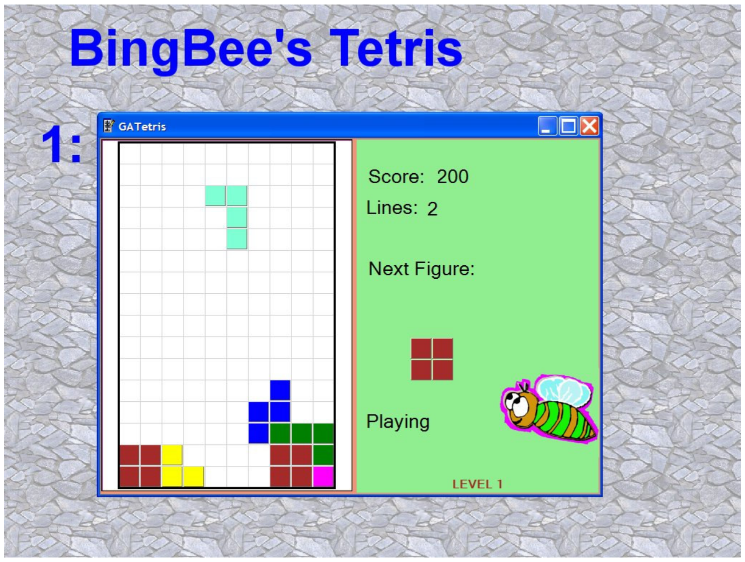


Fig 19 Showing a screenshot of BingBee's Tetris

The table below summarises the current interface design for Tetris

Control Method	Function
Key Pad Left Arrow	Move Left
Key Pad Right Arrow	Move right
Key Pad Up Arrow	Rotate Left
Key Pad Down Arrow	Move Down Faster
Key Pad Enter Button	Drop current shape to bottom
TicklePad/Gesture's	None

Table 8 Showing the current KeyPad based interface for Tetris

The following flow diagram below represents the control process for KeyPad based control.

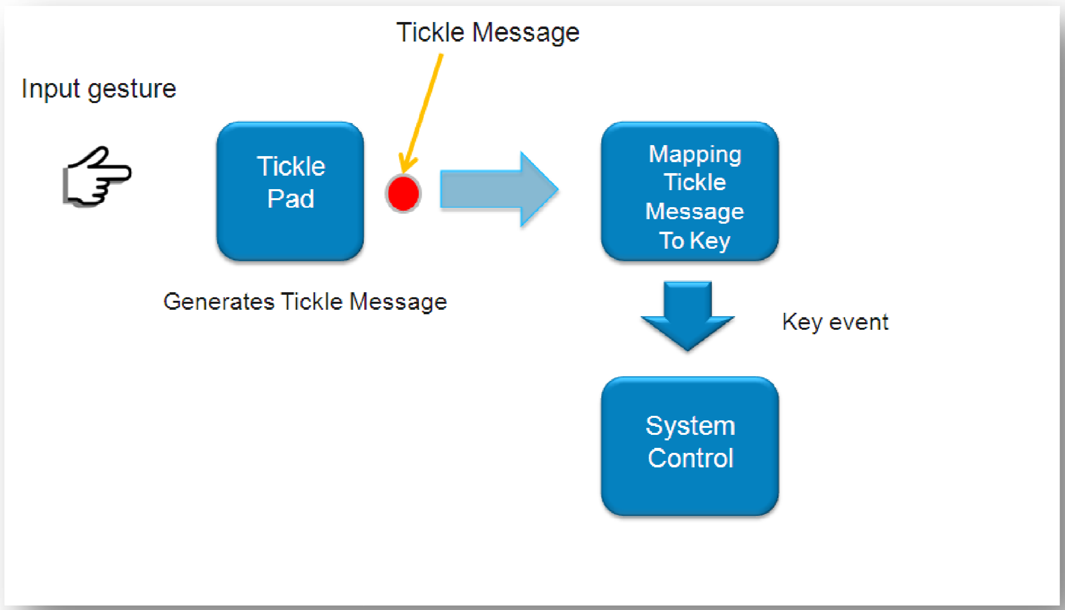


Fig 20 Showing the implementation of Key Pad based control

The following table below represents the proposed gesture interface for Tetris

Gesture	Control action
LeftToRight gesture	Move Right
RightToLeft gesture	Move Left
Down gesture	Move current shape down faster
Anticlockwise Circle gesture	Rotate Right
Clockwise Circle gesture	Rotate Left
DoubleTap	Drop current shape to bottom

Table 9 showing the proposed gesture interface for Tetris

The following flow diagram summarises the proposed gesture based control process

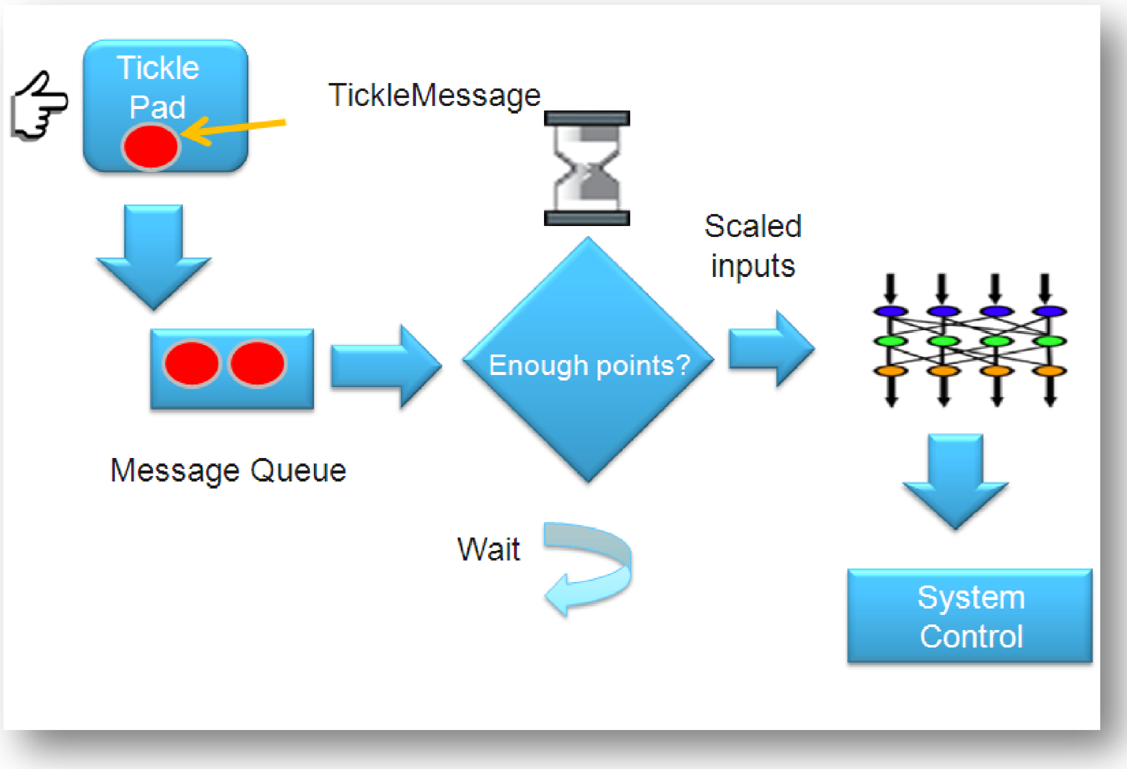


Fig 21 Summarising the proposed gesture interface

As pointed out earlier, the aim of adding a gesture interface was to make interaction with the system more intuitive, and perhaps add an extra modality of interaction as is common with most modern systems. In attempting to improve the interface design, I created a rotate left control, previously not catered for in the previous setup. It is my hope that having a rotate right and rotate left gestures is an improvement on the interface design for Tetris and is more intuitive, as compared to having a single UP arrow representing a single rotate left direction with no provision for rotating right.

5.3.2 Evaluation of the gesture interface in Tetris

5.3.2.1 A comparison between arrow key controls and hand gestures

To evaluate the new gesture interface, I initially compared the performance of the gesture interface with the already existing Key Pad interface.

Table 10 below summarises my initial findings.

Property	Key Pad Action	Hand Gestures	Improvement?
Accuracy of control modality	100%	<100%	✗
Speed of control Modality	Fast	Slower	✗

Table 10 Summarising differences between KeyPad based control and Gesture based

Most neural implementations of do not always demonstrate a 100% accuracy rate, and so is the case with our gesture recognisor. I attribute most of the classification errors to the noisy input from the TicklePad. As my earlier experiments demonstrated, we expect 100% accuracy when there is no noise present and this continues to fall gradually as more and more noise is added.

Another problem with the gesture interface is the interaction speed. In order to classify a single gesture, the gesture recognisor must wait for at least four TickleMessages. The amount of time taken to generate the four TickleMessages depends on the users hand motion speed and on the webcams frame rate. (A higher frame rate would allow more TickleMessages to be capture per unit time) To add to this delay, the inputs must be fed through a three layer, 54 neuron neural net.

5.3.2.2 Initial Test Results

The diagram below highlights the impact of from rate on the total classification interval. The total classification interval in this case is the amount of time spent waiting for the correct number of inputs plus the entire period spent propagating the entire input stream through the 54 neuron net. The result will also need to be mapped to the corresponding key event for system control as with key based control. These figures therefore represent the extra amount of time for controlling the system via gesture recognition as compared to KeyPad inputs.

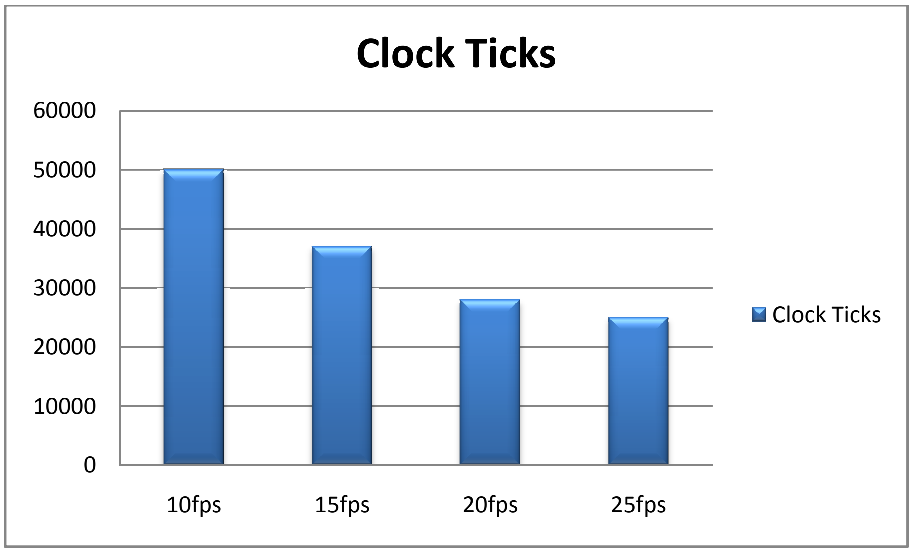


Fig 22 showing the impact of frame rate on interaction speed

A higher frame rate will no doubt increase the interaction speed, yet an evaluation of the gesture recognisor indicated that lower frame rates allowed for better classification. The chart below illustrates this point.

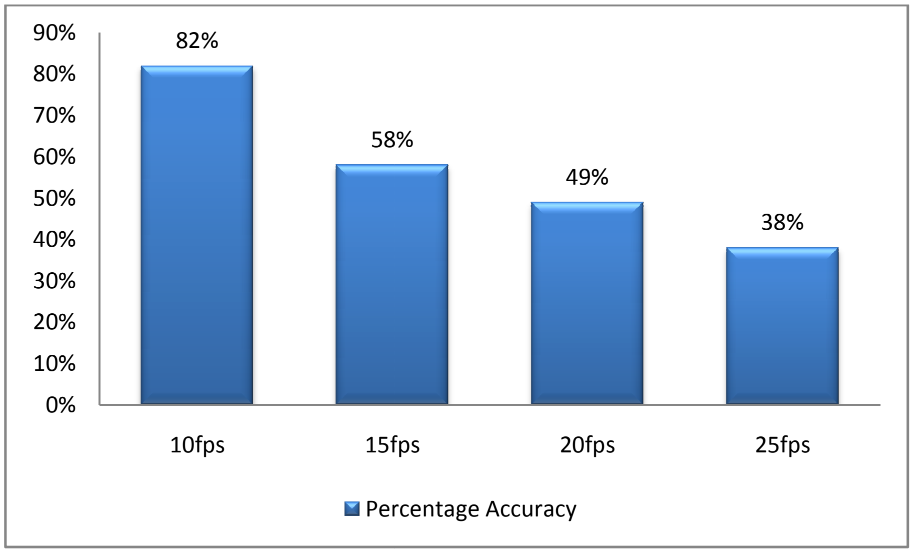


Fig 23 Showing the overall performance of the gesture recognisor at the various frame rates

The above results clearly demonstrate that the gesture interface is fairly distant in terms of classification performance and interaction speed.

5.4.1 Error Analysis and Correction

5.4.1.1 Source of errors

After completing these experiments, I sought after trying to improve the gesture recognisor's classification performance. To begin with, I identified two potential sources of accuracy in the gesture recognisor. These are;

- Misclassification due to noisy input
- Gesture is too short, hence no classification takes place, meaning desired action is not implemented

To evaluate the extent to which noise or gesture length contributed to the observed errors, I took a random sample of some of the observed errors in the above trial. My investigations are summarised below;

Type of error	**Percentage contribution**
Misclassification [input too noisy]	87.5%
Gesture too short [Human Error]	12.5%

Table 11 Showing the main sources of error

Analysis of the gesture input giving rise to incorrect classification also revealed that the first two TickleMessages were usually the noisiest. This may be attributed to hand motion being unstable at the time that the user presses the TicklePad to enter his/her gesture, but gradually stabilising as the user completes the gesture.

5.4.1.2 Conflicts involving diagonal gestures

More detailed test on the gesture recognisor on the entire gesture set revealed that due to noise effects, two of the diagonal gestures were frequently misclassified. These two were not included in the earlier tests with Tetris because they were not deemed intuitive for Tetris's game control as compared to the arrow and circle gestures which were then selected. The frequency of the misclassification was quite high in comparison with the other gestures.

The table below summarises the problem of the two diagonal vectors.

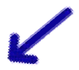
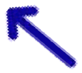
Gesture	Training target	Error outputs
	**[1101]**	**[0101]**
	**[0111]**	**[0101]**

Table 12 Showing two conflicts present in the gesture recognisor

You will notice that the error output is just one bit off the actual target. To compound this problem, the error output belongs to the left-arrow gesture, meaning entering one of these two actions could possibly be erroneously conceived by the system as a left-arrow gesture. This scenario does not hold well for our main aim; namely to build an intuitive gesture interface for BingBee. The system would be highly unusable and very unpredictable.

5.4.1.2.1 Solving the off-by-one-bit problem

This problem could easily be solved by defining the targets set in such a way that no two targets sets differ by only one bit. The only way to implement this would be to enlarge the present length of the target set from four bits. New targets would have to be reassigned and the neural net retrained for the new architecture, differing only in the output layer.

I retrained the net using a [25.25.10] neural net with the following target sets for the respective gestures. I also added more noise to the training sets during this retraining exercise, seeing that my previous classifier could not cope with the encountered noise levels.

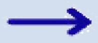
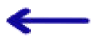
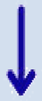
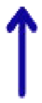
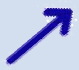
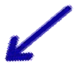
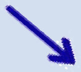
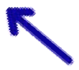
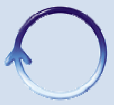
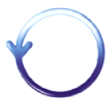
Gesture	Old Targets	New Targets
	[0 1 0 1]	[0 0 0 0 0 1 0 0 0 0]
	[0 0 1 1]	[0 0 0 0 1 0 0 0 0 0]
	[0 1 1 0]	[0 0 0 1 0 0 0 0 0 0]
	[1 0 0 1]	[0 0 1 0 0 0 0 0 0 0]
	[1 1 1 0]	[0 0 0 0 0 0 1 0 0 0]
	[1 1 0 1]	[0 0 0 0 0 0 0 1 0 0]
	[0 0 0 1]	[0 0 0 0 0 0 0 0 1 0]
	[0 1 1 1]	[0 0 0 0 0 0 0 0 0 1]
	[1 0 1 0]	[1 0 0 0 0 0 0 0 0 0]
	[1 1 0 0]	[0 1 0 0 0 0 0 0 0 0]

Table 13 Showing the respective gestures and the new target set

5.4.1.3 Noise during gesture input

Analysis of misclassified gestures revealed that most of the noise was generated at the inception of the gesture and towards the end as the user follows through to complete the gesture. They were usually several, very small, chaotic vectors at the beginning of most misclassified gesture. Towards the end of most misclassified gestures were either the same, small chaotic vectors or very large vectors. These results provided a basis for me to conduct some kind of input sanitisation on the vector inputs.

The diagram below highlights the stark difference between typical training set inputs and actual misclassified inputs obtained from the TicklePad.

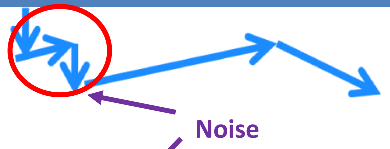
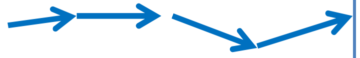
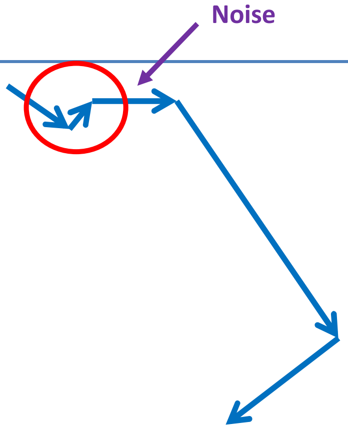
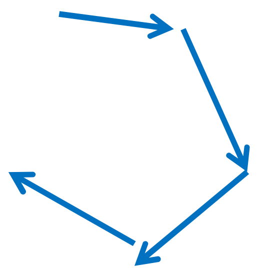
Gesture	Typical input vector (misclassified).	Typical training vectors with some noise added.
		
		

Table 14 Noisy input vs. training samples

It is also important to note that due to low frame rates, inputs were restricted to the first four vectors because not much data could be captured at low frame rates. Unfortunately, these first inputs were the noisiest and scarcely resembling the training examples.

5.4.1.3.1 Input Sanitization/Noise filtering

Having identified the main source of error in my gesture classifier, I sought to improve the performance of the classifier with some input sanitization. It was clear that most of the misclassified gestures either had very large vectors in them, or numerous tiny ones.

This made the decision of how to sanitize the input much easier. Excessively large or small vectors were simply to be filtered out. The main drawback here is that we would require much larger input space.

The diagram below details how a typically noisy input with enough points can be filtered to remove noise for the clockwise circle gesture.

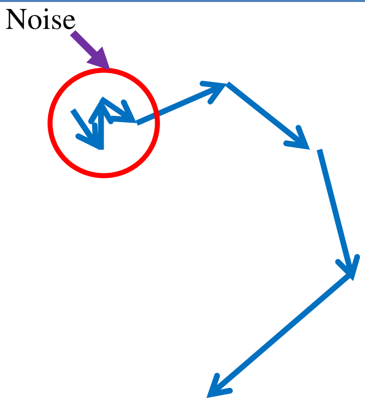
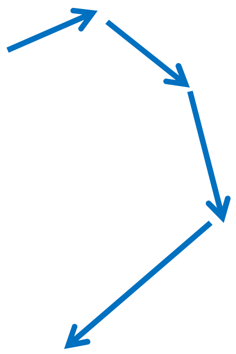
Gesture	Input Vector	Filtered result
		

Table 15 Showing proposed input filtering

The filtered result bears deeper resemblance to the training set (shown in the previous diagram) and hence will more likely be classified correctly than the raw, unfiltered input.

5.5 Evaluation of gesture recognisor with input sanitization and expanded target set.

I then added some input sanitisation to my recogniser and the new neural net structure with ten outputs and evaluated the results of this experiment. The flow diagram representing the new gesture recognition process is as follows;

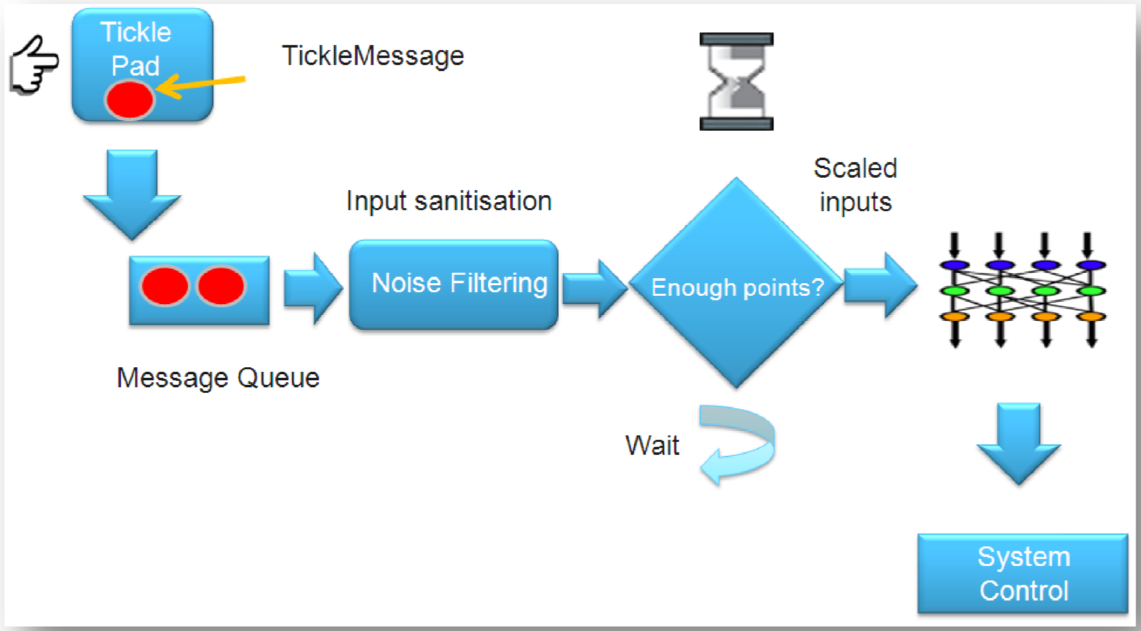
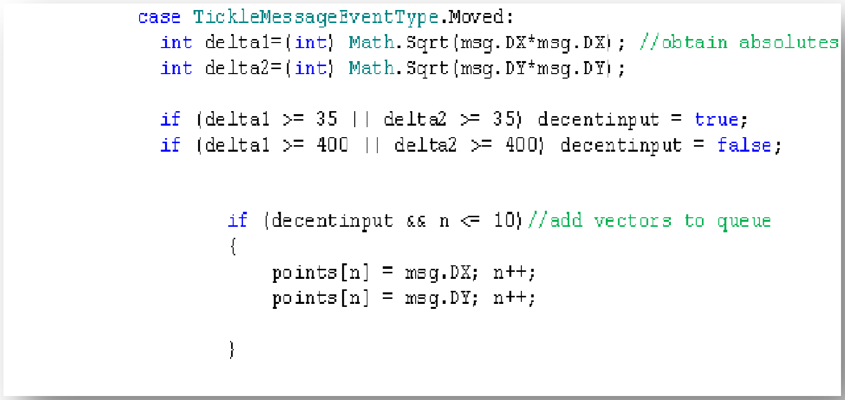


Fig 24 Showing a flow diagram of the new gesture recognition process

I implemented the noise filtering by looking for excessively large or small inputs by setting flags as follows.



```
case TickleMessageEventType.Moved:
  int delta1=(int) Math.Sqrt(msg.DX*msg.DX); //obtain absolutes
  int delta2=(int) Math.Sqrt(msg.DY*msg.DY);

  if (delta1 >= 35 || delta2 >= 35) decentinput = true;
  if (delta1 >= 400 || delta2 >= 400) decentinput = false;


        if (decentinput && n <= 10)//add vectors to queue
        {
            points[n] = msg.DX; n++;
            points[n] = msg.DY; n++;

        }
```

Fig 25 Showing C# code snippet for the gesture recognisor with input filtering

I tested the new gesture interface at the various frame rates using the arrow left gesture, arrow right gesture and the clockwise circle gesture, running between 50 and a hundred tests per frame rate. The results for this evaluation are as follows;

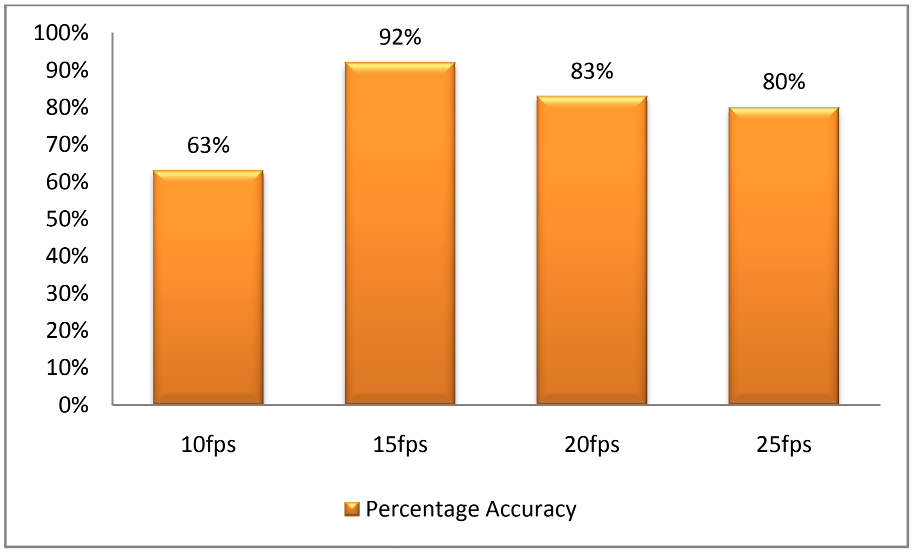


Fig 26 showing final test results

5.5.1 Analysis of final results

The table below provides explanations of the above findings

Frame rate	Accuracy	Comments
10 fps	63%	Input filtering severely constraints the number of accepted inputs, at low frame rates, the main source of error becomes inadequate inputs after filtering. User generally needs to input long gesture for recognition.
15 fps	93%	A fairly acceptable accuracy figure given the amount of noise, input filtering does not filter much data because of the larger frame rate. Inputs most closely match training sets.
20 fps	83%	At higher frame rates, more and more small sized vectors are created, meaning more filtering. Main source of error here is inadequate inputs after noise filtering.
25 fps	80%	Again even higher frame rates results in high input filtering, main source of error is again inadequate inputs resulting in no classification.

Table 16 above providing explanations on recognisor classification

5.6 Summary of this chapter

After implementing these changes, I realised the following changes in the upgraded recognisor with input filtering and expanded target set.

Property	Observed change	Improvement?
Interaction Speed	Faster	✓
Accuracy of Classifier	Better	✓

Table 17 Showing improvements in the gesture recognisor after error analysis and correction

These changes allow for a fairly acceptable gesture based interface. The gesture recognisor may not match KeyPad based control in terms of interaction speed and accuracy, but at 93% optimum classification capability, it does offer an alternative control modality that users may find more intuitive for 2D or 3D gaming control.

Chapter 6: Conclusions and Possible Extensions

6.1 Conclusions

I conclude that the neural net based gesture recognisor has proved to be suitable for our gesture recognition problem in BingBee as seen by the performance tests. This neural net based solution was developed using MATLAB®'s power libraries through its Neural Network Toolbox. The critical weights and biases that define the net were hard coded into a C# implementation for integration in BingBee. The new gesture interface has been tested on BingBee's Tetris game, with some success. The target implementation for the gesture recognisor still remains for 3D gaming control, though some 2D games like Tetris can use part of the gesture set.

It may not meet the market standard of 97% accuracy [37] for complete user satisfaction but at an optimum classification performance of 93%, it does come fairly close to the standard, particularly taking into consideration the amount of noise coming into the recognisor from the Pad. It is also important to note that most classification errors at this point are mainly due to users entering gestures that are too short, and these cannot be classified and hence result in no action.

In this work, I have also endeavoured to develop a highly optimised neural net solution for our gesture recognition problem in BingBee, prudent error analysis has allowed us to identify the main cause of misclassification error and to find ways to work around these, namely input sanitization to handle noise and increasing the target set to avoid conflicts in classification.

MATLAB® has been proven in this work to be an adequate tool for the development and simulation of this project's feed-forward neural net based gesture recognisor. The wide range of training, post and pre-processing functions provides a highly optimised development environment suitable for training and simulation of neural nets.

6.2 Possible Extensions

I have identified several possible extensions for this project. Firstly there can be an increase in the gesture set. This may however require building a larger neural net or perhaps introducing a new gesture recognition approach altogether. In this light, an alphabet can also be added which I think may be popular with the children.

Secondly, the performance of the recognisor does not match industry standards and performance is not uniform across the available frame rates. In this regard, more work can be done to improve upon these attributes of this projects gesture recognisor.

Finally, our recognizer recognizes 2D gestures made on the pad surface. Some modern day systems have moved on to 3D gesture recognition, as seen with Sony's EyeToy® discussed earlier for gaming control. In an effort to make the system more intuitive and in adding to the number of control modalities, perhaps the next step could be to attempt a 3D gesture interface that could be added to BingBee's interface in the near future.

7. References

[1] Aditya Ramamoorthy et al. "*Recognition of dynamic hand gestures* ", page 1-13. Department of Electrical Engineering IIT New Delhi-110016 India. Department of Computer Science and Engineering, IIT New Delhi-110016, India.7 October 2002.

[2] Andrew Vogt and Joe G. Bared. "*Artificial Neural Networks"* <http://www.tfhrc.gov/safety/98133/ch02/body_ch02_05.html> Retrieved 20/4/8

[3] BingBee. www.BingBee.com Retrieved 02/11/8

[4] Burton Mike, lectures and lecture notes, 2008.

[5] Cadoz, C. "*Les realites virtuelles*". Dominos, Flammarion. 1994.

[6] Charles Cohen*," A Brief overview of Gesture Recognition".* 1999 http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/COHEN/Gesture_overview.html Retrieved 19/4/2008

[7] Chowdhury, Badrul H. and Wilamowski, B.M. "*Security assessment using neural computing*". Institute of Electrical and Electronics Engineers, 1991.

[8] Denis Riordan, lectures and lecture slides, 2008. Web site at http://hobbes.ict.ru.ac.za/csdr/cs404/resources/

[9] I. Guyon, P. Albrecht, Y. LeCun, J. Denker, and W. Hubbard. "*Design of a Neural Network Character Recognizer for a Touch Terminal", pages 105-119.* Pattern Recognition, 1990

[10] Hannah Slay et al. *"BingBee, an Information Kiosk for Social Enablement in Marginalized Communities".* SAICS 2006

[11] Hossein Bidgoli. "*Intelligent Management Support Systems",* page 201. Greenwood Publishing Group, 1998.

[12] Hyeon-Kyu Lee and Jin-Hyung Kim "*Gesture spotting from continuous hand motion",* pages 1-8, Department of Computer Science, KAIST, Taejon, South Korea. September 1997

[13] Ian Buck, "*New Applications*", pages 2-17.merrimac.stanford.edu/applications/newaps-merrimac.12.8.03.pdf Retrieved 24/6/08

[14] Japan Singapore AI centre. "*Neural Network Applications*"
http://tralvex.com/pub/nap/index.html Retrieved 20/5/8

[15] Jack V Tu. "*Advantages and Disadvantages of Using Artificial Neural Networks*", pages 1-7. Institute for Clinical Evaluative Sciences, North York, Ontario, Canada, Department of Medicine. 1996

[16] Jie Yang, Yangsheng Xu. "*Hidden Markov Model for Gesture Recognition* ", pages 4-15. The Robotics Institute, Carnegie Mellon University Pittsburgh, Pennsylvania 15213 May 1994. Carnegie.

[17] Kay M. Stanney." *A Handbook of Virtual Environments*", page 223. Lawrence Erlbum Associates, 2002.

[18] K Gurney "*Supervised Learning*"
http://www.shef.ac.uk/psychology/gurney/notes/l10/subsubsection3_3_6_2.html Accessed 04/05/08

[19] Mathworks http://www.mathworks.com/products/MATLAB®/index.html?sec=apps Retrieved 22/6/8.

[20] M D Hasanuzzaman et al. "*Adaptive Visual Gesture Recognition for Human Robot Interaction using Knowledge Based Software platform*", pages 1-12. North-Holland Publishing Co. Amsterdam, Netherlands, 2007.

[21] Melissa Lin. "*Problem Solving with MATLAB®*", pages 12-20, HenEm, Inc. Parkville. The Distance Learning Center.2005

[22] Philipp Koehn "*Combining Genetic Algorithms and Neural Networks: The Encoding Problem*" The University of Tennessee, Knoxville, USA.

[23] Robert Turetsky, **"***Training Neural Networks*", pages 7-58, Systems, Man and Cybernetics Society IEEE, North Jersey Chapter. December 12, 2000
www.ee.columbia.edu/~rob/talks/neuralnet.ppt. Accessed 18/4/8

[24] SAS, 2008 <http://www.sas.com/technologies/analytics/datamining/miner/neuralnet.html> Retrieved 23/6/8.

[25] StatSoft *http://www.statsoft.com/products/stat_nn.html* Retrieved 20/06/08.

[26] Stanford http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Neuron/index.html Retrieved 04/05/08.

[27] Lubomir T. Dechevsky, Arne Lakså. "*A Brief Report on MATLAB®*" pages 1-9. April 2004.

[28] Self Organising Maps http://www.ucl.ac.uk/oncology/MicroCore/HTML_resource/SOM_Intro.htm. Retrieved 20/5/8

[29] Techeblog http://www.techeblog.com/index.php/tech-gadget/video-wiimote-used-for-touchless-Gesture-Recognition-system-microsoft-surface- Retrieved 20/06/08

[30] Tae-Kyu Lee and Roberto Cipolla. "*Gesture Recognition under Small Samples Sizes*", pages 1-10. Sidney Sussex College, University of Cambridge, Cambridge, CB2 3HU, UK. 2007

[31] Tiobe, 2008. "*Programming Community Index for November 2008*". http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html Retrieved 03/10/08.

[32] Tom Mitchell, *"Machine Learning"*. McGraw-Hill, 1997.

[33] Ryotaro Kamimura, "*Analysis of Neural Net Applications Conference*", pages 14-28, ACM, 1991

[34] Vladimir Vacic. "*Summary of the training functions in MATLAB®'s NN toolbox*", pages 1-9, http://www.cs.ucr.edu/~vladimir/cs171/nn_summary.pdf Retrieved 19/04/2008

[35] Java.net. http://wiki.java.net, 2008. Retrieved 04/05/08

[36] Will Dwinnell, Dwinnell Consulting. Correspondence via email, web site at http://will.dwinnell.com/.

[37] Araokar, S. "*Visual character recognition using artificial neural networks*", pages 1-7. MGM's College of Engineering and Technology, 2005

[38] The PCman Website, retrieved 02/11/08, http://www.thepcmanwebsite.com/media/pacman1/pacman1.shtml

[39] Walt Scacchi, *Research and Educational Innovations in Computer Game*s, pages 2-4 California Institute for Telecommunications and Information Technology, 2002

[40] Nintendo, http://www.nintendo.com/wii/what/controllers, retrieved 30/10/08

[41] EyeToy, http://www.us.playstation.com/PS2/Games/EyeToy_Play/ogs/, retrieved 30/10/08

[42] Martin T. Hagan, Howard B. Demuth, and Mark H. Beale**.** **"***Neuron Model and Network Architectures"*, http://hagan.ecen.ceat.okstate.edu/2_Architectures.pdf

[43] Andrea Corradini, *"Dynamic Time Warping for Off-Line Recognition of a Small Gesture Vocabulary"*, IEEE Computer Society, 2001

Appendix A: MATLAB® scripts

This section contains MATLAB® scripts used in the training and simulation of the gestures recognisor

A1. Training sets

```
% Written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Diagonal right up gesture
% Polar Coordinates line 1
r1=0;
deg1=50*rand(1)+20;
%diff1=200*rand()+30;
we1=[];
for qw1=[0:1:4]
    r1=r1+200*rand()+30;%adds that extra required random length factor;
    y1=sin(deg1*pi/180)*r1;
    x1=cos(deg1*pi/180)*r1;
    we1=[we1; [x1 y1]];
   %plot(x,y)
end
we1;
% we1 is a matric containing the coordinates
```

```
% Written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Diagonal left down gesture
% Rectangular Coordinates line 2
% deg varies the slope of the line, simulating similar gestures
r2=1000;
deg2=50*rand(1)+200;
we2=[];
for qw2=[0:1:4]
    r2=r2-(200*rand()+30);
    y2=-sin(deg2*pi/180)*r2;
    x2=-cos(deg2*pi/180)*r2;
    we2=[we2; [x2 y2]];
 end
we2;
% we2 is a matric containing the coordinates
```

```
% Written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Diagonal right down gesture
% Rectangular Coordinates line 3
% deg varies the slope of the line, simulating similar gestures
r3=1000;
deg3=50*rand(1)+290;
we3=[];
for qw3=[0:1:4]
    r3=r3-(200*rand()+30);
    y3=-sin(deg3*pi/180)*r3;
    x3=-cos(deg3*pi/180)*r3;
    we3=[we3; [x3 y3]];
end
```

```
we3;
% we3 is a matric containing the coordinates


% Written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Diagonal right up gesture
% Rectangular Coordinates line 4
% deg varies the slope of the line, simulating similar gestures
r4=0;
deg4=50*rand(1)+110;
we4=[];
for qw4=[0:1:4]
   r4=r4+200*rand()+30;
   y4=sin(deg4*pi/180)*r4;
   x4=cos(deg4*pi/180)*r4;
   we4=[we4; [x4 y4]];
end
we4;
% we4 is a matric containing the coordinates


% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Right arrow gesture
% Coordinates line 5
P5=[];
x5=round(30*rand(1)+5);
s5=linspace(-600,600,x5);
         rn=40*rand(1,x5);
         P5=[P5 s5];
         P5=P5';
         w5=40*rand(1,x5)-40;
         w5=w5(:);
         P5=[P5 w5];
         P5;
% P5 contains the coordinates simulating line 5/right arrow gesture


% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Left arrow gesture
% Coordinates for line 6
r6=0;
P6=[];
for qw6=[0:1:4]
   r6=r6-(40+160*rand());%diff4;
   x6=r6;
   y6=80*rand()-40;
   P6=[P6; [x6 y6]];
end
P6;
% P6 contains the coordinates simulating line 5/left arrow gesture
% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Down gesture
% Coordinates for line 7
P7=[];
x7=round(7*rand(1)+5);
s7=linspace(-300,300,x7);
   P7=[P7 -s7];
   P7=P7';
   w7=20*rand(1,x7);
   w7=w7(:);
   P7=[w7 P7];
   f7=[0;0;0;0;0;0];
   P7;
% P7 contains coordinates for line7/down gesture


% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Up gesture
% Coordinates for line 8
P8=[];
x8=round(7*rand(1)+5);
s8=linspace(-300,300,x8);
   P8=[P8 s8];
   P8=P8';
   w8=20*rand(1,x8);
   w8=w8(:);
   P8=[w8 P8];
   P8;
 % P8 contains coordinates for line7/Up gesture


% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Clockwise gesture
% Coordinates for circle 1

c1=[];
ac1=round(5*rand(1))+5;
bc1=(12*rand(1));%random # from 0-12,ie 360/30=20
rc1=round(400*rand(1)+75);%defining radius range
t = linspace(2*pi,0,ac1);%random # of divisions for training(5-10)
h=0;
k=0;
xp = rc1*cos(t+bc1*pi/6)+h; %randomly select start off points
yp = rc1*sin(t+bc1*pi/6)+k;
c1=[c1 [xp; yp]];
c1=c1';%varies in size ,space btwn points and drawing origin

% c1 contains coordinates for circle1/clockwise circle gesture
% Script written by Ray Musvibe
% This script generates (x,y) coordinates for the
% Anti-Clockwise gesture
% Circle2 anticlockwise
c2=[];
ac2=round(5*rand(1))+5;%too much would be almost linear
bc2=(12*rand(1));%random # from 0-12,ie 360/30=20
rc2=round(400*rand(1)+75);
t = linspace(0,2*pi,ac2);
h=0;
k=0;
x1 = rc2*cos(t+bc2*pi/6)+h;
y1 = rc2*sin(t+bc2*pi/6)+k;
c2=[c2 [x1; y1]];
c2=c2';

% c2 contains coordinates for circle1/clockwise circle gesture
```

A2. Preprocessing script

```
% Script written by Ray Musvibe
% Vectorise (x,y) coordinates from polarline1/ can be used for the other gestures as well
polarline1;
data=we1;
data=data';
data=data(:,[1 2 3 4 5]);% constraining input to the first five coordinates
data=data';
a1=30*rand(1,5)-15;%add random component to handle noise
a1=a1(:);
b1=30*rand(1,5)-15;
b1=b1(:);
ca1=[a1 b1];%5*2 column of random numbers
data=data+ca1;
data=data';
r1=[];%holds vectors
l1=length(data);
k1=1;
  for p1=1:l1-1
       w1=data(:,k1);
       s1=data(:,k1+1);%
       t1=s1-w1;%subtracting consecutive columns to obtain a vector
       r1=[r1 t1];%augment the matrix
       k1=k1+1;
  end
 r1=r1(:);
 r1=r1'; %outputs the vectors , one less than input number
 data=r1/200;%scalling down
 data=data(:);

 %data contain vectors representing polarline1, the same for all other     %gestures
```

A3. Training script

```
% Written by RS Musvibe
% 29/4/8
% Matlab script that takes in preprocessed data and uses as it as input to ANN for training
net2=newff([-5 5;-5 5;-5 5;-5 5;-5 5;-5 5;-5 5;-5 5], [25 25 10],{ 'tansig'  'tansig' 'logsig' },'trainscg');
for kt=1:10000 % 10 000 iterations
        sortdata10;%has a random element added with each iteration for first gesture
        sortdata1;%import vectors from 1st gesture
        sortdata2;%2nd gesture...
        sortdata3;
        sortdata4;
        sortdata5;
        sortdata6;
        sortdata7;
        sortdata8;
        sortdata9;
        net2.trainParam.show = NaN; %speed up training by not plotting training process
        t1=[0 0 0 0 0 0 0 0 0 1];%targets
        t1=t1';
        t2=[0 0 0 0 0 0 0 0 1 0];
        t2=t2';
        t3=[0 0 0 0 0 0 0 1 0 0];
        t3=t3';
        t4=[0 0 0 0 0 0 1 0 0 0];
        t4=t4';
        t5=[0 0 0 0 0 1 0 0 0 0];
        t5=t5';
        t6=[0 0 0 0 1 0 0 0 0 0];
        t6=t6';
        t7=[0 0 0 1 0 0 0 0 0 0];
        t7=t7';
        t8=[0 0 1 0 0 0 0 0 0 0];
        t8=t8';
        t9=[0 1 0 0 0 0 0 0 0 0];
        t9=t9';
        t10=[1 0 0 0 0 0 0 0 0 0];
        t10=t10';
        T=[t1 ,t2 ,t3 ,t4 ,t5 ,t6 ,t7 ,t8 ,t9 ,t10];
        %Add inputs matrix
        input=[data ,data1 ,data2 ,data3 ,data4 ,data5 ,data6 ,data7 ,data8 ,data9];
        net2.trainParam.epochs = 200;
        net2.trainParam.goal = .0001;
        net2=train(net2,input,T);
end
%close all;
save Neurons.mat; % Save the result
```

A4. Simulation scripts (used for testing Matlab implementation)

```
% functions simulates the trained network
% RSM 30/4/8
importer; % grabs inputs from file
(round(sim(net2,data11)))
delete('D:\Documents and Settings\g08m3079\My Documents\MATLAB\MyTextFileTest1.txt');
```

```
% importer script
% Written by R S Musvibe
% Sortdata/Pre-process data from a file
% import data for simulation of Matlab Neural Net
data11=importdata('MyTextFileTest1.txt');
x3=length(data11);%This section of code samples the (x,y) coordinates 'evenly'
y3=floor(x3/5);
v3=data11([y3 y3*2 3*y3 4*y3 5*y3]);
v13=data11([x3+y3 x3+y3*2 x3+3*y3 x3+4*y3 x3+5*y3]);
data11=[v3' v13'];
data11=data11';
ri=[];%holds vectors
li=length(data11);
ki=1;
   for ptu=1:li-1
      wi=data11(:,ki);
      si=data11(:,ki+1);
      ti=si-wi;%subtracting consecutive columns to obtain a vector
      ri=[ri ti];%augment the matrix
      ki=ki+1;
   end
 ri=ri(:);
 ri=ri';
 %outputs the vectors , one less than input number
 data11=ri/100;
 data11=data11(:);
```

Appendix B: C# Code

This section contains C# code use for integrating the Neural based recognisor prototyped in MATLAB® into BingBee.

B1. The Neuron class

```
using System;
using System.Collections.Generic;
using System.Text;
using BingBee.Core;

namespace BingBee.Book
{
    [Serializable]
    public class Neuron
    {
        #region PROTECTED FIELDS (State variables)
        protected double[] w; //holds weight array
        protected double[] input; //array of inputs
        protected double threshold; //bias
        protected int N; //number of weights
        protected ActivationFunction f = null;
        // Value of the last neuron ouput
        protected double o;
        #endregion

        //#region PROTECTED FIELDS (State variables)
        public Neuron(double thresh, ActivationFunction af, int Ni)
        {
            w = new double[Ni];
            f = af;
            threshold = thresh;
            N = Ni;
        }
        public Neuron()
        {
            w = new double[8];
            f = new TanSigmoid();
            threshold = 0;
        }
        //#endregion

        #region PUBLIC METHODS (COMPUTE THE OUTPUT VALUE)
        // returns number of input synapses
        public int N_Inputs
        {
            get { return w.Length; }
        }
        public void setWeight(double[] a)
        {
            w = a;
        }
        public void setActivation()
        {
            f = new SigmoidActivationFunction();
        }
        public void setWeightNum()
        {
            w = new double[8];
        }
        public string tomyString()
        {
            string s = "\nThe Weghts are as follows\n ";
            int k = 0;
            for (k = 0; k < w.Length; k++) s += w[k].ToString() + "\n";
            return (f.ToString() + " is the Activation Function \n" + "The Threshold : " + threshold.ToString() + s);
        }
        // Compute the output of the neurone
        public double ComputeOutput(double[] input)
        {
            if (input.Length != N)
                throw new Exception("NEURON : Wrong input vector size. " + N.ToString() + " Required");
            double ws = new double();
            ws = 0;
            for (int i = 0; i < N; i++)
                ws += w[i] * input[i];//sums up weights and inputs...change ipout name
            ws -= threshold;//subtract the thresholh
            if (f != null)
                o = f.Output(ws);
            else
                o = ws;
            return o;//fires the output
        }
        #endregion
    }
}
```

B2. ActivationFunction interface

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BingBee.Book
{
   public interface ActivationFunction
   {
      double Output(double x);
   }
   #region LogSigmoid

   //for the output layer
   [Serializable]
   public class SigmoidActivationFunction : ActivationFunction
   {

      // Get the name of the activation function

      public string Name
      {
         get { return "Log Sigmoid"; }
      }

      public double Output(double x)
      {
         return (double)(1 / (1 + Math.Exp(-x)));
      }

   }
   #endregion LogSigmoid

   #region TanSigmoid

   //For the first layer and hidden layers

   [Serializable]

   public class TanSigmoid : ActivationFunction
   {

      public double Output(double n)
      {
         return (double)(2 / (1 + Math.Exp(-2 * n)) - 1);
      }
      public string Name
      {
         get { return "TanSig"; }
      }
   }
   #endregion TanSigmoid
}
```

B3. Classifier Class (calls the neuron constructor, instantiates the neural net, performs input sanitization and classifies input, returning the direction)

```
using System;
using System.Collections.Generic;
using System.Text;
using BingBee.Core;
using System.IO;

namespace BingBee.Book
{
   public class Classifier
   {
          public enum Direction { None, N, E, S, W, Rotate2 ,DoubleTap};
          int n;
          double[] points2 = new double[8];
          long timeDown;
          bool decentinput;
          bool classified = false;
          //int largestX = 0;
          //int largestY = 0;
          //double positiveroot;
          string path = @"D:\Documents and Settings\g08m3079\My
Documents\MATLAB\MyTextFileTest1.txt"; //for matlab simulation
          double[] points = new double[16];//holds initial points from messages before classification
          double[] Results1 = new double[8];//input into neural net
          double[] thresh1 = new double[25] { -2.051279621,2.001852342,-0.974504443,-1.145053705,-
0.568929463,0.121177165,-0.274908596,-0.06030077,-0.369015101,0.908433767,-0.297260357,-0.07100664,-
1.19957886,-0.168492324,0.586195711,0.304975398,0.322019748,0.007346143,-0.313086065,2.006670017,-
0.693348007,-1.787040907,1.578222005,1.969755139,1.991356555 };//25 threshold values
          double[] thresh2 = new double[25] { -2.043121698,-2.085151023,-1.613747397,-
1.420984364,1.180875052,-1.14725735,1.242071649,0.778528871,0.443364791,0.556787911,0.437774169,-
0.398541,-0.395154818,-0.260583585,-0.890618422,-
0.603528779,0.226949419,1.04513532,1.215511571,0.618629951,1.555930241,1.44609328,1.619309416,-
0.745154934,-1.568825597 };
          double[] thresh3 = new double[10] { -3.993388436,-2.564790716,0.095210652,-2.269235049,-
0.675450622,-1.13476461,-0.065134988,0.819667375,-3.413841493,2.055070488};//4 threshold values for
outputlayer
          Neuron[] Layer1 = new Neuron[25];//the first layer neuronsS
          Neuron[] Layer2 = new Neuron[25];//the 2nd layer neurons
          Neuron[] Layer3 = new Neuron[10];//the output layer neurons
          double[] OutputLayer1 = new double[25];//output from first layer
          double[] OutputLayer2 = new double[25];//output from 2nd layer
          int[] OutputLayer3 = new int[10];//output from output layer
          int[] Rotate2 = new int[10] { 0, 1,1,1,1,1,1,1,1,1 };
          int[] Down = new int[10] { 1,1,1, 0, 1,1,1,1,1,1 };
          int[] RightToLeft = new int[10] { 1,1,1,1,1, 0, 1,1,1,1};
          int[] LeftToRight = new int[10] { 1,1,1,1, 0, 1,1,1,1,1};
          int[] Rotate = new int[10] { 1,0,1,1,1,1,1,1,1,1};
          int[] err1 = new int[10] { 1,1,1,1,1,1,1,1,1,1 };
          int[] Up = new int[10] { 1,1,0,1,1,1,1,1,1,1};
          int[] err3 = new int[10] { 1, 1, 1, 1, 1, 1, 1, 1, 0, 1 }; //for diagonal gestures not used
          int[] err4 = new int[10] { 1, 1, 1, 1, 1, 1, 1, 1,1, 0 };
          int[] err5 = new int[10] { 1, 1,1, 1, 1, 1, 0, 1, 1, 1};
          int[] err6 = new int[10] { 1,1, 1, 1, 1, 1, 1, 0, 1, 1};
          //int[] err7 = new int[10] { 1,0, 0, 0, 0, 0, 0, 0, 0, 0};
          //int[] err8 = new int[10] { 1,0, 0, 0, 0, 0, 0, 0, 0, 0};
          //int[] err9 = new int[10] { 1,0, 0, 0, 0, 0, 0, 0, 0, 0,};
```

```
            //bool alreadyTriggered;
            long InterArrivalWindowTicks = 500 * TimeSpan.TicksPerMillisecond;
            long timeOfLastDown=0;
            bool isPressed;
            bool Initialised=true;
        public void init(){
                n=0;

                for (int k = 0; k < 25; k++)//instantiating each neuron for layer 1, 8 inputs
                {
                Layer1[k] = new Neuron(thresh1[k], new TanSigmoid(), 8);
                switch (k)
                {
                    case (0): Layer1[k].setWeight(new double[] { 0.223825563, 0.096038338, 0.04234459,
0.148619838, 0.00864996, -0.070325203, 0.168426247, 0.016703687 });
                        break;
                    case (1): Layer1[k].setWeight(new double[] { -0.124795271, -0.186919107, -0.067989483,
0.031619789, -0.046349558, -0.081790489, -0.146788155, -0.048794054 });
                        break;
                    case (2): Layer1[k].setWeight(new double[] { -1.475900953, -0.681635215, -1.187451003,
1.003945594, -0.757184836, 0.909343809, 0.211578995, 0.52107016 });
                        break;
                    case (3): Layer1[k].setWeight(new double[] { 1.04716269, -0.163194355, 0.192635713, -
1.234088427, -0.82026418, -1.251202164, -1.116375355, 0.395348004 });
                        break;
                    case (4): Layer1[k].setWeight(new double[] { 1.411870952, -0.822519897, 1.112854641, -
0.22534622, 0.457467083, 0.873927175, -0.152072524, 0.664561032 });
                        break;
                    case (5): Layer1[k].setWeight(new double[] { -0.811249643, 0.305063267, -1.656841714, -
0.300294947, -1.416861498, 0.088478148, -0.227693035, -0.006896374 });
                        break;
                    case (6): Layer1[k].setWeight(new double[] { 0.088236443, -2.133099497, -0.448972258, -
1.328726918, -1.446855924, 0.280950335, -2.330663959, 1.397182355 });
                        break;
                    case (7): Layer1[k].setWeight(new double[] { 0.567313234, 1.373547298, 1.145388454,
0.523922216, 0.016987521, 0.367223281, -1.157722518, 1.127204113 });
                        break;
                    case (8): Layer1[k].setWeight(new double[] { 3.027260651, 0.672762912, 1.31230223,
0.321917074, 0.375298296, -0.912625979, -0.221604193, -0.677868165 });
                        break;
                    case (9): Layer1[k].setWeight(new double[] { -0.328990098, -0.726789929, -0.638159856, -
1.161562285, -0.917975588, -0.500432176, -1.547311166, 0.771101855 });
                        break;
                    case (10): Layer1[k].setWeight(new double[] { -1.066436194, -0.356566867, -2.889893039,
0.148171954, -2.894403246, 0.252535603, -1.684239359, 0.640992152 });
                        break;
                    case (11): Layer1[k].setWeight(new double[] { 1.753475823, -0.283813725, 1.336781918, -
0.897777219, 0.336568085, -2.289673622, -0.988300792, -3.454984507 });
                        break;
                    case (12): Layer1[k].setWeight(new double[] { 1.456152547, -0.771389254, 0.738741844, -
1.232970749, 0.316142542, 0.307167396, -0.72013601, 1.196201375 });
                        break;
                    case (13): Layer1[k].setWeight(new double[] { -0.832904558, -1.980783643, 0.101236488, -
1.463621275, 0.454911023, -0.6851666, 1.169720504, 0.790033962 });
                        break;
                    case (14): Layer1[k].setWeight(new double[] { 0.569832329, -0.616756123, 0.36474505, -
1.709385867, 0.852630526, -1.264759933, 1.966056398, 0.512375529 });
                        break;
                    case (15): Layer1[k].setWeight(new double[] { 1.365737342, -1.624251916, 0.572335644, -
1.250727293, -0.808120516, -0.593772006, -2.420198033, -1.015669835 });
```

```
                    break;
                case (16): Layer1[k].setWeight(new double[] { 1.176879095, 0.144000288, 0.919133051,
1.789268772, 0.228809814, 3.258560615, -0.188128834, 2.607328045 });
                    break;
                case (17): Layer1[k].setWeight(new double[] { -1.65907038, 1.639237091, -1.027860322,
0.157662857, -0.309649963, -1.238797476, -1.376708526, -2.536615373 });
                    break;
                case (18): Layer1[k].setWeight(new double[] { 1.584581998, 1.535163706, 0.380383013,
1.517404167, -1.169073764, 1.018915888, -2.23601214, 0.571046992 });
                    break;
                case (19): Layer1[k].setWeight(new double[] { 0.287552418, 0.032989501, -0.413946828, -
0.016336508, 0.570028062, 0.168124549, -0.297919008, 0.274730751 });
                    break;
                case (20): Layer1[k].setWeight(new double[] { -0.076312975, 1.122967732, -0.309639715,
1.487345709, -0.270549217, 0.033361551, -0.083547764, -0.877843895 });
                    break;
                case (21): Layer1[k].setWeight(new double[] { -0.102937604, 0.103295579, -0.067267488,
0.036310146, -0.15722179, -0.156010827, -0.015860494, -0.020057941 });
                    break;
                case (22): Layer1[k].setWeight(new double[] { -0.310310668, -0.155974324, -0.215092119, -
0.039063395, 0.078979484, -0.026204081, -0.444639796, -0.043092237 });
                    break;
                case (23): Layer1[k].setWeight(new double[] { 0.030869365, -0.09440003, 0.003892177, -
0.11082302, 0.003896548, 0.084192117, 0.009784272, -0.111316145 });
                    break;
                case (24): Layer1[k].setWeight(new double[] { 0.078636099, 0.123551358, 0.258441081,
0.016404474, -0.101227594, 0.200522153, 0.200177003, -0.045255966 });

                    break;

            }
        }
        for (int k = 0; k < 25; k++)//instantiating each neuron for layer 2, 25 inputs, Tansig activation function
        {
            Layer2[k] = new Neuron(thresh2[k], new TanSigmoid(), 25);
            switch (k)
            {
                case (0): Layer2[k].setWeight(new double[] { 0.438471085, -0.436273026, 0.894297795, -
0.10687904, -1.029684532, -0.370784951, 0.101448186, -0.166737572, 0.027065834, -0.280308394, -
1.626207413, 0.095013246, -0.414813616, -0.625555743, -0.079635496, -0.791761211, -0.27174594,
0.766730915, -0.28689151, -1.046355739, 0.989268537, -0.009830758, 0.246491231, -0.545221721, -
0.367662011 });
                    break;
                case (1): Layer2[k].setWeight(new double[] { 0.822681603, -1.153999363, 0.243810247,
1.072045647, -0.165404869, 0.47037717, 0.805932708, 0.322850955, -0.184084071, 0.205993147,
0.492512156, 0.457829963, -0.184985546, 0.605969828, -0.182503162, 0.59147788, -0.189331042,
1.301527459, -0.759838497, -1.861365831, 0.682747976, 0.541619104, -0.647349509, -0.754531034,
0.021095906 });
                    break;
                case (2): Layer2[k].setWeight(new double[] { 0.718187658, 0.081512388, -0.244072551, -
0.815599983, -0.081449667, 0.159804269, -0.473923415, 0.5893641, -0.087118139, 0.418751739,
1.38258178, -2.377771991, -0.146563706, -0.617759459, -0.606362582, -0.924809286, -0.679202667,
0.543975506, 1.805468874, -0.515910479, 0.183378571, 0.504500938, 0.080218547, -0.186692931, -
0.013439968 });
                    break;
                case (3): Layer2[k].setWeight(new double[] { 0.448085601, -0.663935901, -0.194349271,
0.116662701, 0.836200395, -0.58941108, -0.834374286, 0.469991811, -0.304713238, -0.05136612,
0.530274131, -0.544042112, -0.039818059, -0.980797194, -0.475171416, -0.922806147, 1.096009211,
0.148717384, 1.814123845, 0.148986431, -0.424184295, -0.275587352, 0.037910552, -0.567898885,
0.289575655 });
```

```
                    break;
                case (4): Layer2[k].setWeight(new double[] { -0.196609507, 0.040647744, -1.324026778, -
1.165491532, 0.816831696, -0.42774803, -1.599338558, 0.606941415, 0.100387598, -0.002670366, -
1.940618512, 0.85253692, -0.263548401, 0.122325237, 0.279556048, -0.085300399, -0.276665343,
0.108147319, 0.178156727, -0.460057099, 1.098294159, -0.253167313, 0.488318594, -0.255070762, -
0.198626844 });
                    break;
                case (5): Layer2[k].setWeight(new double[] { 0.569541431, 0.136627981, 0.170528822,
0.294537316, 0.331567221, 0.420474381, -0.223593657, 0.077703377, -0.099000701, 0.160159343, -
0.07334376, -0.334996235, 0.252282262, 0.059500394, 0.293862372, 0.241497898, 0.390901141,
0.302299206, 0.10627943, -0.70608349, 0.210279951, 0.568853317, -0.567855096, -0.646729297, -
0.165502973 });
                    break;
                case (6): Layer2[k].setWeight(new double[] { -0.827105523, 0.328322528, 0.437616889,
0.528763048, 0.74469054, -0.736276544, -0.601286132, 0.253185307, 1.44858533, -0.57219494, -0.17469546,
-1.342254304, 1.362705014, 0.416496355, 0.121397932, 0.716748633, 0.58616136, -1.124839941,
0.548274683, 0.347991263, 0.220793945, -0.940756843, 0.350328076, 0.248555152, -0.04378597 });
                    break;
                case (7): Layer2[k].setWeight(new double[] { -0.606749211, 0.206010371, -0.207990826, -
0.709154266, -0.300306784, -0.510292591, -1.477633613, 0.601698667, -1.13322056, 0.644999966, -
0.678862536, -1.660643284, 0.307795665, -0.740162762, -0.426231609, -2.142735604, 1.581486714,
0.741230219, 1.021946819, -0.180888826, 0.657380949, 0.173506863, 0.291126992, 0.431311965,
0.458737146 });
                    break;
                case (8): Layer2[k].setWeight(new double[] { -0.373490816, 0.276085639, 0.878675006,
0.40602628, -1.021683439, 0.294306425, -0.566051472, -0.041411707, -1.104246914, 0.979960845,
0.741486635, 1.49891259, -0.572612145, -0.77420927, -0.774601432, 0.408453201, 1.245250902,
0.850737519, -1.111039123, 0.36845549, 0.283733201, 0.673962801, 0.409242411, -0.372739885,
0.031985842 });
                    break;
                case (9): Layer2[k].setWeight(new double[] { -0.56125762, 0.621161305, 0.171901053, -
0.428070209, -0.458197758, -0.177778694, -0.210043783, 0.364453762, 0.092188147, -0.342784032,
0.221378602, 0.330686552, -0.209161755, -0.225310426, 0.056575701, -0.272015106, -0.076541284, -
0.110888378, 0.128748634, 0.070541876, -0.512565282, -0.401559289, 0.54308824, 0.443953841,
0.582475337 });
                    break;
                case (10): Layer2[k].setWeight(new double[] { -0.354493804, 0.633760318, -0.283333982,
0.042519386, -0.106229137, -0.037297551, -0.348006798, -0.201358902, -0.219973494, 0.059143885,
0.222487984, -0.270199161, -0.058396837, 0.249324532, 0.362370063, 0.555524805, 0.320161754, -
0.2089552, 0.466322026, 0.054842759, 0.108144337, -0.544677049, 0.587412785, 0.368123471, 0.695740082
});
                    break;
                case (11): Layer2[k].setWeight(new double[] { 0.247747772, -0.907836282, 0.350555901,
0.431965229, 0.222763047, 0.292918096, -0.239089871, -0.277153555, -0.368337289, -0.268886352,
0.00206285, -0.101440043, 0.453062363, 0.45910698, -0.214653143, 0.297383669, -0.169115827,
0.170094552, 0.54543801, -0.829970666, 0.49014636, 0.727432183, -0.363283681, -0.082111524, 0.04226685
});
                    break;
                case (12): Layer2[k].setWeight(new double[] {-0.204121565,-0.651256459,0.156119678,-
0.153614267,0.43760272,-0.316567773,0.331057103,0.355724259,-0.099601327,-0.517949279,0.523045775,-
0.317404975,0.176999055,0.322864822,-0.717023039,-0.148222754,-0.560580564,0.095717864,-
0.01302365,-0.70239556,-0.098168025,0.826097067,-0.266033495,0.006657712,-0.49887397 });
                    break;
                case (13): Layer2[k].setWeight(new double[] { 0.395841001, 0.008250053, 0.763481068, -
0.11421465, -0.565011619, 0.316627341, 1.543312774, -0.003361956, -1.112103176, 0.013823431,
0.223732438, -1.728399377, 0.036932139, -0.694413452, -0.692424908, 0.643591079, -1.304144625,
0.334775942, 0.580985051, -0.529651521, -0.175605382, 0.426652555, -0.011661239, -0.560969406, -
0.814342269 });
                    break;
```

```
            case (14): Layer2[k].setWeight(new double[] { 0.208287996, -0.511363164, -0.561297369, -
0.328699329, -0.440073639, 0.034275499, -0.991408912, 0.766790717, 0.226039331, 0.497189343, -
1.443143661, 0.68445973, -0.920994699, -0.956449144, 1.31317784, -1.392579395, 2.433499195, -
1.039161448, 0.044143707, -0.369081004, 0.653773202, 0.873353698, -0.990431653, -0.793769376, -
0.527784436 });
                break;
            case (15): Layer2[k].setWeight(new double[] { -0.201036468, -0.184055389, -0.414304719, -
0.880744631, 0.695009505, -1.430757769, -0.997331386, -0.441133232, 1.141442589, -0.111150315, -
0.129933164, -0.260929274, 0.605680636, 1.574465549, -0.419741126, 0.350577949, 0.389503905, -
1.614414952, -0.003140918, -0.669170268, -0.260966791, -0.199857064, 0.215639845, -0.171742549, -
0.561825832 });
                break;
            case (16): Layer2[k].setWeight(new double[] { 0.46251183, -0.134689007, 1.205125964,
0.731759818, -0.661474212, 0.080379704, -0.338000774, 0.019401042, -0.130248583, -0.576469091,
0.843922773, -1.984981156, 0.015350311, -0.580061166, -0.052365292, -2.213702537, 2.313110999, -
0.683107897, 0.516212985, 0.056838412, -0.1769518, 0.733851392, -0.444098226, -0.551893411, -
0.67163397 });
                break;
            case (17): Layer2[k].setWeight(new double[] { -0.02226647, 0.664581204, 0.11329233, -
0.003426421, 0.120703104, -0.156548152, -0.278127277, 0.55448878, 0.228442936, 0.326364172,
0.189650713, 0.346777676, -0.504275538, 0.261368491, 0.303513252, -0.318244508, 0.430595813, -
0.252508183, -0.266928096, 0.59864517, -0.445064689, -0.484160015, -0.085448769, 0.671277616,
0.202226904 });
                break;
            case (18): Layer2[k].setWeight(new double[] { 0.083565026, 0.421504844, 0.242330834,
0.116599889, -0.046619581, -0.035576721, -0.24919591, 1.427407046, -0.269503418, 0.215158226,
0.821435787, -0.500605894, 0.95408477, -0.663232623, -0.22738177, -1.203291246, 0.856621325,
0.873064644, 0.560201906, 0.636694671, -0.198084311, -0.368835632, 0.598829094, 0.66504759, -
0.027245772 });
                break;
            case (19): Layer2[k].setWeight(new double[] { 0.443957878, -0.715401973, 0.794746259,
0.780208407, 0.008667494, -0.807288397, -1.264451384, 0.045037289, 0.827079516, -1.437765628, -
2.438006371, 1.177905172, -0.255904349, -0.171254034, 0.583159701, 0.143161055, 1.170644224,
0.046924397, -0.779204787, -0.368877919, 0.824870111, -0.176685374, -0.732630215, -0.219084055,
0.086582636 });
                break;
            case (20): Layer2[k].setWeight(new double[] { 0.049881418, 0.419258471, -0.603802049,
1.068251501, 0.67326185, -0.6629349, -0.889769019, -0.232978592, 0.409970524, 0.067647275, -
2.087727671, -0.029502779, 0.81026397, 0.293624971, 1.301420301, -0.186527341, 1.71374753, -
1.974580445, -0.966138314, -0.008360011, -0.710530434, -0.111366694, 0.086378956, 0.269914261,
0.795987894 });
                break;
            case (21): Layer2[k].setWeight(new double[] { -0.206281161, 0.184638702, -0.19485691, -
0.42511687, 0.006531259, -0.469275492, 0.084423627, 0.994644954, -1.240276664, -0.480342062, -
0.940660039, -0.807750561, 0.237065537, 0.587372509, 0.205252623, -0.250247309, -1.017681542, -
0.447453581, 0.17176192, -0.274103692, -0.667337764, 0.018492814, 0.366665879, 0.445872954,
0.265051576 });
                break;
            case (22): Layer2[k].setWeight(new double[] { -0.262443234, 0.742233808, 0.727586296,
0.652995189, -0.705109552, 0.319451882, -0.294860143, 0.637656842, 0.533952281, 0.393314714,
2.642095117, -0.637414868, 0.72159997, -1.336465062, -0.429268228, -0.162062199, -0.549093362,
1.565768733, 0.159299592, 0.430365849, 1.040540814, -0.077708335, 0.054772889, -0.016183323,
0.090926686 });
                break;
            case (23): Layer1[k].setWeight(new double[] { -0.845040774, 0.80577241, -0.068204269, -
0.752065353, 0.779074509, -0.172897064, -2.182546107, 0.708615219, 1.277605684, -0.047206857, -
0.314338583, -0.121661663, 0.451961516, -0.033293716, 0.398232057, -1.567708262, 1.754483065, -
0.211054892, 1.126809538, 0.59541346, 0.418894774, -0.381574218, 0.470076672, 0.61377033, 1.064073064
});
                break;
                case (24): Layer2[k].setWeight(new double[] { -0.132863387, 0.272572172, -0.976103717,
0.724771412, -0.159985227, 0.878983804, 0.329121461, -1.199284943, -0.015689726, 1.130812416, -
0.172728707, 2.045655352, 0.224756003, 1.22829352, 1.337759652, 1.232771373, 0.132618641, -
0.973996195, -1.610337397, 0.021775094, -0.824287954, -0.310230156, 0.013897892, -0.102591148,
0.01770426 });
                    break;
            }
        }
        for (int k = 0; k < 10; k++)//instantiating each neuron for layer 3, 25 inputs, Sigmoid activation function
        {

            Layer3[k] = new Neuron(thresh3[k], new SigmoidActivationFunction(), 25);
            switch (k)
            {
                case (0): Layer3[k].setWeight(new double[] { 0.411672049, -0.251587702, 1.90160852, -
0.578192733, 2.444197303, 0.55747988, -0.763075709, 4.471591252, -1.102530258, -0.105841965, -
1.307245757, 1.758895612, 1.193230849, -1.457024271, -5.180232214, 1.35446102, -2.472155019, -
0.692672058, 1.210255541, -3.152537718, -0.118185085, -4.611106898, -1.003687814, 1.813132506,
1.517368913 });
                    break;
                case (1): Layer3[k].setWeight(new double[] { 4.013300611, 3.209033002, -2.460855545, -
2.62025627, -2.463521979, -0.376002153, 2.686293276, -0.151976016, 0.084257551, 0.52592128, -
0.237044567, 0.2888329, 0.722459666, -1.072373047, -1.634519164, -2.285711964, 2.094178357, -
0.255559817, 1.675494421, 1.841159743, 2.413193151, -0.190900365, 2.576765581, 0.52063487,
0.438733724 });
                    break;
                case (2): Layer3[k].setWeight(new double[] { -0.309299437, 1.52665324, 1.67296603,
0.771804596, 1.821091431, 1.167429381, 1.95097226, -0.906896516, 0.233567726, -1.720007314, -
1.921525029, 2.180620951, 0.640046887, 0.941703757, 2.926048668, -1.422800604, 1.058068278, -
2.460976167, -2.049451182, -3.854592099, 2.589722393, -0.671667416, 3.433265605, 0.640850193, -
0.037139843 });
                    break;
                case (3): Layer3[k].setWeight(new double[] { 1.262306043, 1.959438831, -0.822716157,
0.906889682, 2.664830605, 1.449925886, -1.698858467, -1.390693572, 0.250681855, -0.487028947, -
0.440455103, 0.797849803, 0.752344495, 1.145959413, 0.566905328, -2.830850227, -0.933547446, -
1.110345904, -1.788006161, -1.829342441, 2.594401313, -0.486637122, -1.61683089, -3.558553265, -
0.627793927 });
                    break;
                case (4): Layer3[k].setWeight(new double[] { 0.499785559, 0.705629336, 1.83834583, -
0.749109894, -0.995909285, 1.691429675, -0.243833821, 2.125679048, -0.584262788, -1.759443042, -
0.000245108, 1.112734078, 2.407868261, 3.202883797, -0.487773372, -0.677961339, -2.196054786, -
1.27652502, -0.734507078, -0.357387733, -1.210473677, 0.338763064, -1.509750048, -3.179264399, -
2.061084949 });
                    break;
                case (5): Layer3[k].setWeight(new double[] { 0.29203351, 1.771988325, -2.695812995, -
0.061665049, 0.039539218, 1.006340229, -0.443653072, 2.24371242, 2.181814392, -1.233501704, -
2.084139817, 2.070446289, 1.064061949, 1.713353804, 4.52555874, 1.243500862, -2.877691572, -
2.109911934, 0.664995693, 1.783518555, 0.823562667, -1.361021811, -1.606163609, 0.816247514,
4.064539368 });
                    break;
                case (6): Layer3[k].setWeight(new double[] { 1.002070682, 1.8298545, 0.662999193,
3.129248504, -1.280608493, 0.818159511, -0.950478469, 0.212037526, -0.850461616, -1.619627419, -
1.671247118, 1.190248977, 1.080953916, 1.049556164, -1.40389815, -0.566483721, 3.404233547, -
1.097921589, -0.317124196, -0.64199143, -3.524635028, 1.313631633, 0.336441244, 2.847541891, -
0.301902261 });
                    break;
                case (7): Layer3[k].setWeight(new double[] { 1.068705057, 0.182257989, -1.33290029,
1.230177388, 0.599741016, 1.721589045, 1.453919777, -4.012625222, -0.832410304, -1.009740078, -
0.390142522, 2.078646007, 1.810519042, 0.908408026, -2.294509199, 2.406199761, -0.92087747, -
0.982987453, -3.333319308, 3.056288572, -0.798792596, -0.326126922, -0.062947163, 1.007385445,
0.588547384 });
                        break;
                    case (8): Layer3[k].setWeight(new double[] { 0.125016769, 0.137343055, -2.043351783,
1.707405761, -2.520783957, 1.270682623, -1.885679185, -2.78223711, 0.433227066, -0.213456585, -
1.284498873, 1.511475896, 1.855126314, -0.593875091, -0.522330084, 0.545052139, -0.264215841, -
0.831983818, -0.581589367, -0.268329536, -3.759311511, 0.464124331, 1.900725494, -1.6657732,
1.877554749 });
                        break;
                    case (9): Layer3[k].setWeight(new double[] { 3.057964271, -0.052570934, 2.471234669,
1.445124961, -0.292643223, 1.914738362, 1.972276262, 0.981849951, -3.769404411, -1.567690127, -
0.110087215, 0.92185617, 1.194015452, 1.704208281, 1.234274519, 1.485708218, 2.069632577, -
0.400391722, -1.361631884, 2.655589523, 1.238128283, -0.70599401, -2.80477845, -0.611454626, -
2.443099529 });
                        break;

                }
            }
        }
    public int[] Outputs(double[] a)
    {
        double[] OutputLayer1 = new double[25];
        double[] OutputLayer2 = new double[25];
        int[] OutputLayer3 = new int[10];
        for (int k = 0; k < 25; k++)
        {
            OutputLayer1[k] = Layer1[k].ComputeOutput(a);
        }
        for (int k = 0; k < 25; k++)
        {
            OutputLayer2[k] = Layer2[k].ComputeOutput(OutputLayer1);
        }
        for (int k = 0; k < 10; k++)
        {
            if (Layer3[k].ComputeOutput(OutputLayer2) >= 0.5) OutputLayer3[k] = 1;//Post Processing...
            else OutputLayer3[k] = 0;
        }
        return OutputLayer3;
    }

    public void Reset()
    {
       n = 0;
       decentinput = false;
       classified = false;
       //Initialised = false;
       //alreadyTriggered = false;
       //timeOfLastDown = 0;
    }

    public Classifier()
    {
        Reset();
    }

    public bool isEqual(int[] a, int[] b)//for comparing arrays
    {
        bool res = true;
        for (int k = 0; k < 10; k++)
        {
            if (a[k] != b[k]) res = false;
        }
        return res;
    }

    public Direction pushGestureTickle(TickleMessage msg)//I used arrays
    {
        Direction result = Direction.None;
        switch (msg.KeyEventType)
        {
            case TickleMessageEventType.KeyDown:
                //alreadyTriggered = false;
                timeDown = DateTime.Now.Ticks;
                long prevDownTime = timeOfLastDown;
                timeOfLastDown = DateTime.Now.Ticks;
                if (timeOfLastDown - prevDownTime < InterArrivalWindowTicks)
                {
                  //return Direction.DoubleTap;
                }
                if (Initialised)
                {
                    init();//initialise the net if not already intialised
                }
                Reset();
                isPressed = true;
                break;
            case TickleMessageEventType.Moved:
                Console.WriteLine(msg.DX.ToString() + " and " + msg.DY.ToString() + "   Delta. KeyMoved");
                int delta1=(int) Math.Sqrt(msg.DX*msg.DX);
                int delta2=(int) Math.Sqrt(msg.DY*msg.DY);

                if (delta1 >= 35 || delta2 >= 35) decentinput = true;//input sanitization
                if (delta1 >= 400 || delta2 >= 400) decentinput = false;


                if (decentinput && n <= 10)
                {
                    points[n] = msg.DX; n++;
                    points[n] = msg.DY; n++;
                    Console.WriteLine(n.ToString() + " inputs added so far");
                }

                if (n == 10 && isPressed) {
                        Initialised = false;
                        using (StreamWriter SW = File.AppendText(path))
                        {
                            for (int k = 0; k < 10; k += 2)
                                SW.WriteLine(points[k] + " " + points[k + 1] + " ");
                            SW.Close();
                        }//FOR testing WITH matlab

                        for (int k = 0; k < 8; k++)
                        {
                            points2[k] = points[k+2] / 200;//clipping off some vectors
                         }

                        int []final=new int[10];
                        final = Outputs(points2);
                        //for (int k = 0; k < 10; k++) Console.WriteLine(final[k].ToString());
                        if (isEqual(final, Rotate)) { result = Direction.N; classified = true; }
                    if (isEqual(final, Down)) { result = Direction.S; classified = true; }
                    if (isEqual(final, LeftToRight)) { result = Direction.W; classified = true; }
                    if (isEqual(final, RightToLeft)) { result = Direction.E; classified = true; }
                    if (isEqual(final, Rotate2)) { result = Direction.Rotate2; classified = true; }
                    if (isEqual(final, err1)) { result = Direction.N; classified = true; }
                    if (isEqual(final, Up)) { result = Direction.N; classified = true; }
                    if (isEqual(final, err3)) { result = Direction.N; classified = true; }
                    if (isEqual(final, err4)) { result = Direction.Rotate2; classified = true; }
                    if (isEqual(final, err5)) { result = Direction.N; classified = true; }
                    if (isEqual(final, err6)) { result = Direction.Rotate2; classified = true; }
                    //if (isEqual(final, err7)) { result = Direction.Rotate2; Console.WriteLine(" Rotate2 Err"); } for
diagonal vectors not used here
                    //if (isEqual(final, err8)) { result = Direction.N; Console.WriteLine(" North Err"); }
                    //if (isEqual(final, err9)) { result = Direction.N; Console.WriteLine(" North Err"); }
                    if (!classified)
                    {
                        //Produce deep sound to alert user that gesture could not be matched
                    }
                    Reset();
                    isPressed = false;
                    //long timeDown2 = DateTime.Now.Ticks - timeDown; For evaluating interaction speed
                    //Console.WriteLine(timeDown2.ToString());
                }
                decentinput = false;

                break;
                case TickleMessageEventType.KeyUp:
                Reset();
                break;
                case TickleMessageEventType.Other: break;
                }

                return result;
        }
    }
}
```