Artificial Intelligence 175 (2011) 673–693

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

From Bidirectional Associative Memory to a noise-tolerant, robust Protein Processor Associative Memory ☆

Omer Qadir *, Jerry Liu, Gianluca Tempesti, Jon Timmis, Andy Tyrrell

Department of Electronics, University of York, Heslington, YO10 5DD, York, UK

ARTICLE INFO

Article history:
Received 8 March 2010
Received in revised form 19 October 2010
Accepted 21 October 2010
Available online 27 October 2010

Keywords:
Self-organising
Self-regulating
Associative Memory
Protein processing
Hetero-associative
BAM
PRLAB
SOIAM
SABRE
Mobile robotics

ABSTRACT

Protein Processor Associative Memory (PPAM) is a novel architecture for learning associations incrementally and online and performing fast, reliable, scalable hetero-associative recall. This paper presents a comparison of the PPAM with the Bidirectional Associative Memory (BAM), both with Kosko's original training algorithm and also with the more popular Pseudo-Relaxation Learning Algorithm for BAM (PRLAB). It also compares the PPAM with a more recent associative memory architecture called SOIAM. Results of training for object-avoidance are presented from simulations using player/stage and are verified by actual implementations on the E-Puck mobile robot. Finally, we show how the PPAM is capable of achieving an increase in performance without using the typical weighted-sum arithmetic operations or indeed any arithmetic operations.

1. Introduction

The work in this paper stems from research into parallel architectures targeted towards problems in the domain of AI. Although, many architectures exist that attempt solutions for problems in this domain [38,2,22,37,3,21]. These are based on Arithmetic and Logic Units (ALUs) and often are simply Von Neumann style processors (with a few modifications) connected together in parallel. Despite the abundance of AI algorithms and machine learning techniques, the state of the art still fails to capture the rich analytical properties of biological beings or their robustness. In our architecture, computation is moved into memory and intelligence is expected to emerge from memory operations rather than ALU operations. It is based on the connectionist approach, which affords a level of inherent fault-tolerance, which is further enhanced by a design composed of a large number of elements, none of which are individually critical to the overall correct operation. This is in contrast to more traditional approaches to fault-tolerance where each module or component is designed to be unique and (hot or cold) spares are maintained which can be used to replace faulty modules. In our architecture, faults should result in graceful degradation, rather than nodes having to be replaced. It is postulated that such an architecture is better suited to implementing artificial intelligence than the more traditional ALU based architectures. Although some portions are *inspired* by biological neural networks, the objective is not to build an architecture *for* Artificial Neural Networks. Note that

☆ The research is supported by EPSRC under grant no. FP/F06219211.

* Corresponding author.
E-mail addresses: oq500@ohm.york.ac.uk (O. Qadir), yl520@ohm.york.ac.uk (J. Liu), gt512@ohm.york.ac.uk (G. Tempesti), jt517@ohm.york.ac.uk (J. Timmis), amt@ohm.york.ac.uk (A. Tyrrell).
URL: http://www-users.york.ac.uk/~oq500/ (O. Qadir).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.10.008

the current version does not fully satisfy the original motivation of designing an architecture for hardware, however, it is suitable for software implementations.

Traditional memory stores data at a unique address and can *recall* the data upon presentation of the complete unique address. Auto-associative memories are capable of retrieving a piece of data upon presentation of only partial information from *that* piece of data. Hetero-associative memories, on the other hand can recall an associated piece of data from *one* category of input upon presentation of data from *another* category of inputs. Hopfield networks [13] have been shown to act as auto-associative memories [6] since they are capable of remembering data by observing a portion of the data. Biological neural networks, on the other hand, are hetero-associative memories since they can remember a completely different item to the one presented as input. Bidirectional Associative Memories (BAM) [18] are Artificial Neural Networks that have long been used for performing hetero-associative recall. This paper starts by investigating the efficacy of BAMs for performing hetero-associative recall in the context of AI for robotics. Association is performed between sonar values and the actions needed to avoid obstacles on a robot. Two different training algorithms are examined for the generation of the correlation matrix in the BAM, namely the original BAM training algorithm outlined in Kosko [18] and the popular Pseudo-Relaxation Learning Algorithm for BAM (PRLAB) first presented in Oh and Kothari [27]. Even though the PRLAB guarantees optimal learning, distinguishing between values that are closely spaced is a weakness of the BAM. Nevertheless, such values are realistic inputs for a real-time, noisy environment and we show that although parameters are optimized to maximize recall, even small confusions can result in major problems for a mobile robot, particularly in an online learning environment. The paper goes on to compare the structural differences between the Protein Processor Associative Memory (PPAM) and the BAM and then presents some results of implementations of the PPAM on the same problem. It discusses the robustness and noise tolerance of the PPAM architecture and also compares this with noise tolerance reported by Sudo et al. [35] for the Self-Organising Incremental Associative Memory (SOIAM). The paper is structured as follows: Section 2 is a brief discussion of BAM and PRLAB – their constraints and advantages. Section 3 outlines the experimental method, setup, parameter tuning and details the experiments performed. Section 4 presents the results of the BAM implementation on the player/stage robot. Section 5 describes the PPAM ending with a structural comparison of the BAM and the PPAM. Section 6 reviews the results from the PPAM implementation on the player/stage robot. Section 7 discusses the results and noise tolerance properties comparing them with SOIAM and also presents some inferences that can be drawn from the results. Section 8 summarises, concludes and discusses future directions.

2. Bidirectional Associative Memories

Bidirectional Associative Memories [18] are a generalization of the Hopfield networks [13] both of which have their beginnings in the Correlation Matrix Memories ([17], cited in [1]). BAMs have long been the subject of analysis and have formed the basis of many later models. Attempts to find the best training algorithm include [5] which uses an exponential rule, [33] which uses genetic algorithms, [45] which uses descending gradient method, [7] which uses linear programming techniques among others. [32] is a non-iterative Morphological BAM while [41] is a non-iterative feedforward BAM. [1] is yet another variation based on two binary meta-operators and called the Alpha–Beta BAM. Cao et al. [4] discuss higher order BAMs with time delays and Lu et al. [23] consider the effects of topology on the performance of Hopfield-type associative memories.

Like other traditional neural networks, this 2-layer, non-linear, recurrent neural network starts by training on a defined training dataset and then moves on to the actual test set. The weighted-sum approach means that the capabilities can be analysed and proven on paper using mathematics. Furthermore, the original training algorithm does not require batch learning and therefore is easily adaptable for online learning. On the other hand, it results in sub-optimal utilization of capacity. Furthermore, as shown by Kosko [18], the BAM can get confused when storing one-to-many or many-to-one relationships. In addition, if too many pairs are trained, the correlation matrix becomes unstable and the BAM is likely to forget all previously learnt pairs.

Obviously it is desirable to maximise the storage capacity or at least to quantify it and many attempts have been made in this (latter) direction. Tanaka et al. [36] perform statistical physical analysis and estimate the capacity of pairs to be retrievable allowing a finite fraction of retrieval error to be $0.1998N$ where N is the number of neurons in each layer of two layers. The fact that the capacity of a BAM is tied to the number of nodes is a drawback in terms of scalability. Wang and Vachtsevanos [39] derive the storage capacity of a discrete BAM which is further verified by our experimental results. It shows that the storage capacity is much smaller than expected because of the *mis-learning behavior* where the BAM connection matrix confuses similar patterns because of the bipolar encoding scheme. Since bipolar encoding performs better on average than binary encoding, as shown by Kosko [18], this mis-learning is unavoidable.

In essence, the problem is to maximise the number of patterns superimposed on one memory medium, namely, the weights of connections in a correlation matrix. Therefore, to guarantee recall, the training vectors *must* be orthogonal. One obvious solution is to re-encode non-orthogonal training data so that it becomes orthogonal as done by Simpson ([34], cited in [27]). Of course this does imply that the complete training set is known in advance, an assumption that is not valid for online, real-time learning where training data is *encountered* sequentially and the complete size is not known in advance. PRLAB [27] is an iterative learning algorithm for discrete BAMs that maximises the storage capacity of the BAM without the need for any preprocessing or re-encoding of training data. Furthermore, it guarantees perfect recall of all training pairs if it is possible to store them as stable states in the correlation matrix. All training pairs are examined cyclically in each

Algorithm 1: Pseudo-code for innate, object-avoidance algorithm

```
Input: sonar values array A
Output: forward speed s_o, rotation speed r_o
// s_f is constant forward speed used to move forward
// s_s is constant forward speed used to avoid obstacles
// r_i is constant rotation speed to avoid obstacles
C ← true if any A[i] shows distance to object is less than a threshold value;
if C == true then
    // set forward speed to 0
    s_o ← s_s;
    if object to left closer than object to right then
        r_o ← −r_i;
    else
        r_o ← r_i;
else
    s_o ← s_f;
```

iteration (called an *epoch*) and if an associative pair is not stored in a stable state, the weights are adjusted further. Where other algorithms suffer from parameter tuning issues (notably deciding the optimum step size to maximize learning for iterative algorithms), PRLAB is highly insensitive to parameter values and initial configurations. To guarantee training, the PRLAB needs to see *all* training pairs and adjust weights in multiple iterations. Sudo et al. [35] adapt PRLAB for online incremental learning by sequentially providing it with associative pairs so that the algorithm observes only one pair. They show how this results in the BAM (and also Hopfield networks) forgetting previously learned data. However, this is an unfair comparison because the entire strength of PRLAB lies in the fact that it cycles through training pairs iteratively and makes changes in the weight matrix if the pair is not stored properly.

3. Experiments

3.1. Method

A single agent is placed in a static environment and initialized with a simple reactive object-avoidance algorithm (Algorithm 1), which controls its movements. If the sonar values observed indicate that a collision is imminent, the agent is rotated to the left or the right depending on which direction indicates that objects are farthest. The movement value generated and the sonar value observed, form the two variables in an associative pair. The idea being that after running for a while on the *innate* object-avoidance algorithm, the agent should be able to switch to the *recalled* object-avoidance algorithm seamlessly. The associative memory can be seen as observing the sonar values (as they are encountered by the agent) in conjunction with the corresponding actions or movement values generated by the *innate* object-avoidance algorithm, and associating these two together. The same experiments are performed on the BAM with the original Kosko's learning method, BAM using PRLAB and also the PPAM.

3.2. Setup

Player/stage [8,28] was used to simulate the environment. The agent used was the Pioneer P3-DX Robot[1] with 7 functional sonars pointing towards its relative north with an angular difference of 22.5° from each other. The sonars have a range of 5 meters and the area within which the robot navigates is 14×14 meters. The stage driver for the pioneer sonars returns real values in the range $0.0 \leqslant s \leqslant 5.0$. Since analogue, real-world data would need to pass through analogue-to-digital converters which perform quantization, this is approximated by discretizing the sonar values into 10 bins so that each bin is 0.5 meters. Since there are 7 sonars, this makes a total of 10^7 or 10 million possible unique values for the 7-dimensional variable that represents sonar input. Movement is represented by a *forward speed* and a *rotation speed* both of which are real values. The object-avoidance algorithm generates one of two possible forward speeds (slow or fast), depending upon the sonar values. Similarly, for rotation, the algorithm can generate one of three possible values – "turn left", "turn right" or "go straight". Therefore, rotation has 3 discrete values and forward speed has 2 discrete values making 6 total possible unique values for the 2-dimensional variable that represents movement. Note that since the object-avoidance algorithm does not have memory, quantization errors can cause the agent to get stuck in race conditions where it first turns left, then right, then left and so on. If the "slow" forward speed is set to a non-zero value, this means that the agent continues to move forward while turning, which in turn means that the agent would be able to navigate out of such race conditions without any effort by the object-avoidance algorithm. This is undesirable because the motivation of using the simple algorithm is to test (in future) if the associative memory can perform better than the algorithm given higher level "goals" as input in associative pairs.

[1] http://www.activrobots.com/ROBOTS/p2dx.html.

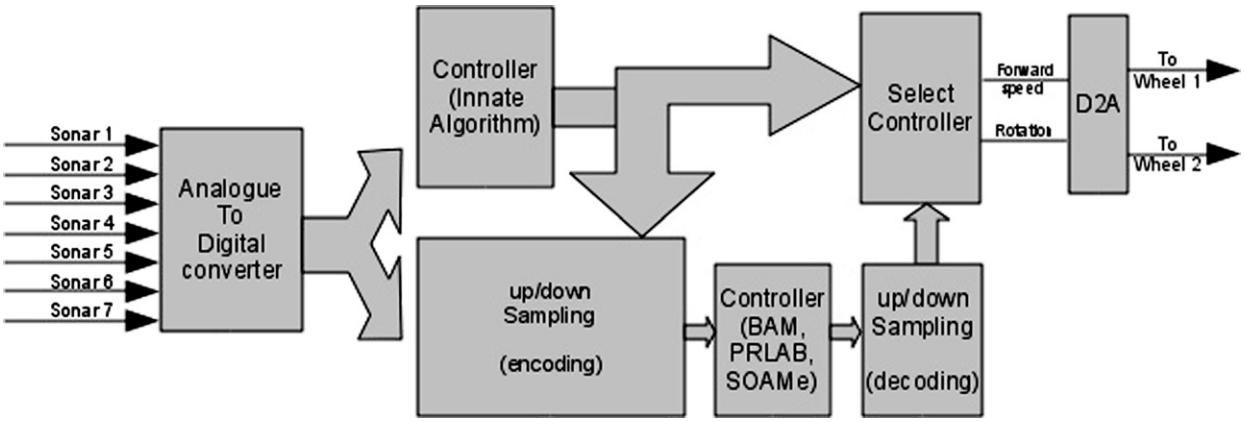


Fig. 1. Controller's connection topology in the Agent.

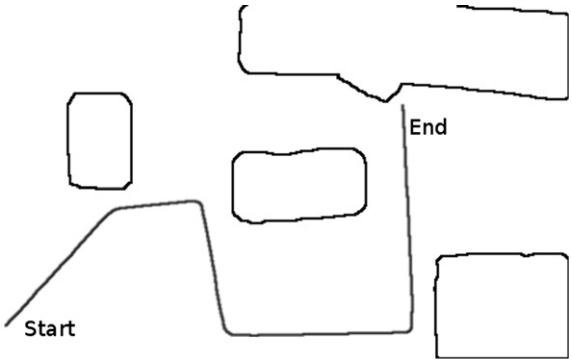


Fig. 2. Agent Trajectory using Algorithm 1.

Fig. 1 illustrates the topology for the controller (BAM or PPAM) in an agent. The sonar values are fed to analogue-to-digital converters which output digitally encoded values. As stated above, the analogue-to-digital (A2D) converters quantize the sonar values into 10 uniform, discrete bins. Thus the output of the A2D converters are 4-bit binary values, which are directly used by the innate controller (Algorithm 1). Depending on the size of the network forming the associative memory (number of nodes in the BAM or PPAM), the discretized values may need to be up or down sampled. In a BAM with N nodes in a layer, the input vector V for that layer will be $V = I_1, I_2, \ldots, I_N$. Assuming a bipolar implementation, I_j may be described as $I_j \in \{-1, +1\}$. In a $3x + 7y$ BAM network, there are 3 nodes in the movement layer and 7 nodes in the sonar layer, and so $N = 7$ for the sonar layer. This means that the 4-bit output of the A2D converters needs to be down-sampled to a 1-bit value, since there are 7 sonar values and 7 nodes. For the $3x + 28y$ network, there are 28 nodes in the sonar layer, this means that each sonar value can be represented using 4 nodes ($\frac{28}{7}$), thus re-sampling is not required. Similarly, for the $12x + 63y$ network, an up-sampling is required. The sampling function for the PPAM is very similar and is discussed in more detail in Section 6.1 after the description of the architecture. The advantage of using this up/down sampling is that it realistically models quantization errors. This is due to the fact that from the perspective of the associative memory, the movement values generated by the innate algorithm are also a source of (analogue) external inputs. Other than this re-sampling (and conversion to bipolar for the case of BAM/PRLAB) no other encoding is performed on the input data. At the output, a corresponding up/down sampling process is performed if one was required at the input. Next, the movement values from either the innate algorithm or the recalled movement values from the associative memory are forwarded on to the actuators in the agent. The choice can be made online and is under user-control. The selected movement values are then converted to analogue values and transmitted to the relevant motors.

The trajectory of the agent while being controlled by the object-avoidance algorithm is shown in Fig. 2 (objects are shown in black). The end point of the trajectory is where the algorithm gets stuck in a race condition. This occurs after 670 time steps where, in each time step the agent has observed one 7-D sonar value and the algorithm has generated one 2-D movement value, together making one associative pair. These 670 pairs are the training dataset. Note, firstly, that the training dataset is not pruned to select *key* points but instead includes all observed values. Secondly, the dataset has a one-to-many relationship between the pairs (movement vs sonar) as many different sonar readings can result in the agent continuing to move straight ahead. This places the BAM at a disadvantage, since dealing with such relationships is a known weakness. Furthermore, although the dataset is not guaranteed or indeed even likely to be composed of orthogonal vectors,

it is our opinion that real-time data is very unlikely to be orthogonal and realistic associative memories cannot depend upon *a priori* knowledge for preprocessing this training data to convert it into orthogonal pairs.

3.3. Tuning the BAM correlation matrix

The training dataset (of 670 pairs) was composed of 228 unique associative pairs. Each pair can be represented using a 3-bit value for movement and a 28-bit value for sonar (4-bit value for each of 7 sonars). The perfect associative memory would be able to learn all 228 associative pairs and recall them perfectly so that if the agent is re-initialized at the start coordinates (in Fig. 2), it would follow the trajectory exactly and get stuck in the race condition at the end. In order to get the best performance from the BAM, parameters for both the original Kosko's learning algorithm and also PRLAB were tuned. Whether *any* BAM correlation matrix using *any* BAM configuration is capable of learning all 228 pairs is dependant upon whether the 228 pairs are orthogonal. The rest of this section presents our attempts to maximize the number of associative pairs that are learnt by the BAM by tuning parameters. Once the correlation matrix that can remember the maximum number of pairs is found, this is used to control the agent in the player/stage environment (Section 3.4). These player/stage simulations explore the effect of imperfect recall in BAMs and compare them with imperfect recall in PPAM.

3.3.1. Kosko's algorithm

For the original BAM learning algorithm, as presented by Kosko [18], the parameters that need tuning are the initial values of the weight matrix and the number of nodes in the 2 layers of a BAM. Node configurations of $3x + 7y$, $3x + 28y$, $6x + 28y$, $12x + 28y$, $12x + 63y$ and $30x + 28y$ were attempted, where $3x + 7y$ means 3 nodes in layer 1 and 7 nodes in layer 2. The numbers were chosen to be multiples of the movement values and the sonar values. Given that the BAM operates on bipolar data, 7 nodes in a layer means that the layer can distinguish between 2^7 or 128 bipolar values. Therefore, the $3x + 7y$ configuration represents a case where sonar values have to be down-sampled. The $3x + 28y$ configuration is sufficient to represent the entire range of values. Higher node configurations are used to test whether up-sampling has any effect on the number of pairs learnt, as the BAM would train better if the associative pairs are farther apart. Where necessary, the training dataset was re-encoded to up-sample or down-sample values. For example, the $3x + 7y$ configuration requires the 4-bit sonar values for each sonar to be down-sampled to 1 bit. In this case, the training dataset turned into 18 unique associative pairs (for all other configurations it remained 228). For higher node configurations ($6x + 28y$, $12x + 28y$ and so on), re-encoding is not essential but is preferable because otherwise the same data points become closer together. To illustrate, when using 3 nodes, the range of data is 2^3 or 8 values and therefore the maximum distance between any two numbers is 6 integer values. Thus, for instance, 4 is separated from 1 by 3 integer values out of a possible of 8. On the other hand, when using 6 nodes, the range of data is 2^6 or 64 values. If the same dataset (as used for 3 nodes) is used without re-encoding (or up-sampling) then in this case, 4 would be separated from 1 by 3 integer values out of a total possible of 64.

For each node configuration, initial weight values were set between -1.0 to $+1.0$ in steps of 0.02. The $3x + 28y$ node configuration was further tested in an extra long range with weight values in the range of -100.0 to $+100.0$ in steps of 0.02. Whereas the maximum number of pairs learnt (including almost-recalled pairs) in the smaller range was 186 out of 228, in the larger range, the maximum number of pairs learnt was 187 out of 228, thereby indicating that the effect of initial weights on the correlation matrix is minor. For each initial weight value, the BAM was trained with an increasing number of associative pairs and the number of pairs successfully recalled was tested in each iteration. A value is considered to be *perfectly recalled* if the value recalled is exactly the same as was trained. Considering all values trained as attractors, a recalled value is *almost-recalled* if the closest attractor is the correct one. Euclidean distance is chosen as a measure for *closeness* because of the need for orthogonality in BAMs. Since the BAM uses weighted-summation, Euclidean distance is more appropriate than other measures like hamming distance for testing if data points are far apart and, therefore, less likely to be confused with each other. Note that detecting almost-recalled values implies post-processing of the output and *a priori* knowledge of the attractors to determine which attractor is closest. This means that the memory must have another copy of all the associative pairs stored and test the recalled value against all the associative pairs to find the closest one – in essence, duplicating its own behaviour. Despite being unrealistic (particularly for online incremental learning), this method was used to maximise performance. The objective of this tuning is to determine the BAM correlation matrix that can recall the most number of pairs in the training dataset. This can then be used to control an agent in the player/stage environment. If a correlation matrix can be found that can recall *all* the pairs, it would be able to follow the trajectory of the object-avoidance algorithm exactly.

As mentioned above, for each initial weight configuration, an increasing number of associative pairs were presented for training. Fig. 3 shows how the capacity of the BAM varies as the number of associative pairs in the training dataset is increased. This is an important step for this learning algorithm because, unlike PRLAB, it does not guarantee utilization of maximum storage capacity. Therefore, as can be seen from the plots, if too many pairs are presented, the capacity of the memory begins to fall. Each point in the plots in Fig. 3 is the one with the *best* initial weight value where the primary fitness criterion is to maximize the number of pairs recalled and the secondary fitness criterion is to maximize the number of pairs recalled perfectly. Note that up to size 8 for configuration $3x + 7y$ and up to about size 25 for other configurations the BAM can successfully recall *all* training pairs presented. However, any pairs presented beyond this point, are only learnt

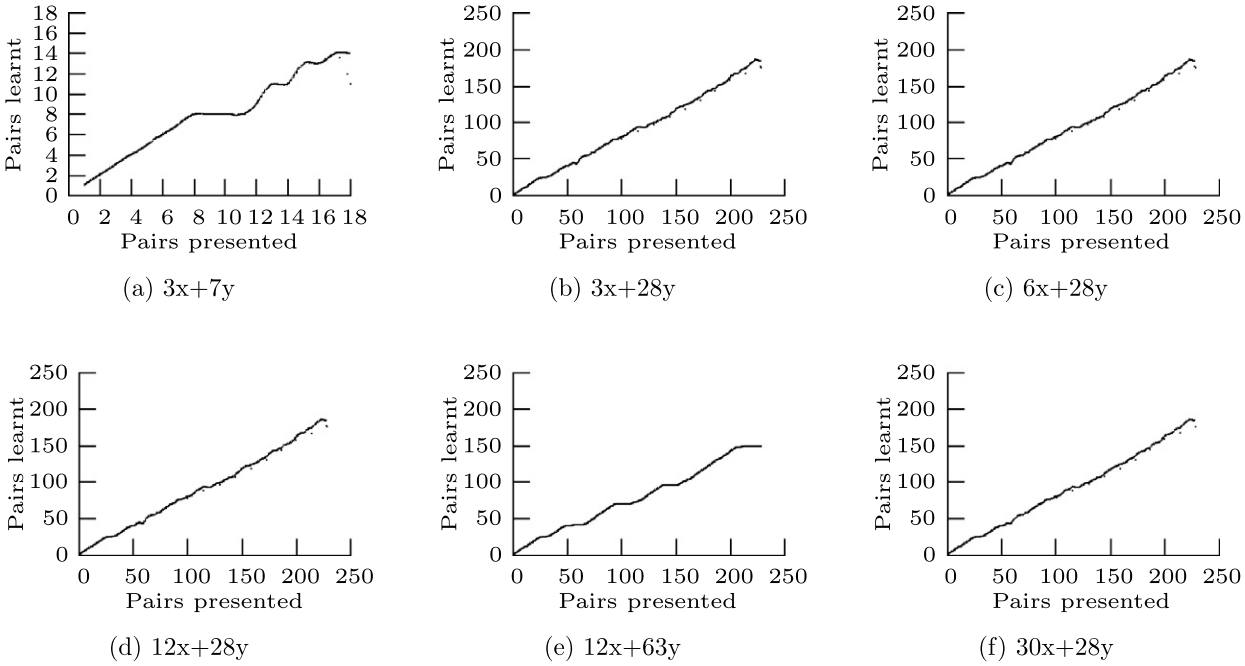


Fig. 3. Memory capacity using Kosko learning.

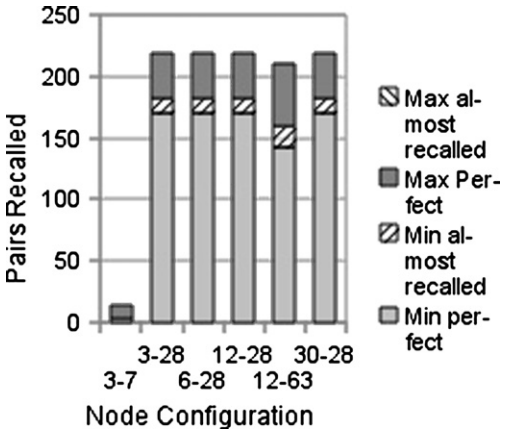


Fig. 4. Capacity of BAM using PRLAB.

at the expense of some other pair. The maximum number of associative pairs learnt using any configuration of nodes and initial weights is 187.

3.3.2. PRLAB

The parameters that need tuning are the initial values of the weight-threshold matrix, number of nodes, number of epochs, relaxation factor λ and constant ξ deciding the basins of attraction. Oh and Kothari [27] show that PRLAB is insensitive to parameter values, and indeed other implementations don't train the parameters at all [35]. However, as shown by Fig. 4 we find that if the dataset is not entirely orthogonal, training can result in a difference of up to 68 pairs in the worst case, which is the difference between PRLAB and Kosko's original method. The same node configurations and training dataset were used as for Kosko's algorithm. Details of the training dataset can be found in Sections 3.1 and 3.2. Parameters were tuned in two steps, the first performing a long-range sweep (with larger steps) and the second performing a short-range sweep (with smaller steps) close to the best region found in the long-range sweep. Table 1 shows the long-range sweep parameters which result in a total of 10,000 iterations, while Table 2 shows the short range sweep parameters. The same method for perfectly recalled and almost-recalled as used for tuning Kosko's algorithm (Section 3.3.1) is used here to tune PRLAB. The $6x + 28y$ node configuration was further tested in an extra long range as shown in Table 3. Whereas the maximum number of pairs learnt (including almost-recalled pairs) using the long-range/short-range search was 219, using the extra long range search, it was 220. Fig. 4 is a bar-chart showing the capacity of the BAM trained using PRLAB. The *best*

Table 1
Long range search parameters for PRLAB.

Parameter	Min	Max	Step
Weight-threshold	−0.5	+0.5	0.1
Epochs	5	70	5
λ	0.5	2.3	0.2
ξ	0.05	0.2	0.015

Table 2
Short range search parameters for PRLAB.

Parameter	Step	Iterations
Weight-threshold	0.01	10
Epochs	1	10
λ	0.1	4
ξ	0.002	5

Table 3
Extra long range search parameters.

Parameter	Min	Max	Step
Weight-threshold	−5.0	+5.0	0.1
Epochs	5	75	3
λ	0.1	2.4	0.1
ξ	0.01	1.0	0.005

Table 4
Best parameters for PRLAB.

Network	Epochs	ξ	λ	Weight
$3x + 7y$	11	0.05	0.5	−0.3
$3x + 28y$	20	0.120	1.60	−0.34
$6x + 28y$	35	0.9505	1.54	0.9
$12x + 28y$	23	0.125	1.58	−0.3
$12x + 63y$	65	0.2	0.5	−0.1
$30x + 28y$	23	0.125	1.58	−0.3

parameter values (Table 4) are used where the fitness criterion maximizes the number of pairs recalled while maximizing the number of pairs recalled perfectly and minimizing the number of epochs required to learn the associative pairs. The thatched portion indicates the number of values that were almost-recalled correctly. The absolute minimum number of pairs learnt using any combination of parameters is depicted by the light-grey bar plus the thatched bar for "Min Almost Recalled". The maximum number of pairs learnt is the full height of each bar. Note that max-almost-recalled is 0. This means that when the correlation matrix is fully optimized (so that the maximum number of pairs is stored) all pairs stored are stored perfectly and the Euclidean distance measure (almost recalled) does not enhance performance further. Note that with the exception of the $3x + 7y$ node configuration, the worst-case PRLAB correlation matrix performed as well or better than Kosko's original correlation matrix. Furthermore, the trained PRLAB correlation matrices all performed approximately equally, indicating that the number of associative pairs learnt by the BAM was unaffected by the number of nodes in the layers and that the maximum number of (orthogonal) pairs that could be learnt had been reached. More pairs can only be learnt by increasing the orthogonality of the pairs. Thus, the dataset would have to be re-encoded. Disregarding the implications of this on the structure of the BAM, and assuming that it is possible online in real-time, no encoding can ensure orthogonality for all real-world datasets – and therefore at some point, the associative memory would have to deal with imperfect recall. The objective here is to explore the behaviour of the BAM and compare it with that of the PPAM in this situation where data is imperfectly stored and recalled as described in Section 3.4.

3.3.3. Verifying with alternate dataset

An alternate dataset was also generated and used as a training dataset in the parameter tuning process explained above. The object was to verify that the results observed are not due to some property of this *one* dataset. For this purpose, the agent was placed in a completely different environment. Fig. 5 shows the modified environment as well as the trajectory taken by the agent. This trajectory was composed of 770 time steps before the agent got stuck in a race condition at the "end" position. These 770 time steps were composed of 283 unique training data points, which formed the alternate dataset. Tuning for both Kosko's method and PRLAB was performed in the same manner as outlined above. Results from some of the network configurations are shown in Fig. 6.

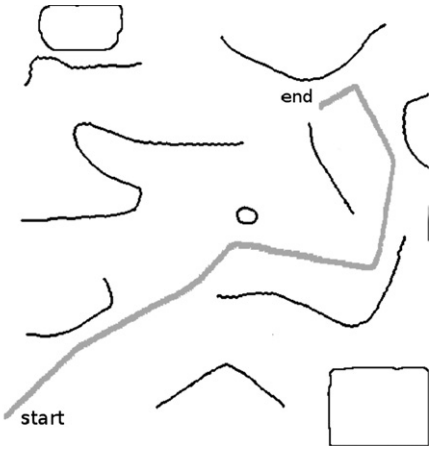
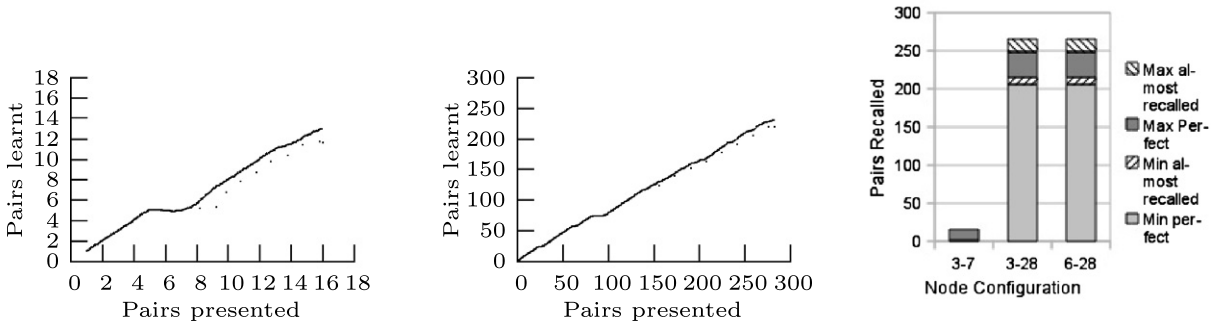


Fig. 5. Trajectory in alternate environment.



(a) 3x+7y using Kosko (b) 6x+28y using Kosko (c) PRLAB

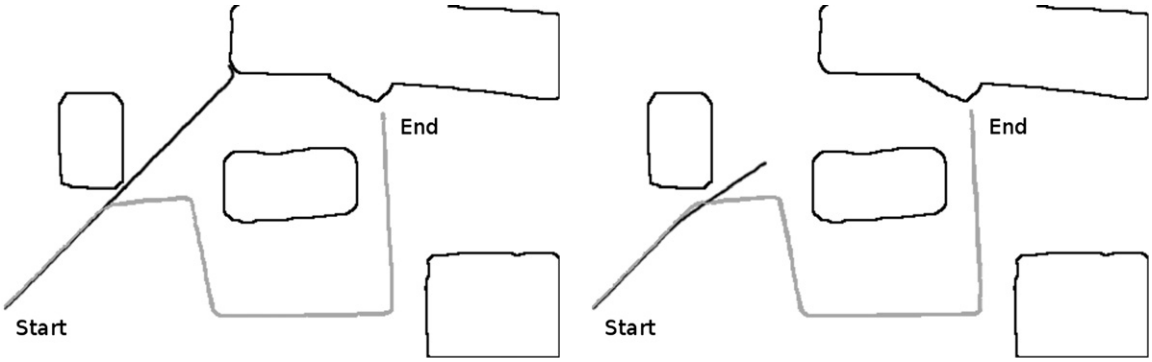
Fig. 6. Alternate dataset memory capacity.

3.4. Robot simulations

The objective of the previous experiments was to determine the *best* correlation matrix for the BAM. If a correlation matrix had been found that could recall all 228 associative pairs in the training data, then it would have been trivial to reset the agent to the initial starting location, since the agent would have followed the path of the object-avoidance algorithm exactly. In that case, it would only remain to observe the behaviour of the agent in new, previously unobserved environments. However, since no such correlation matrix (that could recall all 228 pairs) was found, experiments in both environments are useful. Although we already know that the maximum number of pairs that can be recalled using *any* BAM configuration is 220 out of 228, the objective of the following experiments is to determine how this effects a robot in a real-world, real-time environment. Note that the following description is relevant for the experiments performed with the PPAM as well.

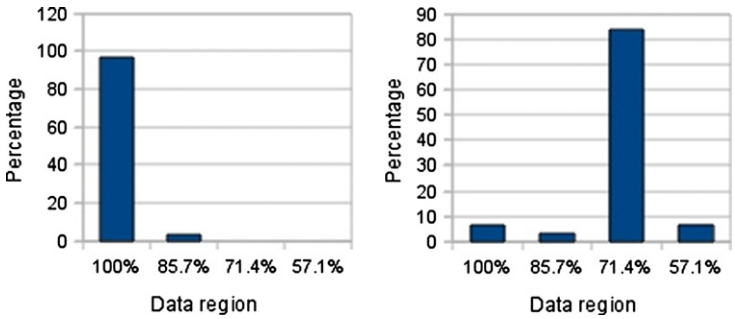
In the first phase, the agent in player/stage is reset to the initial starting position and orientation and allowed to run for another 670 time steps. This time, however, the movements are *recalled* from memory and not *calculated*. If the associations have been learnt correctly, the agent should follow the exact same path as it took the first time around and at the end of 670 steps, it would be at the "end" position in the correct orientation. The second phase begins after these 670 steps when the agent is placed in new environments. The sonar values encountered now would only have been (at best) partially observed before and therefore the associative memory receives inputs that have been only partially learnt. This is the same as observing a noisy training vector. The agent is placed in three different locations and run for a total of 763 more time steps altogether.

Each of the network configurations described above ($3x+7y$, $3x+28y$, $6x+28y$, $12x+28y$, $12x+63y$ and $30x+28y$) was used to control the agent in the player/stage simulation. The best trained correlation matrix (Section 3.3.2) was used and the method to detect almost-recalled values was also implemented. Note that for the case of PRLAB, using a trained matrix is not incremental learning. Therefore, as the agent moves through the environment and learns associations, it needs to store another copy of the entire training dataset separately from its associative memory so that this copy can be presented to the associative memory when the weights are to be updated! Nonetheless, this is necessary because implementing an incremental version, as done by Sudo et al. [35], loses the guaranteed learning capability of PRLAB.



(a) $3x + 7y$ (b) $3x + 28y$, $6x + 28y$, $12x + 28y$

Fig. 7. Agent trajectory for Kosko's method.



(a) $3x + 7y$ (b) $3x + 28y$, $6x + 28y$, $12x + 28y$

Fig. 8. Environment input by percentage previously observed for Kosko's algorithm.

Table 5
Associativity in $3x + 7y$ Kosko's network.

Percentage previously observed	100%	85.7%
Recall (confidence)	83.8%	76.1%
Accuracy	34.0%	54.3%
Critical (decisions to avoid collision)	80.2%	50.0%
C+A (critical and correct)	17.8%	8.7%
R+C (critical decisions claimed to be correct)	76.4%	34.3%
R+A (correct when claimed to be correct)	23.6%	65.7%

4. BAM results

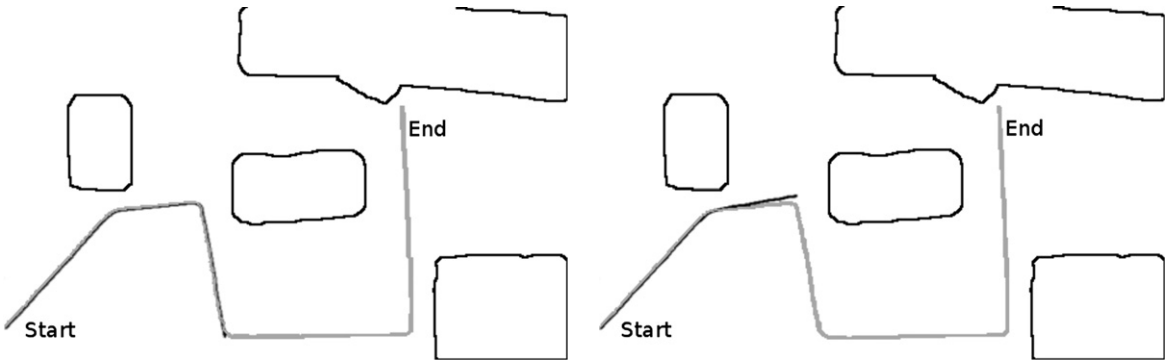
4.1. Kosko's method

The trajectory taken by the agent when it was re-initialized to the original start point is shown in Fig. 7. The trajectory in light-grey is the one taken by the agent while it was being controlled by the object-avoidance algorithm, while the dark-grey trajectory is the one taken by the agent when being controlled by the BAM. As can be seen from Fig. 7a, the agent, in fact, had a collision with the wall and stalled. Increasing the nodes from $3x + 7y$ to $3x + 28y$ improved the performance so that the agent now successfully avoided collisions, however, it suffered from race conditions. The distance travelled by the agent is shorter because the agent stops when it encounters a race condition where the memory first decides to turn left then right and so on. Increasing the nodes further (after $3x + 28y$) had no effect on the trajectory. This is because the race condition would always occur at a certain set of sonar inputs as the BAM was incapable of storing the required associative pair in a stable state.

Fig. 8 shows a break-down of the environmental inputs in terms of percentage of the environment observed before. The 100% region contains data points that would occur as the robot follows the path after it is reset to the initial start location and heading. Partially observed sonar inputs would occur after the robot is placed in a new environment or if the robot strays from the original path (as it does in Fig. 7). The 85.7% region contains data points where only 6 of the 7 sonar values have been observed before. This represents the hetero-associative recall where an extrapolation of relationships is required to determine the output. Table 5 shows the *associativity* for the $3x + 7y$ configuration with Kosko's learning algorithm. Data is broken down in terms of "percentage environmental inputs previously observed" so that each column presents data from

Table 6
Associativity in $3x + 28y$, $6x + 28y$ and $12x + 28y$ Kosko's network.

Percentage previously observed	100%	85.7%	71.4%	57.1%
Recall (confidence)	100%	75.6%	45.9%	38.9%
Accuracy	98.9%	62.2%	20.7%	41.0%
Critical (decisions that avoid collision)	0.0%	37.8%	30.7%	6.3%
C+A (critical and correct)	N/A	5.9%	48.5%	100%
R+C (critical decisions claimed to be correct)	N/A	20.6%	35.0%	16.2%
R+A (correct when claimed to be correct)	98.9%	82.3%	42.3%	100%



(a) $3x + 7y$ (b) $3x + 28y$, $6x + 28y$, $12x + 28y$

Fig. 9. Agent trajectory using PRLAB.

its corresponding bar in the bar-chart showing a break-down of environmental inputs (Fig. 8). Therefore, for example, the 100% column in Table 5 corresponds to the 100% bar in Fig. 8a. Since there are no data points in the 71.4% region and below, these columns don't exist in Table 5. "Recall" is a measure of the confidence with which the BAM recalls the values. If a value is perfectly recalled, this would be 100% confidence. The distance from the attractor is used to measure the confidence for almost-recalled values. Thus, given that a BAM is trained using a set of training vector pairs $T = \{X^{(k)}, Y^{(k)}\}_{k=1,\ldots,S}$, each element of set T can be viewed as an attractor. Furthermore, while recalling X, each of $X^{(k)}_{k=1,\ldots,S}$ is an attractor and similarly for Y. Assuming that the BAM is now provided an input $\tilde{X}^{(l)}$ to generate an output $\tilde{Y}^{(l)}$, the confidence of the output $\tilde{Y}^{(l)}$ can be measured as:

$$C = \frac{\lambda - \min\{\text{abs}[\|\tilde{Y}^{(l)} - Y^{(k)}\|_2]_{k=1,\ldots,S}\}}{\lambda}$$

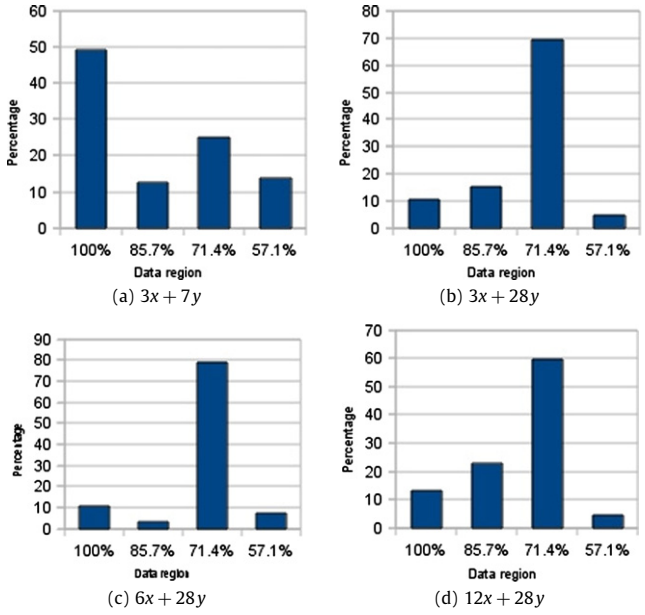
where the subtract operation is performed using Euclidean distance measure for the multi-dimensional vectors Y. λ is the maximum possible distance from the attractor using the current network size and therefore is the lower bound for the confidence measure. For a BAM with n nodes performing recall, it can be represented as $\lambda = 2^n - 1$.

"Accuracy" is measured as the number of memory recalls which matched the outputs of the object-avoidance algorithm. Note that according to this definition, an action may be accurate but still result in a race condition since the object-avoidance algorithm is intentionally simple. This facilitates later versions of PPAM (in the future) attempting to improve upon the behavior of the agent as controlled by the object-avoidance algorithm. "Critical" lists the percentage of memory recalls which were critical to avoiding a collision and "C+A" is the percentage of memory recalls which were critical and also correct. "R+A" measures the percentage of memory recalls (both critical and non-critical) which were recalled with 100% confidence and were also accurate in what they recalled. Therefore, this is a measure of how accurate the memory *thinks* it is, scaled by how accurate it actually was (higher is better). "R+C" measures the number of memory recalls that were recalled with 100% confidence and were critical actions. Therefore this is the same as "R+A" but only for critical actions.

The associativity tables for the $3x + 28y$, $6x + 28y$ and $12x + 28y$ configurations are almost identical and are shown in Table 6.

4.2. PRLAB

The trajectory taken by the agent when it was re-initialized to the original start point after training using PRLAB is shown in Fig. 9. The performance is decidedly better in all configurations, particularly since the agent does not collide with objects no matter what the node configuration. Note however, that increasing the nodes from 7 to 28 deteriorates the performance as the agent now gets stuck in a race condition. This is primarily because of over-training. The $3x + 7y$ training set actually operates on a reduced training set of 18 (unique) pairs. Thus, although the $3x + 28y$ correlation matrix can recall 219 pairs



(a) $3x + 7y$ (b) $3x + 28y$

(c) $6x + 28y$ (d) $12x + 28y$

Fig. 10. Environment input by percentage previously observed for PRLAB.

Table 7
Associativity in $3x + 28y$ using PRLAB.

Percentage previously observed	100%	85.7%	71.4%	57.1%
Recall	98.7%	99.1%	40.7%	66.2%
Accuracy	98.7%	97.7%	34.5%	51.5%
Critical (decisions that avoid collision)	6.6%	88.1%	69.5%	50.0%
C+A (critical and correct)	90.0%	99.0%	36.4%	2.9%
R+C (critical decisions claimed to be correct)	6.0%	88.0%	62.0%	37.8%
R+A (correct when claimed to be correct)	100%	98.6%	14.1%	64.4%

Table 8
Associativity in $6x + 28y$ using PRLAB.

Percentage previously observed	100%	85.7%	71.4%	57.1%
Recall	99.3%	91.1%	67.4%	60.9%
Accuracy	99.3%	95.6%	44.2%	60.9%
Critical (decisions that avoid collision)	6.7%	13.3%	75.0%	78.1%
C+A (critical and correct)	90.0%	100%	50.7%	50.0%
R+C (critical decisions claimed to be correct)	6.0%	9.8%	77.6%	64.1%
R+A (correct when claimed to be correct)	100%	100%	33.4%	100%

Table 9
Associativity in $12x + 28y$ using PRLAB.

Percentage previously observed	100%	85.7%	71.4%	57.1%
Recall	54.8%	99.4%	43.1%	58.5%
Accuracy	54.8%	72.8%	45.9%	47.7%
Critical (decisions that avoid collision)	2.7%	61.1%	82.4%	53.8%
C+A (critical and correct)	80.0%	99.0%	36.2%	2.9%
R+C (critical decisions claimed to be correct)	3.9%	60.9%	71.5%	44.7%
R+A (correct when claimed to be correct)	100%	73.3%	30.6%	57.9%

out of 228 pairs trained, because of the many-to-one relationship between the associative pairs, the BAM gets confused in *that one* critical set of sonar inputs. Further increases in the number of nodes had no effect on the trajectory.

Fig. 10 shows a break-down of the environmental inputs in terms of percentage of the environment observed before. Table 10 shows the *associativity* for the $3x + 7y$ configuration trained using PRLAB while Tables 7, 8 and 9 show the *associativity* for the $3x + 28y$, $6x + 28y$ and $12x + 28y$ configurations respectively.

Table 10
Associativity in $3x + 7y$ using PRLAB.

Percentage previously observed	100%	85.7%	71.4%	57.1%
Recall	100%	100%	100%	100%
Accuracy	74.8%	97.7%	80.7%	97.9%
Critical (decisions that avoid collision)	57.7%	15.7%	58.0%	9.7%
C+A (critical and correct)	56.4%	85.7%	66.7%	78.9%
R+C (critical decisions claimed to be correct)	57.7%	15.7%	58.0%	9.7%
R+A (correct when claimed to be correct)	74.8%	97.7%	80.7%	97.9%

5. Protein Processor Associative Memory

This section describes the architecture and *learning* mechanism of the Protein Processor Associative Memory (PPAM) [30].

5.1. Biological inspiration

Proteins form a critical part of all communication and processing at the cellular level. Cells receive signals in terms of proteins and respond in terms of other proteins. Proteins may cause cells to change their genetic expression or generate other proteins, or both. This forms a complex network of signals called a Genetic Regulatory Network (GRN) [19,16]. At the heart of this GRN is DNA and the process of *transcription*. Details of the processes at the membrane of a cell and within it are extremely complex and many are dependant upon the physical shape and structure of proteins [42,12]. A simplified version that is the source of the inspiration is presented here.

Proteins enter a cell through various channels or receptors and cause (conformational) changes within the cell. If the protein excites a *cis regulatory* region,[2] it initiates the process of transcription where the DNA is read and a protein is encoded. Genes code for a protein or RNA and are composed of *codons* each of which codes for a specific amino acid. Proteins are composed of a variable number of amino acids, which can be as small as 466 (in yeast) or as large as 34,350 (in Titin – the giant protein) [20]. During transcription, the DNA is read serially in a *reading frame* beginning from a start marker called a *start codon* and ending in a stop marker called a *stop codon*. The sequence of information between the start and stop codons is a description of the protein that is to be produced as a result of receiving the original protein at the cell's input. The resultant output protein may be output from the cell or it may become a further input to the cell itself resulting in another iteration of the transcription process. Proteins may also cause a change in genetic expression which means that the cis regulatory regions presented for excitation to incoming proteins can also be altered as a result of processing proteins. Codons are composed of sub-units called *nucleotides* which are labeled U, C, A and G for their chemical names. Codons are tri-nucleotide substances so that a combination of 3 nucleotides makes a codon. This means there are a total of 64 (4^3) codon combinations possible. However, there are only 20 valid amino acids and one start and stop codon. Despite this, all 64 combinations are assigned to either amino acids or start/stop markers. The advantage of this becomes obvious when considering the effects of *point mutations* which alter a single codon. Together with natural selection, which controls the effect of *frame-shift mutations*, this results in a highly robust transcription process. For more details of the transcription process, the reader is referred to Nakamoto [26]; Reil [31].

As expected, with such a transcription process, the resulting DNA inside the cell is extremely long. Therefore, it needs to be compressed so that instead of being more than a meter in length, it can fit inside a few hundred micrometers. This is achieved by winding the DNA strands around a *histone*, resulting in a selected number of *cis regulatory interfaces* that can be excited, thereby allowing a pattern-matching process for incoming proteins. This is the basis of *genetic expression* and *differentiation*. For details of cis regulatory regions and transcription regulation by histones the reader is referred to Wolpert [40]; Zhang and Reinberg [44]; Zhang [43].

5.2. Principle of protein processing

Existing bio-inspired architectures [24,15,29,9] employ reconfigurable fabrics and store a configuration bit stream (DNA) to (re)configure hardware. Other than being an overhead, such a DNA is also not very biologically plausible since DNA (in a biological cell) is not just an initial configuration for the cell but is actively involved in the function of a cell throughout its life (Section 5.1). The transcription process performed inside a biological cell can be considered akin to the looking up of a table of answers. Incoming data is tested to see if any action or data is expected to be produced in response and the expected action or data is looked up in a table or database (DNA) using the incoming data as the unique field for record identification. As can be seen from Section 5.1, this seemingly simple lookup-based processing can actually result in an extremely complex system (GRN) which is capable of very complex (developmental) tasks, without having to *calculate* results. In theory, with unlimited memory for the database, lookup tables can be used to perform any kind of processing. If the lookup table contains all possible "answers" to all possible combinations of incoming data, the cell can be used to

[2] This can be seen as a precondition or a selection process.

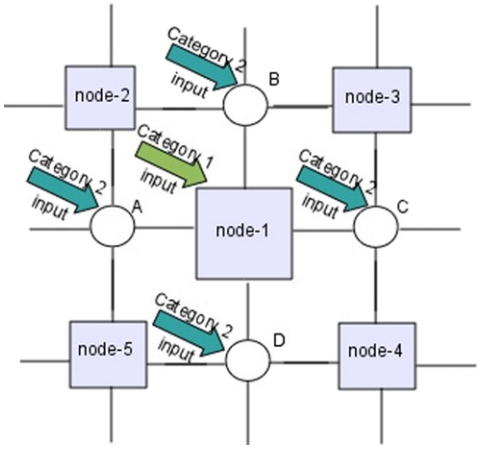


Fig. 11. Abstract view of nodes.

generate any function. In a real world scenario, however, with limited memory only a limited number of "answers" can be stored. As may be seen from Section 5.1, the biological cell also suffers from a similar issue and employs histones to *compress* the data. Future versions of the PPAM architecture are expected to use biologically inspired solutions to compress the memory (or DNA) in each node.

The PPAM is envisioned as a network of nodes, receiving input from multiple categories of inputs that are being associated together. Each node in the network receives one of these (external) inputs along with the outputs from other nodes in its neighbourhood (internal inputs). Fig. 11 illustrates part of a network of 2 types of nodes (the squares and the circles). The squares (node-1, node-2, node-3, node-4, node-5) receive external input from the first category of external input (for example movement values), while the circles (A, B, C, D) receive external input from the second category of external input (for example sonar values). External inputs are indicated with the thick diagonal arrows. In addition, all nodes receive inputs from neighbouring nodes, which are indicated by the thinner, grid lines connecting nodes. Although the nodes illustrated in Fig. 11 are arranged in a grid, note that this topology is presented only to facilitate the concept of neighbourhood and that other network configurations are possible. In our experimental setup, the neighbourhood of each node in one category consists of all nodes of the other category.

Input data from the external world is a time series, which means that it is a sequence of data points sampled at uniform time intervals. In each node, input data (both environment input and neighbouring node input) is looked up in a custom Content Addressable Memory (CAM) – compare with cis regulatory excitation described in Section 5.1. If the data is found in the CAM, a corresponding output value is generated as a result (compare with transcription resulting from cis regulatory excitation described in Section 5.1). From another perspective, the CAM can be likened to a database with two fields, generating an output for the node upon observation of particular inputs. Data is not *calculated* and therefore nodes are free of arithmetic units. Instead, data is considered as abstract symbols and as long as symbols can be uniquely identified, real-world data can be encoded and decoded correctly. Although this is the only encoding *requirement* for the PPAM (unique symbols), in order to enhance robustness encoding can be composed of symbols in redundant combinations. This translates into the following (based on the codon encoding principles outlined in Section 5.1):

Requirement. "*Let data S_1 be defined as being composed of (a subset of) symbols from the set S, with elements $s_1, s_2, s_3, \ldots, s_n$. Let S_1 be encoded into E_1 using (a subset of) symbols from the set E, with elements $e_1, e_2, e_3, \ldots, e_m$. Then there should be multiple encodings $E_1, E_2, E_3, \ldots, E_M$ all using a (possibly different) subset of E that all decode to generate S_1.*"

This has two advantages. Firstly, there is inherent fault-tolerance for transmission errors. Secondly, if decoding results in unknown strings this can be used as a further indication of faults that can be signaled to lower or upper layers (if any). Note that since the PPAM considers input data as abstract symbols, this encoding is not a requirement but rather an optional enhancement.

A node is said to *fire* if the input to the node causes it to generate an output. Firing nodes *self-regulate* which means that they update the tables in their memory in response to the inputs (compare with proteins causing a change in genetic expression by altering the cis regulatory region presented for excitation).

Learning is continuous so that there is no separate, distinct training phase. Memory recall is an asynchronous operation while learning is synchronous to avoid the requirement of asynchronous RAMs in hardware. If a node does not receive an input from the environment, it looks up the neighbourhood input word in the CAM and determines what (if any) output value is to be generated. Therefore in the absence of one category of environment input, each node recalls a view of its neighbours which is a partial view of the relationship between the two categories of inputs. Together, all the nodes reconstruct the "complete picture".

Table 11
Learning values in CAM.

Input pattern	Output value
0x0001	0x5
0x0002	0x6
0x0004	0x7

(a) Initial values in CAM.

Input pattern	Output value
0x0001	0x5
0x0002	0x6
0x0004	0x7
.	.
.	.
0xBDCA	0x5

(b) Values in CAM after regulation.

Table 12
Relationship between 2 multi-dimensional variables.

A_{D1}	A_{D2}	B_{D1}	B_{D2}	B_{D3}
3	0	5.0	5.0	5.0
0	−1.5	0.5	1.0	0.75
3	−0.25	3.5	4.0	4.75

5.3. Learning mechanism

Learning is based on the Hebbian principle [11,6]. A firing node records a snapshot of its neighbours, indicating which neighbours fired in conjunction with itself. To illustrate, let the CAM in node-1 (Fig. 11) contain values as shown in Table 11a. Let the environmental inputs be such that when 0x0001[3] is input to node-1, the environment inputs for its neighbouring nodes (the circular nodes) make them fire with values A, B, C and D. Then node-1 fires with value 0x5 and observes 0x$BDCA$ from its neighbours (ordering the data North–South–East–West). It therefore adds a tuple to its CAM to reflect this view of its neighbourhood, resulting in a table as shown in Table 11b. Note that the symbol set being used by external inputs (including all categories) must never overlap with the symbol set of the word generated from the neighbourhood inputs. For instance, if the concatenated word from the neighbours of node-1 is 0x0004 (instead of 0x$BDCA$), this could add a tuple to the table where the *input pattern* field contains the word 0x0004 which is the same value as the third tuple in the table. Thus a fallacious conflicting record would be entered into the table which does not stem from any lack of relationship between inputs but an overlap of symbols.

5.4. Resolving conflicts

Conflicting tuples can result if input variables are not related, or if the relationship is one-to-many (or many-to-many or many-to-one). In this case, nodes observe the same neighbourhood inputs in conjunction with different output values from the node itself. Consider two multi-dimensional variables, A and B, that have a one-to-one relationship as shown in Table 12. Although A (A_{D1}, A_{D2}) has a one-to-one relationship with B (B_{D1}, B_{D2}, B_{D3}), this does not imply that A_{Di} also has a one-to-one relationship with any one of B_{Dj} or any combination of them. This is evident in Table 12. The memory in an individual node represents a portion of the relationship between one dimension of one category (for example A_{Di}) and the dimensions of the other category (for example B_{Dj}). This means that even when there is a one-to-one mapping in the dataset, the relationship in the nodes may be otherwise, resulting in conflicting tuples. However, this has the advantage that extrapolating from this information does not necessitate solving mathematical equations which require ALUs. Conflicts are resolved by a swapping process illustrated in Algorithm 2. Over time, the tuples observed most often will move to the top and will be arranged in the order of the most frequent to the least frequent. The least frequent tuples are the first to be removed when a node runs out of memory.

Note that during recall, if variable B has value $(5.0, 5.0, 4.0)$, from the perspective of the PPAM, this is equivalent to $(5.0, 5.0, x)$ where x can take any value not observed before by the CAM – which would therefore not be present in any of the nodes. Thus, this would result in hetero-associative recall. If, on the other hand, the variable B has value $(5.0, 5.0, 5.0)$ then this is auto-associative recall, since B is fully observed before and considering one tuple in Table 12 as one data point, the complete data point would have to be recalled from partial information.

5.5. Comparing the structure of BAM with PPAM

Table 13 compares the structure and the kind of operations required for a BAM and for the PPAM. BAM implementations are assumed to be optimized such that calculations are not repeated and results are considered to be stored in temporary memory (CPU registers). Memory access (read/write) does not include access to these temporary locations or to

[3] 0x indicates hexadecimal notation.

Algorithm 2: Pseudo-code for frequency detection algorithm

```
Input: neighbourhood firing pattern p_n, current node firing value f, memory M of size 2 × N
// M[0][x] is neighbourhood pattern at memory location x
// M[1][x] is firing value in CAM at memory location x
// j stores index of record to swap down
j ← unique init value;
for i ← 0 to N do
    if p_n == M[0][i] then
        if f == M[1][i] then
            if j! = unique init value then
                swap tuple at i with tuple at j;
                break loop;
        else
            j ← i
if j == unique init value then
    add new tuple at end of M – location x;
    M[0][x] ← p_n;
    M[1][x] ← f;
```

Table 13
Comparison of structure of BAM and PPAM.

	BAM	PPAM
Connections	M nodes connected to N and N nodes connected to M	M nodes connected to N and N nodes connected to M
Memory requirements	$(M \times N) + M + N$ floating point numbers	worst case – $S \times (M+N)^2$ bits
Encoding requirements	bipolar preferred	none
Operations per each of N nodes during recall		
Multiplications	M	0
Additions	$M+1$	0
$>$ Comparisons	1	0
$<$ Comparisons	1	0
$==$ Comparisons	0	up to S
Iterations	till stable	till stable
Memory reads	$M+1$	S
Operations per each of N nodes during store		
Multiplications	$(2 \times M) + 3$	0
Additions	$(2 \times M) + 2$	0
$>$ Comparisons	0	0
$<$ Comparisons	1	0
$==$ Comparisons	1	S
Iterations	Epochs $\times S$	1
Memory reads	$M+3$	S
Memory writes	$M+1$	1

any other *working* memory. Furthermore, memory requirements (or operations) for stimulating nodes or providing input are not included. In addition, the multiplication and addition operations for calculating indexes of the weight-threshold matrix for the BAM are also not included in Table 13 because this is considered to be similar to the significantly simpler and smaller number of memory index counting operations for the PPAM. Table 13 is generated for storing a set of vector pairs $T = \{X^{(k)}, Y^{(k)}\}_{k=1,\ldots,S}$. For the BAM, $X^{(k)} \in \{-1,+1\}^N$ and $Y^{(k)} \in \{-1,+1\}^M$, while for the PPAM $X^{(k)} \in \{0,1\}^N$ and $Y^{(k)} \in \{0,1\}^M$.

Both BAM and PPAM require fully connected networks, where M nodes have N connections and N nodes have M connections. The BAM requires one $M \times N$ connection weight matrix storing floating point numbers, even if the data is not real-valued (as in this case). In addition, it also requires one floating point threshold value for each node. On the other hand, each node of the PPAM requires an M bit wide memory for Y and an N bit wide memory for X, each of which could be up to S locations deep in the worst case. It is likely to be much lower and in the experiments conducted, the worst case observed was 72 locations for one node while the average requirement was 12 and the minimum was 1 location (where $S = 228$). Depending on the encoding, there can be $M \times N$ nodes in the network. Details of the calculations required to recall values and store associative pairs for the BAM and using PRLAB can be found in Kosko [18]; Oh and Kothari [27]. Note that all memory variables in the BAM are real values and therefore all BAM calculations require floating point arithmetic. Details of the operations for recalling values and storing pairs in the PPAM can be found in Section 5.

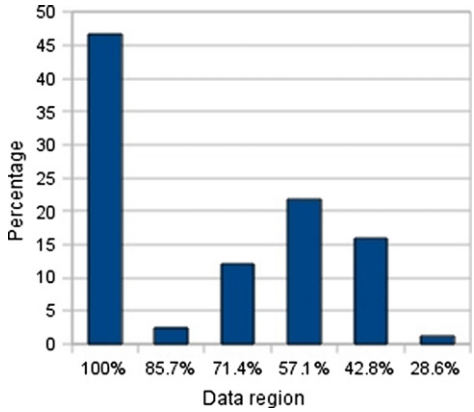


Fig. 12. Environment input by percentage previously observed for $3x + 35y$ PPAM.

6. PPAM experiments and results

6.1. Experiment setup

Although non-binary nodes are also possible in a PPAM, this implementation uses binary nodes. This not only simplifies the implementation, it also reduces the number of connections between nodes, since binary outputs require 1-bit connections. This is particularly useful from a hardware perspective.

The same experiments as performed for the BAM (Section 3) were repeated using the PPAM. The $3x + 14y$ PPAM node configuration is equivalent to the $3x + 7y$ BAM configuration because of the following. The 14 (PPAM) nodes may be divided into two sets of 7 each. The same seven sonar values are input to each set so that two nodes (one from each set) receive values from one sonar. The nodes are configured so that one and only one of these two nodes will *fire* (generate an output). Since the two nodes are mutually exclusive, two nodes represent one binary value. Therefore, 14 nodes can distinguish between 2^7 binary values – which is the same as for the $3x + 7y$ BAM configuration. Similarly, the 70 nodes in the $3x + 70y$ PPAM network can be divided into 10 sets of 7 each, with each set receiving the same 7 sonar values. The 10 nodes in each set being mutually exclusive, each set represents one decimal value, thus the 70 nodes can distinguish between 10^7 values which is the same as the $3x + 28y$ BAM configuration. One consideration for simulating BAM using the $6x + 28y$, $12x + 28y$, $12x + 63y$ and $30x + 28y$ node configurations was to test whether this had any effect on performance, since, like the corresponding PPAM implementation, these configurations also use more bits than is necessary to represent sonar values and movement values.

6.2. Results

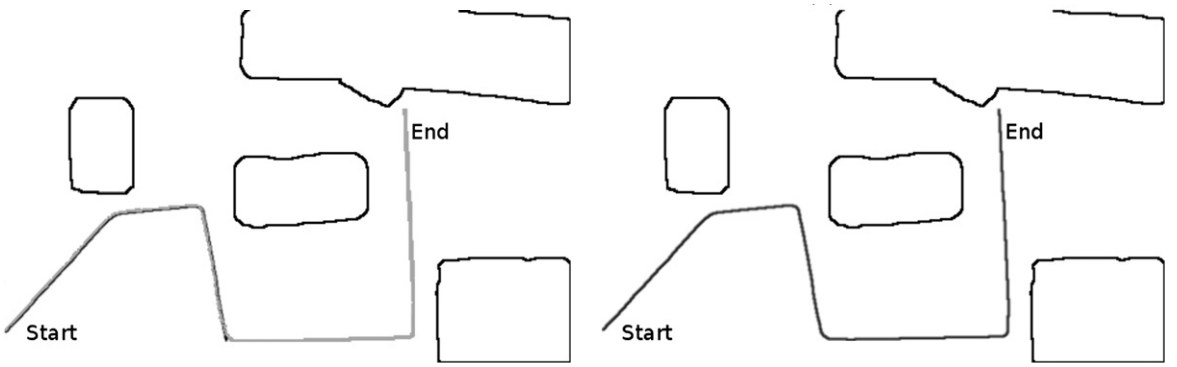
Whereas the process of tuning the correlation matrix for the BAM revealed that it was not possible to store all the associative pairs in the BAM, some PPAM node configurations ($3x + 28y$ and above) were successfully able to recall all 228 associative pairs in the training dataset. Thus, the first phase of the experiment (Section 3.4) would be trivial (for higher PPAM node configurations), however the experiment was performed for verification and results are presented in order to do a full comparison.

Fig. 13 shows the trajectory taken by the agent when it was re-initialized to the original start point while being controlled by the PPAM. As can be seen from the figure, although the $3x + 14y$ PPAM network suffered from getting stuck in the race condition, all other network configurations followed the trajectory of the original object-avoidance algorithm exactly. Only results for the $3x + 35y$ configuration are reproduced here in order to offer a comparison. More details of all the results and experiments can be found in Qadir et al. [30]. Fig. 12 shows a break-down of the environmental inputs in terms of the environment having been observed before. Table 14 measures the associativity of the $3x + 35y$ PPAM.

For the case of the PPAM, confidence is an output of the memory and is directly proportional to the portion of the neighbourhood input that was found in the memory. If a node has a neighbourhood of N nodes, it receives a total N words from its neighbours as input. If during a recall operation, the node is able to locate F of these words in its memory, then the confidence is simply $C = F/N$. Note that this does not imply a division operation within each node, because the nodes only output the value F. The division is performed only to normalize the output and can be implemented (if required) in a companion controller based on the more traditional ALU-based architecture.

6.3. Fault-tolerance

The $3x + 60y$ PPAM node configuration was simulated to test the case where a $3x + 70y$ node configuration network has 10 nodes broken. Note that this is the same as the case where a $3x + 70y$ network is trained and then 10 nodes stop firing



(a) $3x + 14y$ (b) $3x + 28y$, $3x + 35y$, $3x + 70y$

Fig. 13. Agent trajectory using PPAM.

Table 14
Associativity in $3x + 35y$ PPAM.

Percentage previously observed	100%	85.7%	71.4%	57.1%	42.8%	28.6%
Recall	100%	58.3%	24.8%	18.6%	15.8%	68.7%
Accuracy	100%	97.2%	80.9%	77.5%	83.8%	68.7%
Critical (decisions that avoid collision)	15.4%	0.0%	29.5%	13.2%	12.7%	0.0%
C+A (critical and correct)	100%	N/A	54.9%	56.1%	58.6%	N/A
R+C (critical decisions claimed to be correct)	15.4%	N/A	2.3%	3.4%	5.5%	N/A
R+A (correct when claimed to be correct)	100%	100%	100%	100%	100%	100%

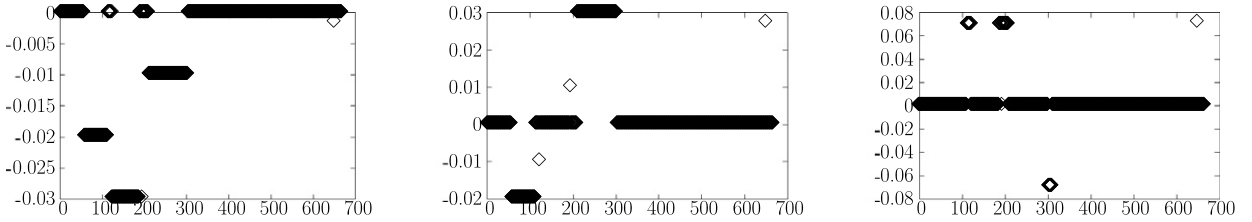


Fig. 14. Error in path following in $3x + 60y$ network.

Table 15
Associativity in $3x + 60y$ network.

Percentage previously observed	100%	85.7%	71.4%	57.1%	42.8%	28.6%
Recall	100%	58.3%	24.8%	18.7%	15.7%	68.7%
Accuracy	100%	97.2%	80.9%	77.4%	83.0%	68.7%
Critical (decisions that avoid collision)	15.5%	0.0%	29.5%	13.2%	14.0%	0.0%
C+A (critical and correct)	100%	N/A	54.9%	56.1%	58.2%	N/A
R+C (critical decisions claimed to be correct)	15.5%	N/A	2.3%	3.4%	5.6%	N/A
R+A (correct when claimed to be correct)	100%	100%	100%	100%	100%	100%

or the 10 nodes get stuck at 0 (where 0 represents not firing in this encoding). This is because during the lookup process in each node, a non-firing node is the same as a node firing with a value not seen before which would not be found in the CAM. The error between the robot's path using the innate algorithm and the path using PPAM is shown in Fig. 14, where the y-axis is the error and the x-axis is the discrete time steps. As can be seen, the error is quite small – so small in fact that the difference between the two paths cannot be detected from observation with the naked eye. Table 15 measures the associativity of the PPAM. The important thing to note is that the associativity is almost exactly the same as for the $3x + 35y$ configuration.

7. Discussion

When comparing the associativity, all results must be biased according to the number of vectors present in that category (which can be seen from the bar-chart with the break-down of the environment). Furthermore, for the case when the environmental input has been fully observed (100% column), an ideal associative memory should be able to recall the correct output *every time* and with *full* confidence. Otherwise, it means that the memory has forgotten the training pair.

Even if some of the pairs have been forgotten, the memory should at least be able to correctly indicate a confidence level with the value recalled. The "R+A" values are critical since they measure how accurate the memory *thinks* it is, scaled by how accurate it actually was (higher is better).

7.1. Comparing Kosko's method with PRLAB

Comparing the "accuracy" field in the associativity tables (Tables 6–9) it is evident that PRLAB is more accurate. The two anomalous cases (100% columns in $3x + 28y$ and $6x + 28y$ configurations) are unreliable because the number of test vectors in these regions is very low (less than 10% – Figs. 8 and 10). In none of the node configurations, with either Kosko's original method or using PRLAB was the memory able to successfully recall all the training pairs. This was expected from the results of the experiments conducted to train the correlation matrix (Section 3.3). Using Kosko's method, it can be seen that even for the 100% observed case, a value recalled with complete confidence could still be wrong ("R+A" field in the 100% observed column in associativity tables is less than 100%). Although the $3x + 7y$ PRLAB node configuration also suffered from the same problem, higher PRLAB node-configurations overcame this and if a value recalled was claimed to be correct, it *was* correct. Furthermore, in general, the "R+A" measure for PRLAB was higher than the "R+A" measure for Kosko's method. For the one anomalous case where "R+A" for the 71.4% column in Table 6 is higher than its corresponding PRLAB associativity tables, note that it is best to compare the $6x + 28y$ PRLAB configuration with the $6x + 28y$ Kosko configuration since these are closest in terms of environment input break-down (79.06% and 83.74%). Comparing these two, even though the PRLAB measures at 33.4% "R+A" compared to Kosko's 42.3%, bearing in mind that Kosko's method is only confident about 45.9% of the recalls, while PRLAB is confident about 67.4% of the recalls, the PRLAB result is still better. In addition, the agent controlled by PRLAB was able to follow the trajectory better than the agent controlled by Kosko's algorithm (Figs. 7 and 9). From the above comparison of Kosko's method with PRLAB, it can be seen that PRLAB performs better, not only in terms of the number of associative pairs recalled (Section 3.3) but also in terms of the agent's behaviour in a real-time environment. Therefore, it should be sufficient to compare PPAM with PRLAB.

7.2. Comparing PRLAB with PPAM

The trajectory of the agent controlled using the $3x + 7y$ PRLAB configuration is almost identical to the trajectory of the agent controlled using the $3x + 14y$ PPAM configuration. However, for higher configurations, the performance of the PPAM is significantly better. Furthermore, from the associativity in Table 14, it can be seen that in higher configurations, the accuracy of the PPAM for the 100% previously observed sonar values was perfect. Also, the "R+A" field indicates that whenever the PPAM indicated 100% confidence in the value recalled, it would always be the correct value. This is true not only for the 100% previously observed sonar values, but even when only 28.6% of the environmental input had been previously observed (one out of 7 sonar values).

From the perspective of the PPAM, inputs with less than 100% observed sonar values are exactly the same as 100% observed values, corrupted with noise. Therefore, less than 100% observed sonar values trigger a hetero-associative recall. This is because, although the remaining sonar inputs actually have a value (as a bus in hardware will always have *some* value), from the perspective of the architecture, this is equivalent to the inputs being absent, since values not seen before are all treated the same. Since the BAM does not operate on abstract symbols but uses arithmetic calculations, a proper noise test would involve using a random noise pattern based on a known distribution. Nonetheless, for the following discussion, the aforementioned, much simpler noise test is considered where the performance of the BAM is expected to be better than with the random noise source. In addition, a more accurate estimate of the performance of BAM in the presence of noise is included from Sudo et al. [35].

As can be seen from the results, the PPAM is more noise tolerant. Although the $3x + 70y$ PPAM configuration is the equivalent of the $3x + 28y$ BAM node configuration, comparing the $3x + 35y$ PPAM with any of $3x + 28y$, $6x + 28y$ or $12x + 28y$ BAM, PPAM has an accuracy of 80.9% when noise reaches 28.6% while that of the BAM is 45.9% at best. This is one of the two types of noise considered by Sudo et al. [35] – namely, "noise-added original patterns". For the case of random noise (from Sudo et al. [35]), when the noise level reaches 15%, accuracy drops to approximately 95% for SOIAM, 85% for KFMAM-FW (Kohonen Feature Map Associative Memory with Fixed Weights), 60% for KFMAM with batch learning, 38% for KFMAM with sequential learning and 0% for BAM with batch or sequential learning. Our implementation of BAM with PRLAB uses a redundant memory to store all the associative pairs. Although this defeats the purpose of having an online associative memory, it does result in the BAM displaying a much higher level of noise tolerance – up to 97.7% accuracy in the best case and 72.8% in the worst case (with 15% noise). The accuracy for the $3x + 35y$ PPAM with 15% noise is 97.2% (Table 14) and for the $3x + 70y$ PPAM (the equivalent of the SOIAM and $3x + 28y$ and higher BAMs) is 99.1% [30]. When noise is at 28%, SOIAM has less than 75% accuracy and all others are lower than 30% (from Sudo et al. [35]). Our implementation of the BAM with PRLAB has at best 46% and at worst 35% accuracy. PPAM, on the other hand, has 81% accuracy for the $3x + 35y$ configuration and 90% accuracy for the $3x + 70y$ case. Note that an increase in number of nodes always results in a corresponding increase in performance, indicating that the PPAM architecture does not easily succumb to issues of over-training.

The second type of noise indicated by Sudo et al. [35] is "faultily presented random patterns" that must be identified as noise or unknown patterns. In the SOIAM, this is achieved by generating an "unknown" output. In the PPAM, this is

(a) E-Puck Mobile Robot (b)E-Puck Maze

Fig. 15. E-Puck platform.

achieved by the confidence value in the output. Whatever pattern is presented as input, the PPAM generates an output and a confidence measure. In cases where the input pattern is unrecognisable, the confidence measure is zero or almost zero. Note that results reported in Sudo et al. [35] do not include "R+A". To our knowledge, results reported in other existing literature on associative memories also do not include any measure equivalent to "R+A", which measures the *claimed* accuracy of recall against an *absolute* accuracy of recall. In our opinion, such a measure is critical to determining performance of associative memories.

8. Conclusion

A mobile robot in the player/stage environment was controlled using Bidirectional Associative Memory and also Protein Processor Associative Memory. Two training algorithms were attempted for the BAM – the original learning algorithm presented by Kosko [18] and also the more popular PRLAB [27] that guarantees recall of all training pairs if it is possible to store them in a BAM correlation matrix. Parameters were tuned in an attempt to maximize memory utilization of BAMs. Although the PRLAB is relatively insensitive to initial parameter values (as claimed), we found that these values can, in the worst case (when the dataset is not orthogonal), drop the performance of the PRLAB to the level of the original Kosko's algorithm. Various network configurations for the BAM were attempted with number of nodes being selected based on the level of quantization desired for input pairs. However, none of the configurations were successfully able to recall *all* associative pairs despite tuning the parameters. This is because of the fact that BAMs require associative pairs to be orthogonal. Therefore, although increasing the number of nodes in the 2 layers increased the capacity of the BAM, it was still unable to store and recall all the pairs. In fact, in some cases, increasing the number of nodes had the opposite effect because the associative pairs became *less orthogonal* because of the new encoding required by a higher node count in each layer. In order to store all pairs in a BAM, either the data should be re-encoded to become orthogonal ([34], cited in [27]) or the training dataset should be parsed to select critical data points that are orthogonal and also sufficient to describe the underlying distribution/algorithm. For both solutions, this means that generating the training dataset for a BAM requires *a priori* knowledge of the associative pairs, which is not a realistic requirement for online, real-time learning environments. Furthermore, there is no such requirement for the PPAM.

Equivalent PPAM and BAM configurations were attempted as controllers for the agent and the PPAM was shown to perform better in all cases, displaying higher accuracy and noise tolerance. Results were also compared with SOIAM and the PPAM was shown to be more tolerant to both types of noise identified in Sudo et al. [35], thus achieving higher accuracy. The advantages of the player/stage environment were utilized in transferring the player implementation to the E-Puck mobile robot [25] and the results were verified on the embedded environment. A close-up of the E-Puck in its maze is shown in Fig. 15a while an overall view of the robot in the maze can be seen from Fig. 15b.

The PPAM architecture described in Section 5 has been shown to be better at associativity and noise tolerance. It has also been shown to be fault-tolerant so that a node configuration of $3x + 70y$ degrades gracefully even with 10 nodes *broken* [30]. Furthermore, the PPAM architecture achieves this without the use of arithmetic operations. Nonetheless, there is significant room for improvement, particularly since the architecture has issues for hardware design. These are outlined in Qadir et al. [30]. A future version of the PPAM is being attempted which is expected to retain (or improve) the level of associativity and address these hardware issues so that the architecture is more suitable for a hardware implementation. In addition, there are some parallels which can be drawn between the PPAM and the Hamming Associative Memory ([10], cited in [14]) and future implementations are expected to be compared against the performance of these as well.

Acknowledgements

The authors would like to thank their partners at BRL-UWE.[4] The research is funded by the EPSRC funded SABRE[5] project[6] under grant no. FP/F06219211.

References

[1] M.E. Acevedo-Mosqueda, C. Yáñez-Márquez, I. López-Yáñez, Alpha–beta bidirectional associative memories based translator, International Journal of Computer Science and Network Security 6 (2006) 190–194.
[2] J. Amaral, J. Ghosh, An associative memory architecture for concurrent production systems, in: Proc. IEEE International Conference on Systems, Man, and Cybernetics, Humans, Information and Technology 3 (1994) 2219–2224.
[3] G. Anzellotti, R. Battiti, I. Lazzizzera, G. Soncini, A. Zorat, A. Sartori, G. Tecchiolli, P. Lee, Totem: a highly parallel chip for triggering applications with inductive learning based on the reactive tabu search, International Journal of Modern Physics C 6 (1995) 555–560.
[4] J. Cao, J. Liang, J. Lam, Exponential stability of high-order bidirectional associative memory neural networks with time delays, Physica D: Nonlinear Phenomena 199 (2004) 425–436.
[5] S. Chen, H. Gao, W. Yan, Improved exponential bidirectional associative memory, Electronics Letters 33 (1997) 223–224.
[6] B. Coppin, Neural Networks, 1st ed., Jones and Bartlett Publishers, Inc., USA, 2004, pp. 291–326 (Chapter 11).
[7] T.-D. Eom, S.-K. Oh, J.-J. Lee, Guaranteed recall of all training pairs for exponential bidirectional associative memory, Electronics Letters 37 (2001) 153–154.
[8] B.P. Gerkey, R.T. Vaughan, A. Howard, The player/stage project: Tools for multi-robot and distributed sensor systems, in: Proceedings of the 11th International Conference on Advanced Robotics, 2003, pp. 317–323.
[9] A.J. Greenstead, A.M. Tyrrell, Extrinsic evolvable hardware on the RISA architecture, in: Proceedings of ICES 2007, 7th International Conference on Evolvable Hardware, 2007, pp. 244–255.
[10] M. Hassoun, P. Watta, The hamming associative memory and its relation to the exponential capacity dam, in: IEEE International Conference on Neural Networks, vol. 11, 1996, pp. 583–587.
[11] D.O. Hebb, The Organization of Behavior: A Neuropsychological Theory, new ed., Wiley, New York, 1949.
[12] P. Hogeweg, Computing an organism: on the interface between informatic and dynamic processes, in: S. Kumar, P.J. Bentley (Eds.), On Growth, Form and Computers, 1st ed., Academic Press, Amsterdam, London, 2003.
[13] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, in: Proceedings of the National Academy of Sciences of the United States of America, vol. 79, 1982, pp. 2554–2558.
[14] N. Ikeda, P. Watta, M. Artiklar, M.H. Hassoun, A two-level hamming network for high performance associative memory, Neural Networks 14 (2001) 1189–1200.
[15] A.H. Jackson, A.M. Tyrrell, Asynchronous embryonics with reconfiguration, in: Proceedings of 4th International Conference on Evolvable Systems, vol. 2210, Springer, Berlin/Heidelberg, 2001, pp. 88–99.
[16] H.D. Jong, J. Geiselmann, D. Thieffry, Qualitative modeling and simulation of developmental regulatory networks, in: S. Kumar, P.J. Bentley (Eds.), On Growth, Form and Computers, 1st ed., Academic Press, Amsterdam, London, 2003.
[17] T. Kohonen, Correlation matrix memories, IEEE Transactions on Computers C 21 (1972) 353–359.
[18] B. Kosko, Bidirectional associative memories, IEEE Transactions on Systems, Man, and Cybernetics 18 (1988) 49–60.
[19] S. Kumar, P.J. Bentley, An introduction to computational development, in: S. Kumar, P.J. Bentley (Eds.), On Growth, Form and Computers, 1st ed., Academic Press, Amsterdam, London, 2003.
[20] S. Labeit, B. Kolmerer, Titins: Giant proteins in charge of muscle ultrastructure and elasticity, Science 270 (1995) 293–296.
[21] P. Lee, E. Costa, S. McBader, L. Clementel, A. Sartori, LogTOTEM: A logarithmic neural processor and its implementation on an FPGA fabric, in: International Joint Conference on Neural Networks, 2007, IJCNN 2007, 2007, pp. 2764–2769.
[22] S.W. Lee, J.T. Kim, H. Wang, D.J. Bae, K.-M. Lee, J.-H. Lee, J.W. Jeon, Architecture of RETE network hardware accelerator for real-time context-aware system, in: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), KES (1), in: Lecture Notes in Computer Science, vol. 4251, Springer, 2006, pp. 401–408.
[23] J. Lu, J. He, J. Cao, Z. Gao, Topology influences performance in the associative memory neural networks, Physics Letters A 354 (2006) 335–343.
[24] D. Mange, M. Sipper, A. Stauffer, G. Tempesti, Toward robust integrated circuits: The embryonics approach, Proceedings of the IEEE 88 (2000) 516–543.
[25] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in: P. Gonçalves, P. Torres, C. Alves (Eds.), Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, IPCB: Instituto Politécnico de Castelo Branco, Portugal, 2009, pp. 59–65.
[26] T. Nakamoto, Evolution and the universality of the mechanism of initiation of protein synthesis, Gene 432 (2009) 1–6.
[27] H. Oh, S. Kothari, Adaptation of the relaxation method for learning in bidirectional associative memory, IEEE Transactions on Neural Networks 5 (1994) 576–583.
[28] J. Owen, How to use player/stage, On player/stage website, http://playerstage.sourceforge.net, 2009.
[29] L. Prodan, G. Tempesti, D. Mange, A. Stauffer, Embryonics: electronic stem cells, in: ICAL 2003: Proceedings of the Eighth International Conference on Artificial Life, MIT Press, Cambridge, MA, USA, 2003, pp. 101–105.
[30] O. Qadir, J. Liu, J. Timmis, G. Tempesti, A. Tyrrell, Principles of protein processing for a self-organising associative memory, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, 2010.
[31] T. Reil, Artificial genomes as models of gene regulation, in: S. Kumar, P.J. Bentley (Eds.), On Growth, Form and Computers, 1st ed., Academic Press, Amsterdam, London, 2003.
[32] G.X. Ritter, J.L. Diaz-de Leon, P. Sussner, Morphological bidirectional associative memories, Neural Networks 12 (1999) 851–867.
[33] D. Shen, J.B. Cruz, Encoding strategy for maximum noise tolerance bidirectional associative memory, IEEE Transactions on Neural Networks 16 (2005) 293–300.
[34] P. Simpson, Bidirectional associative memory systems, Technical Report GDE-ISG-PKS-02 General Dynamics Electronics Div., 1988.
[35] A. Sudo, A. Sato, O. Hasegawa, Associative memory for online learning in noisy environments using self-organizing incremental neural network, IEEE Transactions on Neural Networks 20 (2009) 964–972.
[36] T. Tanaka, S. Kakiya, Y. Kabashima, Capacity Analysis of Bidirectional Associative Memory, 2000.

[4] Bristol Robotics Lab University of Western England.
[5] Self-healing cellular Architectures for Biologically-inspired highly Reliable Electronic systems.
[6] http://www.brl.ac.uk/projects/sabre/index.html.

[37] J. Teich, Invasive algorithms and architectures (Invasive Algorithmen und Architekturen), Information Technology 50 (2008) 300–310.
[38] T. Toffoli, CAM: A high-performance Cellular Automaton Machine, Physica D 10 (1984) 195–204.
[39] B. Wang, G. Vachtsevanos, Storage capacity of bidirectional associative memories, in: Proc. IEEE International Joint Conference on Neural Networks, vol. 2, 1991, pp. 1831–1836.
[40] L.L. Wolpert, Relationships between development and evolution, in: S. Kumar, P.J. Bentley (Eds.), On Growth, Form and Computers, 1st ed., Academic Press, Amsterdam, London, 2003.
[41] Y. Wu, D. Pados, A feedforward bidirectional associative memory, IEEE Transactions on Neural Networks 11 (2000) 859–866.
[42] S. Zhan, J.F. Miller, A.M. Tyrrell, A developmental gene regulation network for constructing electronic circuits, in: G. Hornby, L. Sekanina, P.C. Haddow (Eds.), ICES, in: Lecture Notes in Computer Science, vol. 5216, Springer, 2008, pp. 177–188.
[43] Y. Zhang, Transcriptional regulation by histone ubiquitination and deubiquitination, Genes & Development 17 (2003) 2733–2740.
[44] Y. Zhang, D. Reinberg, Transcription regulation by histone methylation: interplay between different covalent modifications of the core histone tails, Genes & Development 15 (2001) 2343–2360.
[45] G. Zheng, S. Givigi, W. Zheng, A new strategy for designing bidirectional associative memories, in: J. Wang, X. Liao, Z. Yi (Eds.), Advances in Neural Networks – ISNN 2005, in: Lecture Notes in Computer Science, vol. 3496, Springer, Berlin/Heidelberg, 2005, pp. 398–403.