Automated Adaptation of Input and Output Data for a Weightless Artificial Neural Network

Ben McElroy, Gareth Howells

School of Engineering and Digital Arts, University of Kent
bm208@kent.ac.uk W.G.J.Howells@kent.ac.uk

Abstract

The ability to adapt automated guided vehicles for employment to a range of practical situations can significantly enhance their usability in hazardous situations where security is a major concern and it is inadvisable for humans to enter. Robot guidance is still a very challenging issue computationally in both the academic and industrial worlds. Whilst considerable progress has been made in robotics in the last few decades, many still experience difficulties in the recognition of dynamically changing situations such as our daily environments. With so many different scenarios it is difficult to find one system that can effectively deal with both the expected and unexpected issues that may arise. This paper examines the possibility of manipulating the potential inputs and outputs to a system to tailor a better solution to the current problem. Weightless neural networks will be used as a classification tool to determine the direction of a robot in an open loop simulation.

Keywords: *robot guidance, weightless neural networks*

1. Introduction

Finding the best architectural configuration for an artificial neural network (ANN) is difficult even when the ideal number of output classes are known. However, when there are multiple possible output sets, the problem is increased exponentially. This paper introduces a method for allowing robot guidance software to modify the number of its outputs to best suit its needs given its environment. The paper builds on previous initial work [1][2][3] by adding the ability to modify the number of input and output classes and will discuss the method, the reasoning behind this and how it will be implemented. Although ANN's have been successfully applied to many fields, their application in real world situations is still a challenge.

Adaptive systems offer the ability to learn and generalize from a set of known examples allowing them to recognize previously unseen inputs based on their similarity of characteristics with previously seen examples. However, not all of these may be useful to the system overall or may be to similar to another class. It would thus be advantageous if an automated way of adding and removing input/output classes for use with the network could be devised. Weightless Neural networks (WsNNs) are a class of ANN’s that may easily be conFigured for expressing the solution to problems that may easily be expressed by simple Boolean logic [4]. This type of network will be employed to demonstrate the potential of an automated system for modifying output classes. The data to be used is collected from a robot possessing a set of ultrasonic sensors. This data is then parsed and several training sets created, each with a different number of output classes. These classes represent changes in direction. To effect the desired output, a meta-network [5][6] is used which analyses and improves various attributes of the WsNN using a Genetic Algorithm (GA). It uses the theory of the Evolutionary Artificial neural networks (EANNs) which are a specific type of ANN whereby evolution is used as another form of adaptation which supplements learning [7][8]. This will be explained in detail in a later section.

2. Ultrasonic Sensors

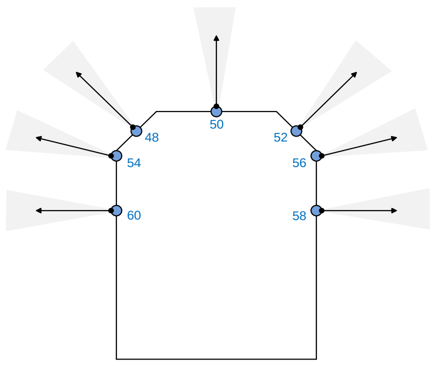


Figure 1 - The Robot Sensor Setup

A robot equipped with seven ultrasonic sensors was used in the current investigation. Although additional sensors could be required for a more practical setup, this composition was used with the intention of assessing the network configuration as it created a balance between complexity of the network and utility of the data. The sensors were adorned on the robot in a fashion shown in Figure 1. The sensors each have specific IDs used to identify them in the output data. The triangular grey regions in Figure 1 illustrates the approximate cone that the sensors operate within. The maximum range of the sensors is approximately 4 metres, and they return the distance to the closest object within their detection range. The sensors are set to return data every half a second with a distance given in millimetres.

3. Weightless Networks

WsNNs, or RAM-based neural networks store knowledge in Random Access Memories (RAM) within the networks nodes (neurons) rather than in the connections themselves [9]. These neuron have binary input values and make use of RAM effectively as lookup tables: the synapses (connections of the neuron) of each neuron collect a vector of binary bits from the network's input. This is then used as a RAM address where the vector/value stored is the neuron's output. One shot training can be attained whereby the system stores the desired output in the address associated with the input vector of the neuron.

There are several variations of WsNN [10]. For this investigation the Generalised Convergent Network will be employed [11]. It utilizes several layers which operate independently on a given sample pattern. The outputs from these layers are then merged to create an output matrix which is the same size as the original sample pattern. Illustrated below is the GCN architecture.

- The network is built out of a group of layers, whereby each of these layers have a set number of neurons. The number of neurons in each layer is defined by the size of the network input. That is, if this matrix is x by y, then the neurons are arranged in an x by y matrix as well.

- The Pattern elements are connected to a particular neuron within each of the layers. These elements (determined by a connectivity pattern – the collective relative locations of the neurons within a given layer) are small sections of the code matrix.
- The layers are put into one of two groups of layers - the 'Pre' group or the 'Main' group (see Figure 2)
- The outputs of the component layers of the group are then combined via a Merge layer. The Merge operation is performed on the corresponding neurons from each layer within the group and for each position within the layer. Therefore, the number of layers within a particular group is tied directly to the neural connectivity of the merge layer.
- The unaltered output of this merge layer is subsequently delivered back into the inputs of each of the layers in the group.
- The connectivity of the neurons and number of layers within each group is typically defined arbitrarily when the network is initially conFigured and is governed by the performance requirements for the network. However, in this system, these are set by the meta network.
- The constituent layers of a group differ in the selection of elements attached to the inputs of their constituent neurons (termed the connectivity pattern).
- Each of the neurons in a single layer is connected in the same way relative to their location within the parsed code matrix (and hence all have the same connectivity).

A full description of this architecture can be found in [4].

During the training phase of each network, layers are adjusted independently of each other. All the RAM-based architectures follow the above abstract pattern, making changes to the number of layers and groups as required. They differ however in the structure of the neurons employed within the constituent layers of the architecture. Although they all employ RAM-based neurons, the number of symbols employed by the neurons differs between the architectures.

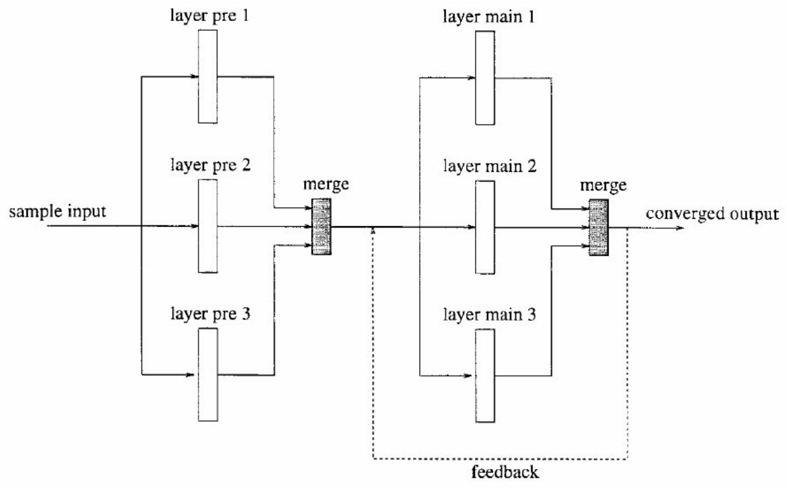


Figure 2 - The Two Groups of Layers (pre and main) in the GCN Network Architecture

4. The Input

As stated previously, the example problem domain is data gathered from the ultrasonic sensors attached to a robot as described in section 2. This data is collected as shown in the table 1 using example values.

Table 1 - Example Data Collected from the Robot

Sensor ID	Distance (mm)
48	1982
50	2967
52	3001
54	4371
56	2489
58	1001
60	443

The sensor ID from Figure 1 is linked to the ID shown in Table 1. For these experiments, direction and collision data is also included, so a parsing method is required.. Each sensor reading, direction and obstacle data must be encoded into a matrix. Firstly, the readings from the sensors were quantized so that they could be binarised more easily. These quantized values were then binarised using Gray coding [12].

Binary	Gray Code
31: 0 1 1 1 1 1	31: 0 1 0 0 0 0
32: 1 0 0 0 0 0	32: 1 1 0 0 0 0

Figure 3 - The difference between Gray Code and normal Binary

Regular binary suffers from large changes at certain values, as shown in Figure 3. Gray code removes this weakness and provides a much smoother change in bits from one value to the next. The direction (where the goal is compared to where the robot is headed) is also encoded using this method in such a way that similar values are represented by similar codes, as it is this similarity that the network is searching for. The obstacle data is triggered when the direction of travel is different to the direction of the goal, indicating that there is an obstacle (or several) in between the robot and a goal it wishes to reach. Should there be no obstacles between the robot and this goal, it will be represented by a zero code.

Once all the data had been generated, it was placed into a matrix as shown in Figure 4. Originally additional obstacle data was to be encoded in the grayed out areas however this was removed before experimentation began.

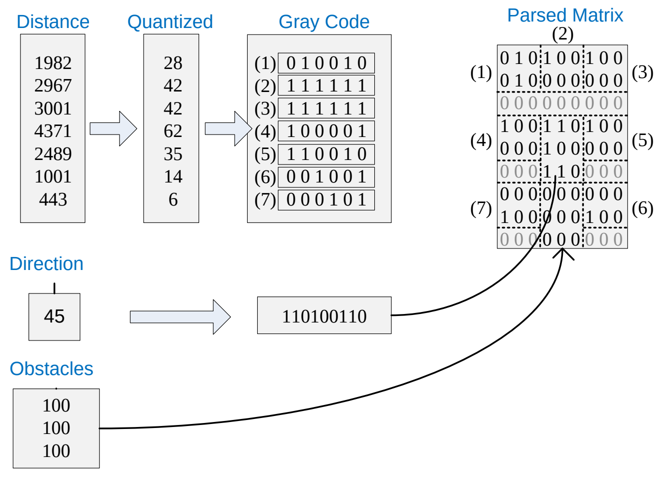


Figure 4 - How the data is collated into an input matrix for the WsNN

5. Meta Network Design

The meta-network is a Genetic Algorithm (GA) with modified mutation and crossover functions to handle the complex nature of the specific task. A typical GA takes a string of alpha-numeric inputs and modifies them according to the mutation and crossover functions. The system that has been devised must account for multiple component parameters - number of layers, neurons per layer, neuron placement (within each layer) and now additionally it must record details of inputs and outputs and be able to add and remove classes as necessary.

5.1. Inputs

Typically the inputs to a genetic algorithm are strings or numbers which are the ‘genes’ the genetic algorithm can modify. However, for this experiment, simple numbers would not suffice due to the complexity of the problem. As such, a custom input was defined as shown in Figure 5. On the left, Figure 5 shows an example 3 ‘layers’ (each considered a ‘variable’) – each pair of zeros represents the relative coordinates for a neuron from which its inputs will be derived, remembering that the dimensions of the layer and input pattern are identical. So the layer on the right in Figure 5 translates as a straight line.

5.2. Initial Population

The initial population is creates arbitrarily generated architectures. Once the initial population has been created tests are carried out on the data and the error rate is returned for each individual. This error rate is then multiplied by a function of that particular networks complexity (number of layers) to create a fitness value. For example, if a network returned an error rate of 45% using 5 layers, this would be multiplied by 1.25 – each layer adding 0.05 to this value – returning a fitness value of 56.25. By doing this, similar error rates are ranked by lowest complexity, as they are more efficient. These values are passed to the GA, which ranks them and prunes the worst fitness values.

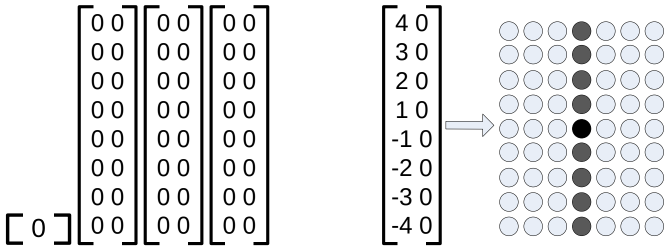


Figure 5 - (Left) Shows input design for the genetic algorithm, including the new parameter which defines the number of outputs the architecture will have. (Right) Shows graphic translation of coordinates to layer for a neuron in the centre of the layer

5.3. Crossover

Crossover takes selected individuals and 'mates' them by taking the first half of each layer of the first individual and the second half of each layer of the second individual, as shown in Figure 6 The number of input/output classes is chosen at random between the two crossover individuals. This creates a new individual (or architecture) which is then added to the next generation for testing.

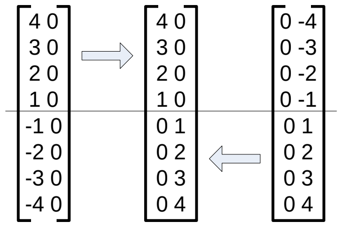


Figure 6 - Describes what happens during crossover

5.4. Mutation

Mutation takes a few individuals and modifies them slightly to create new architectures. It does this by 'adding' a matrix of integers that range between -1 and 1, as shown in Figure 7. The new parameter is modified accordingly as well within the range (between 5 and 11 input/output classes).

$$\begin{bmatrix} 4 & 0 \\ 3 & 0 \\ 2 & 0 \\ 1 & 0 \\ -1 & 0 \\ -2 & 0 \\ -3 & 0 \\ -4 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 4 & 1 \\ 1 & 0 \\ 2 & -1 \\ -1 & 0 \\ -1 & 0 \\ -3 & 1 \\ -4 & -1 \end{bmatrix}$$

Figure 7 - Describes what happens during mutation

5.5. Modifying Inputs and Outputs

The main aim of this paper is to determine whether the addition of the ability to increase or decrease the number of inputs will improve the decision making ability of the system/robot. To this end, each individual within the GA was given an additional parameter (See Figure 5) that could modify the number of training and test classes between 5, 7, 9 and 11. Each of these used the same data, but as the number of classes increases the distinction between them decreases. During crossover and mutation, the child's class size was selected at random from one of the parents.

6. Experimentation

To test the system, a scenario was created where several obstacles were placed in front of a virtual robot and sensor readings were recorded at critical points along the robots path. Two routes were created with a decision made part way through the robots journey, indicating two correct possible decisions. Depending on the decision made, the testing is sent down an alternate scenario path. Both paths ultimately lead to the goal; however the sensor data and therefore the input testing patterns differ. These paths are shown below in Figure 8.

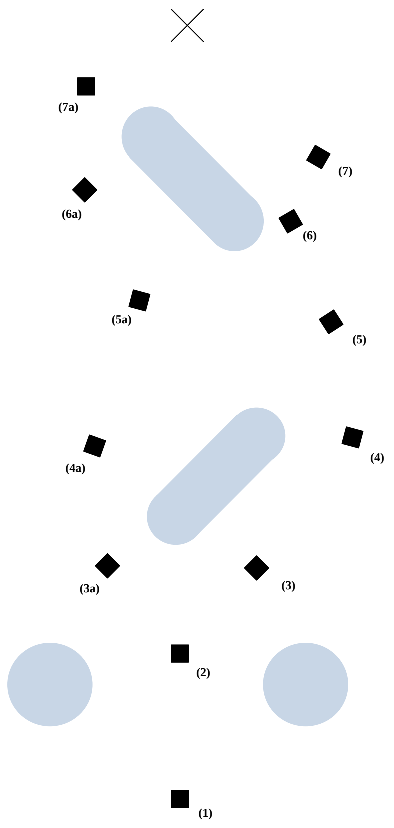
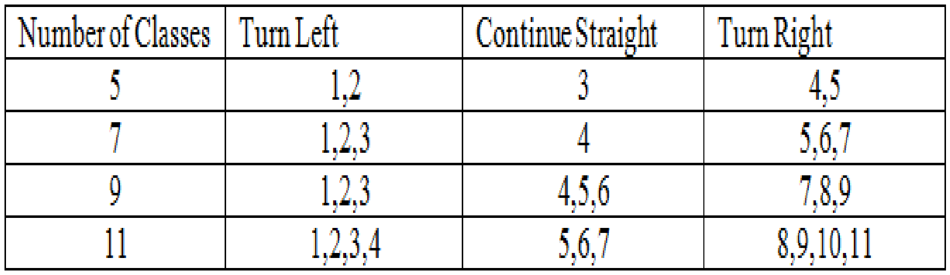


Figure 8 - Describes the Scenario with 2 routes. At step 2, choosing either left or right are correct, however a different route is used depending on this answer.

The initial population is created at random, including the number of potential output classes each individual would have. Two separate experiments were run, each with slightly different settings within the GA. This was done to see how the elite count, defined as the number of individuals considered to be the best by fitness are carried on unaltered to the next generation, could affect the outcome. The first would use an elite count of 2, keeping only the best pair for each generation. The second increased the elite count to 4. Each architecture was trained on one of the training sets shown below in table 2.

Table 2 - Describes how the classes are divided depending on the total number allotted



Number of Classes	Turn Left	Continue Straight	Turn Right
5	1,2	3	4,5
7	1,2,3	4	5,6,7
9	1,2,3	4,5,6	7,8,9
11	1,2,3,4	5,6,7	8,9,10,11

The fitness value was calculated using several measures. Firstly, a recognition rate for each step of the scenario in Figure 8 was calculated. Any incorrect decisions (for example, at step 2 seen in Figure 8, if the robot were to continue straight ahead rather than move left or right around the obstacle) were taken as 0. The results were then tallied and an average error rate for the architecture was found. From this, the fitness value is calculated as described earlier.

7. Results

The meta-network was applied to the scenario using the data created following Figure 8. The initial population of the network was set to 30, with limits on how large the network could get (12 layers). The reason that the limit is 12 is that past experience has shown little benefit to having networks larger than this. Additionally, larger networks a more computationally expensive, limiting their use. Separate experiments were conducted and the results were recorded, displayed in Figure 9. A control test was also performed using a system without the ability to modify the number of output/input classes. This is indicated in the lower portion of Figure 9. This control test used 11 classes for every individual. As can be seen from the results, it reduces the error as well, however not to the same extent. This is due to the proposed new system finding good architectures using a different number of inputs/outputs – significantly allowing it to improve the results beyond that of the control test.

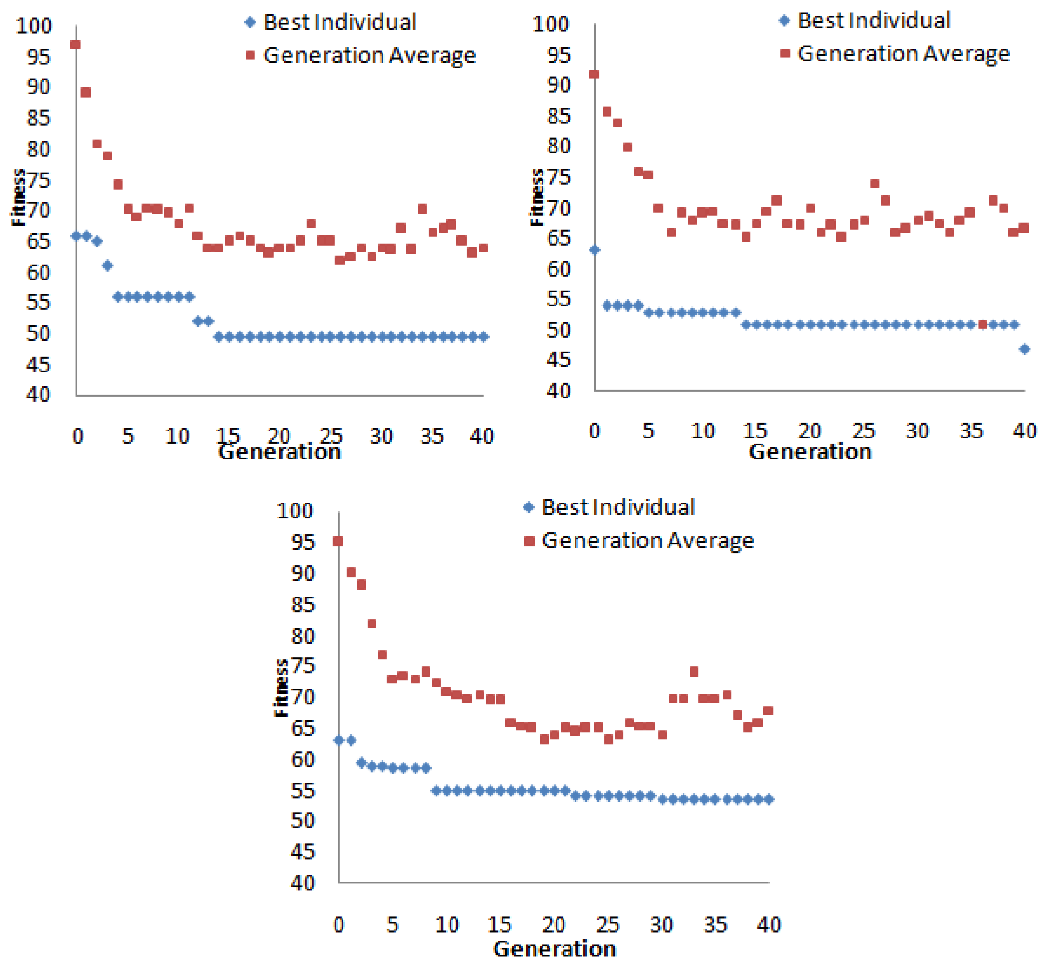


Figure 9 - (Above Left) Results using modifiable inputs and an elite count of 2. (Above Right) Results using modifiable inputs and an elite count of 4. (Below) Using a set class input size of 11.

8. Conclusion

A novel way of optimising an artificial neural architecture by modifying the number of classes a classifier has to choose from has been described. The system has been applied to the problem of robotic guidance using just sonar sensors, a significant and topical problem domain which is an area of continued research. The results are encouraging and leads itself naturally to the possibility of making the process more intelligent by removing and adding classes more dynamically.

Acknowledgements

This work was supported by INTERREG IV A 2 Mers Seas Zeen Cross-border Cooperation Program 2007-2013 under SYSIASS 6-20 project.

References

[1] *Evaluating the Application of Simple Weightless Networks to Complex Patterns.* **McElroy, B. and Howells, G.** Canterbury : s.n., 2010. 978-0-7695-4175-4.

[2] *Adaptive Meta-Network Design Employing Genetic Algorithm Techniques.* **McElroy, B and Howells, G.** 2011, Artificial Intelligence and Soft Computing (ASC 2011).

[3] *Improved Adaptive Meta-Network Design Employing Genetic Algorithm Techniques.* **McElroy, B and Howells, G.** 2011, ICINCO - 8th International Conference on Informatics in Control, Automation and Robotics.

[4] *GCN: the generalised convergent network.* **Howells, G., Fairhurst, M.C. and Bisset, D.L.** Edinburgh : Fifth International Conference on Image Processing and its Applications, 1995 . 0-85296-642-3.

[5] *Meta-learning approach to neural network optimization.* **Kordík, Pavel, et al.** s.l. : Neural Networks, 2010, Vol. 23. 0893-6080.

[6] *Meta learning evolutionary artificial neural networks.* **Abraham, Ajith.** 38, s.l. : Neurocomputing, 2004, Vol. 56. 0925-2312.

[7] *Evolutionary artificial neural networks by multi-dimensional particle swarm optimization.* **Kiranyaz, Serkanand et al.** 10, s.l. : Neural Networks, 2009, Vol. 22.

[8] *Evolutionary Reinforcement Learning of Artificial Neural Networks.* **Siebel, Nils T and Sommer, Gerald.** 3, s.l. : IOS Press, International Journal of Hybrid Intelligent Systems, October 2007, Vol. 4. 1448-5869.

[9] *Pattern recognition and reading by machine.* **Bledsoe, W. W. and Browning, I.** Boston, Massachusetts : AFIPS Joint Computer Conferences, 1959.

[10] **Austin, James.** *RAM-Based Neural Networks.* s.l. : World Scientific Publishing Company, 1998. 981-02-3253-5.

[11] *BCN: an architecture for weightless RAM-based neural networks.* **Howells, G., Fairhurst, M.C. and Bisset, D.L.** Orlando, FL : IEEE International Conference on Neural Networks, 1994 . 0-7803-1901-X .

[12] *Gray codes and paths on the n-cube.* **Gilbert, E.** s.l. : BellSystem Technical Journal , 1958, Vol. 37. 815-826.

[13] **Nolfi, Stefano and Parisi, Domenico.** Evolution of Artificial Neural Networks. *In Handbook of brain theory and neural networks, pp. 418-421.* s.l. : MIT Press, 2002.

Authors

Ben McElroy recently completed a Master of Science in Information Security and Biometrics. Working in association with the SYSIASS project he is currently undertaking Doctoral studies at the University of Kent. He is utilising weightless artificial neural networks in the field of automated robot guidance with the objective of developing a fully autonomous wheelchair.

Dr Gareth Howells has been involved in research relating to image processing, artificial neural networks and pattern recognition for twenty years. He has developed several novel pattern classification systems employing both weightless and logic based artificial neural network technologies. In addition to the development of novel architectures, previous work has centred around the merging of techniques taken from formal mathematics and formal logic with existing algorithms pertaining to Artificial Evolutionary Systems. Further work has addressed issues concerning document security and Biometric techniques, image processing and pattern recognition.