2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Robot Navigation Using a Sparse Distributed Memory

Mateus Mendes
ESTGOH, IPC Coimbra
ISR - Institute of Systems and Robotics
Dep. of Electrical and Computer Engineering
University of Coimbra
Portugal
mmendes@estgoh.ipc.pt

Manuel Crisóstomo
ISR - Institute of Systems and Robotics
Dep. of Electrical and Computer Engineering
University of Coimbra
Portugal
mcris@isr.uc.pt

A. Paulo Coimbra
ISR - Institute of Systems and Robotics
Dep. of Electrical and Computer Engineering
University of Coimbra
Portugal
acoimbra@deec.uc.pt

***Abstract*— Despite all the progress that has been made in Robotics and Artificial Intelligence, traditional approaches seem unsuitable to build truly *intelligent* robots, exhibiting human-like behaviours. Many authors agree that the source of *intelligence* is, to a great extent, the use of a huge memory, where sequences of events that guide our later behaviour are stored. Inspired by that idea, our approach is to navigate a robot using sequences of images stored in a Sparse Distributed Memory—a kind of associative memory based on the properties of high dimensional binary spaces, which, in theory, exhibits some human-like behaviours. The robot showed good ability to correctly follow most of the sequences learnt, with small errors and good immunity to the kidnapped robot problem.**

***Index Terms*— robot navigation, view sequence, sparse distributed memory**

I. INTRODUCTION

Over the last 50 years Artificial Intelligence (AI) and Robotics have gone through many peaks and ebbs, and many approaches have been tried to build truly *intelligent* machines. The ideal model, though, has yet to be found. It is possible that a good model arises from the study of the human brain, which is not, itself, very well understood. According to recent evidence, though, it is believed that the human brain is essentially a large memory system [1], [2], [3]. J. Hawkins [1] proposes the Memory Prediction Framework, a model which states that the brain is continuously making predictions about the world. When one of those predictions is violated, the brain learns, adjusting its memories according to the new data. The memory seems to be organised in a hierarchy, each level being responsible for learning a small part of the overall model.

Hawkins' work was a new approach to renew AI, but it was initially published as just an idea. Research is still being done towards formalisation of the model [4]. On the other hand, there's a sound mathematical model available that, in theory, offers many of the characteristics that a human memory exhibits: Kanerva [2] created a Sparse Distributed Memory (SDM) model, and developed the mathematical support to implement it.

D. Rogers [5], A. Anwar et al [6], R. Rao and D. Ballard [7], Furber et al [8], [3], among others, have implemented SDMs and improved the original model, but the SDM has never been pushed farther, despite some preliminary results. The lack of interest in this idea is not clear. One reason might be that the average storage capacity starts at about 0.1 bits per bit of traditional memory. Another may be the speed of operation, for such a memory has to be implemented using neural networks or linked lists. Anyway, the practical implementations have worked as the theory predicted, and the original idea is still promising, according to the recent results from neuroscience on the principles of human intelligence.

Our approach is to navigate a robot (called Lizard) using an SDM at its core. We describe the architecture of an SDM suitable to store and predict sequences of images, and the software model designed to navigate the robot. Lizard has two operating modes: one for learning, the other to navigate autonomously. During learning it acquires and stores in the SDM images and some additional information, such as the timestamp and the motion the robot was performing. During the autonomous run, Lizard captures its current view and uses it to retrieve the closest image from the SDM. It then uses the data associated with the retrieved image to predict the next action that must be performed. Small drifts are corrected based on the horizontal displacement between the retrieved image and Lizard's current view. This approach is inspired by the one developed by Y. Matsumoto [9].

II. SPARSE DISTRIBUTED MEMORIES

The underlying idea behind an SDM is the mapping of a huge binary memory onto a smaller set of physical locations, so-called *hard locations*. As a general guideline, those hard locations should be uniformely distributed in the virtual space, to *mimic* the existence of the larger virtual space as accurately as possible. Every datum is stored distributed by a set of hard locations, and retrieved by *averaging* those locations. Therefore, recall may not be perfect, accuracy depending on the saturation of the memory.

Kanerva's proposal is based on four basic ideas, as presented by the author: the space 2^n, for $100 < n < 10^5$, exhibits properties which are similar to our intuitive notions of relationships between the concepts; neurons with n inputs can be used as address decoders of a random-access

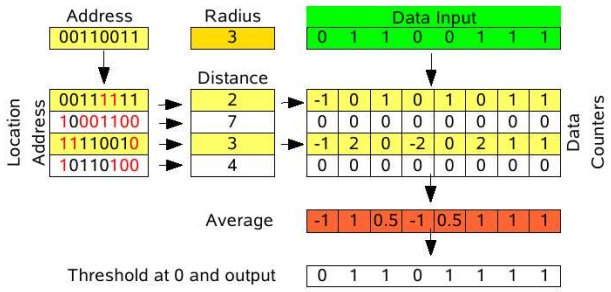

Fig. 1. One model of an SDM.

memory[1]; unifying principle: data stored in the memory can be used as addresses to the same memory; time can be traced in the memory as a function of where the data are stored.

In [2] and [11], Kanerva and Mendes et al present thorough demonstrations of how those properties are guaranteed by an SDM. Therefore, we will only concentrate on the implementation details.

A. A Sparse, Distributed Memory

Figure 1 shows a model of an SDM. In this model, the main modules of the SDM are an array of addresses, an array of bit counters, an adder and a thresholder.

"Address" is the reference address where the datum is to be stored at or read from. In conventional memories, this reference would activate a single location. In an SDM, though, it will activate all the addresses in a given access radius, which is predefined. Kanerva proposes that the Hamming distance, that is the number of bits in which two binary vectors are different, is used as the measure of distance between the addresses. In consequence of this, all the locations that differ less than a predefined number of bits from the reference address (within the radius distance, as shown in figure 1), are selected for the read or write operation.

In this model, writing is done by incrementing or decrementing the bit counters at the selected addresses. As the figure shows, data are stored in arrays of counters, one counter for every bit of every location. To store 0 at a given position, the corresponding counter is decremented. To store 1, it is incremented. This means that every counter may store either a positive or a negative value. Kanerva proves that, under normal circumstances, the value should fit in the range [-40, 40].

Reading is done by summing the values of all the counters columnwise and thresholding at a predefined value. If the value of the sum is below the threshold, we consider the bit to be zero, otherwise it is one. For a memory where the counters are incremented or decremented one by one, 0 is a good threshold value.

[1]Kanerva proposes that an SDM can be built based on a neural network (NN). Apart the fact that the NN model is biologically inspired, which is not a significant advantage for engineering, we see no other advantage over the symbolic model. Therefore, we'll be using a model based on linked lists, inspired by B. Ratitch's work [10].

Initially, all the bit counters must be set to zero, for the memory stores no data. The bits of the address locations should be set randomly, so that the addresses would be uniformely distributed in the addressing space.

One drawback of SDMs becomes now clear: while in traditional memories we only need one bit per bit, in an SDM every bit requires a counter. Nonetheless, every counter stores more than one bit at a time, making the solution not so expensive as it might seem. Kanerva calculates that such a memory should be able to store about 0.1 bits per bit, although other authors state to have achieved higher ratios [12].

As the theory predicts, there's no guarantee that the data retrieved is exactly the same that was written. It should be, providing that the hard locations are correctly distributed over the binary space and the memory has not reached saturation.

B. SDM Advantages for mobile robotics

The SDM model exhibits some characteristics which make it look attractive to apply in mobile robots based on computer vision, namely:

- According to Rao and Ballard [7], SDMs can be used in pattern (image) recognition, where they have shown to be tolerant to occlusion, illumination changes, scale changes and rotations in 3D. Our results also show that they can store sequences of images with good tolerance to noise and minimum errors [11].
- They're immune to noise up to a high threshold. Using coding schemes such as n-of-m codes, their immunity is even increased [8], at the cost of reducing the addressable space.
- They're robust to failure of individual locations, just like neural networks. This may be important specially for memories implemented in hardware, where occasional or localised errors may occur.
- SDMs degrade gracefully, when some locations fail or the memory approaches its maximum capacity.
- One-shot learning is possible. If the memory is not close to saturation, it will learn in a single pass. This is also a desirable feature for a robot, where long learning periods should be avoided.
- Unlike traditional neural networks, SDMs can be "open" and subject to analysis of individual locations. This is important namely for debugging purposes, or to track the learning process.
- It's possible to change memory's structure without retraining all the memory [10]—an important characteristic to build modular robots.
- Under normal operation we can only retrieve a sequence in the order it was stored, not its reverse sequence. Nonetheless, it's possible to invert the process and, using data as addresses, retrieve the reverse sequence. This characteristic allows a robot to learn one path or task and be able to follow or perform it from the beginning to the end or from the end to the beginning.

The main drawbacks are:

TABLE I

SUMMARY OF THE TOTAL DIMENSIONS OF THE INPUT VECTOR.

Image Resolution	Image bytes	Overhead	Total bytes	Total bits
80x64	5120	13	5133	41064

- Once a datum is written, it cannot be erased, only *forgotten* as the time goes by. Under certain circumstances this may be an undesirable feature, as unnecessary memories cannot be deleted, and they may interfere with more recent and important data.
- Storage capacity may be only about 0.1 bits per bit of traditional computer memory.
- If implemented in software, a lot of computer processing is required to run the memory alone.

III. BUILDING A SPARSE DISTRIBUTED MEMORY

In our implementation, input and output vectors consist in arrays of bytes, meaning that each individual value must fit in the range [0, 255]. Every individual value is, therefore, suitable to store the graylevel value of an image pixel or an 8 bit integer.

The composition of the input vectors is as summarised in table I and equation 1:

$$x_i =< im_{i-1}, im_i, seq\_id, i, timestamp, motion > \quad (1)$$

where im_i is the last image. seq_id is an auto-incremented, 4 bytes integer, unique for each sequence. It is used to identify which sequence the vector belongs to. i is an auto-incremented, 4 bytes integer, unique for every vector in the sequence. It is used to quickly identify every image in the sequence. $timestamp$ is a 4 bytes integer, storing Unix timestamp. It is read from the operating system, but not being used so far for navigation purposes. $motion$ is a single character, identifying the type of movement the robot was performing when the image was grabbed.

Images 80x64 or 160x128 are used. Since every pixel is stored as an 8 bits integer, the smaller image alone needs 80×64 = 5120 bytes = 40960 bits. The overhead information comprises 13 additional bytes, meaning the input vector, for the small size images, contains 41064 bits.

The memory is used to store vectors as explained, but addressing is done using just one image. During the autonomous run, the robot will predict im_i from im_{i-1}. Therefore, the address is im_{i-1}, not the whole vector. The remainder bits could be set at random, as Kanerva suggests, but it was considered preferable to set up the software so that it is able to calculate similarity between just part of two vectors, ignoring the remainder bits. This saves computational power and reduces the probability of false positives being detected. According to the theory, 20% of the bits of im_{i-1} coincident with the robot's current view should suffice for correct retrieval, assuming non-coincident bits are random noise. If we used $im_{i-1} + overhead$, with the overhead bits set randomly, the image's noise level could be, at most, slightly below 20% of the bits. Since we're not using the $overhead$, the tolerance to noise increases a little bit, resulting in less possible errors during normal operation.

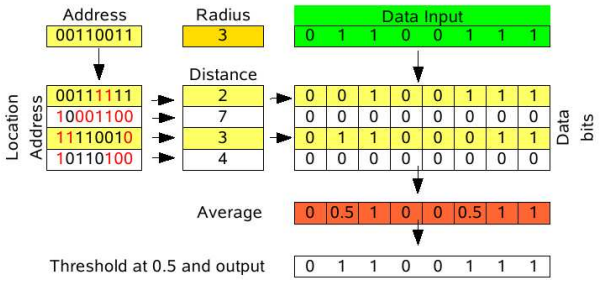

Fig. 2. Bitwise SDM, which works with bits but not bit counters.

Another difference in our implementation, relative to Kanerva's proposal, is that we don't fill the virtual space placing hard locations randomly in the addressing space in the beginning of the operation. Instead, we use Ratitch et al's Randomised Reallocation algorithm [10]: start with an empty memory, and allocate new hard locations when there's a new datum which cannot be stored in enough existing locations. The new locations are allocated *randomly* in the neighbourhood of the new datum address.

A. Bitwise implementation

Kanerva's model has a small handicap: the arrays of counters are hard to implement in practice and require a lot of unnecessary processing, which increases the processing time. Furber et al [8] claim their results show that the memory's performance is not significantly affected if a single bit is used to store one bit, instead of a bit counter, under normal circumstances. For real time operation, this simplification greatly reduces the need for processing power and memory size. In our case, the original model was not implemented, and the system's performance was acceptable using this implementation where the bit counters are replaced by a single bit each, as shown in figure 2.

Writing in this model is simply to replace the old datum with the new datum. Additionally, since we're not using bit counters and our data can only be 0 or 1, when reading, the average value of the hard locations can only be a real number in the interval $[0, 1]$. Therefore, the threshold for bitwise operation is at 0.5.

B. Arithmetic implementation

Although the bitwise implementation works, we also implemented another version of the SDM, inspired by Ratitch et al's work [10]. In this variation of the model, the bits are grouped as integers, as shown in figure 3. Learning is achieved using a reinforcement learning algorithm, and addressing is done using an arithmetic distance, instead of the Hamming distance.

When writing to the memory, the following equation is applied to update every byte value:

$$h_t^k = h_{t-1}^k + \alpha \cdot (x^k - h_{t-1}^k), \quad \alpha \in \mathbb{R} \wedge 0 \leq \alpha \leq 1 \quad (2)$$

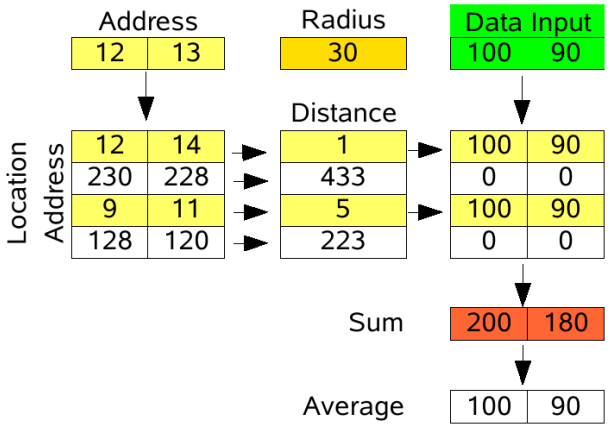

Fig. 3. Arithmetic SDM, which works with byte integers, instead of bit counters.

h_t^k is the k^{th} 8 bits integer of the hard location, at time t. x^k is the corresponding integer in the input vector x and α the learning rate. $\alpha = 0$ means to keep the previous values unchanged. $\alpha = 1$ implies that the previous value of the hard location is overwritten (one-shot learning). In practice, $\alpha = 0.5$ might be a good compromise, and that's the value being used for us. Nonetheless, this means the memory may loose it's one-shot learning ability.

IV. Experimental platform

Lizard is a Surveyor[2] SRV-1, a small robot with tank-style treads and differential drive via two precision DC gearmotors. The speed range is 0.20 m/s to 0.40 m/s. Among other features, it has a built in digital video camera with 80x64 to 640x480 resolution, 4 infrared sensors and a Zigbee 802.15.4 radio communication module. The field of view of the camera was experimentally determined to be approximately 40°. This robot was controlled in real time from a laptop with an Intel 1.8GHz Pentium IV processor and 1Gb RAM.

The overall software architecture is as shown in figure 4. It contains three basic modules:

1) The SDM.
2) The Focus (following Kanerva's terminology), where the navigation algorithms are running.
3) A low level layer, responsible for interfacing the hardware and some low level tasks.

When an image is out of focus or the infrared proximity detectors detect an obstacle close to the robot, Lizard stops an autonomous run and waits until further order is received from the user. In historical terms, Level 1 can be considered as inspired by the first level of competence of the Brooks' subsumption architecture [13]. In biological terms, it can be considered as inspired by the "primitive brain", which controls basic functions such as breathing and other instinctive behaviours. This layer is also responsible for converting the images to 8 bit grayscale and equalise the brightness to improve quality and comparability.

[2]http://www.surveyor.com.

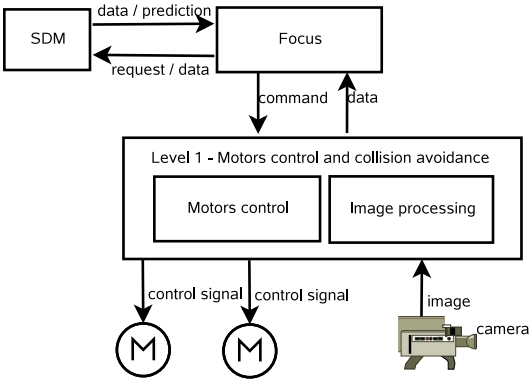

Fig. 4. Architecture of the implemented software.

A. Navigation using a view sequence

Navigation using a view sequence is based on Y. Matsumoto et al's proposal [9]. This approach requires a learning stage, during which the robot must be manually guided. While being guided, the robot memorises a sequence of views automatically. While autonomously running, the robot performs automatic image based localisation and obstacle detection, taking action in real-time.

Localisation is calculated based on the similarity of two views: one stored during the learning stage and another grabbed in real-time. The robot tries to find matching areas between those two images, and calculates the horizontal distance between them in order to infer *how far* it is from the correct path. That distance is then used to correct eventual drifts.

To calculate the drift in each moment, a block matching process is used. A search window selected from the center of the memorised image is matched against an equivalent-size window in the current view, calculating the horizontal displacement that results in the smallest error e. Considering two images I_1 and I_2, the matching error is defined as:

$$e(u) = \sum_{x=s}^{w-s} \sum_{y=0}^{h} |I_1(x,y) - I_2(x+u,y)| \qquad (3)$$

w and h are the width and height of the image, in pixels. u is the offset, or horizontal displacement of the search window s, which was set to 34 pixels. $I_i(x,y)$ is the graylevel intensity of pixel (x,y) of image i. And the error that is of interest for us is the minimum $e(u)$, defined as follows:

$$e = min(e(u)), -s \leq u < s \qquad (4)$$

This technique is not as robust as stereo matching techniques, but it is more interesting for our approach because of its computational simplicity. Moreover, considering the camera cannot move, vertical displacements are not likely to occur.

B. Dealing with noise

To calibrate the system, it is necessary to previously estimate noise levels. We consider noise the distance between

two consecutive pictures taken without any environmental change (i.e., the lighting and scenario are the same).

To make navigation a little more robust, a dynamic algorithm was implemented, to automatically adjust to the noise level before learning. Once the system is turned on, Lizard captures 3 consecutive images and computes the difference between the first and the second, and between the second and the third. The average of the two values is taken as a good approximation to the actual noise level.

As proposed in [11], to make sure the images are retrieved from the memory when following the sequence, the access radius was set as a function of the noise level. It was set to the average value of the noise increased 40% for bitwise operation, and 70% for arithmetic operation.

Under the conditions specified, Lizard showed a good ability to learn and follow simple paths, even when "kidnapped" in the middle of a sequence.

V. Results

Several parameters were tested in practice, namely to get experimental results that could show the differece in performance of the robot with and without equalisation, as well as the impact of using the arithmetic or bitwise implementation.

All the data grabbed from the robot was recorded in the computer hard disk, so that the same path could be learnt again by the system, without the need of moving the robot throughout the same exact path. This procedure made it possible to test autonomous runs both with and without preprocessing of the images, and different working parameters.

A. Processing time

To measure the processing time, the memory was loaded with 100 images in 3 hard locations each (a total of 300 addresses). The search for the closest matching image takes about 190 ms in bitwise mode and less than 80 ms in arithmetic mode. It should be mentioned that the bitwise operations take longer because they are simulated in software. An hardware implementation should provide much faster results for the bitwise operation. Nonetheless, these measures confirm that real time operation of such a memory requires the use of a fast processing unit.

B. Illumination changes

The effect of image processing was also tested, in order to analyse the advantages of using algorithms to improve the quality of the images grabbed from the camera. Although simple, this processing is also time consuming, which is a cost for real time operation.

Table II compares the performance of the memory with and without application of image processing algorithms. The results were obtained using a sequence of 80 images. Those images were captured under a strong light and stored in the computer's hard disk without processing. The robot was then placed in the beginning of the path, where it was expected to see image 1, and the various algorithms were applied.

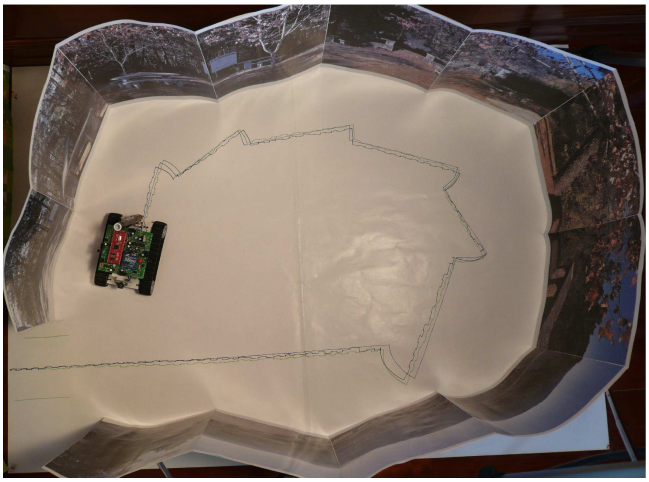

Fig. 5. Paths learnt and predicted.

As table II shows, image processing succeeds in increasing the matching error between the best matching image and the the other images, specially if the lighting conditions are changed. Additionally, changing the lighting conditions, a gross image recognition error occurs if the images are not processed, for image 53 is wrongly recognised. With contrast stretching, image 2 is detected, which is close to the first, and equalisation solves the problem.

C. Full sequence tests

Figure 5 shows an overall view of the paths followed during a test run. The lines were drawn using a pen attached to the rear of the robot, as shown in the image. Therefore, they mark the path described by its rear, not its centre. The blue line shows the path that was taught during the learning stage. The whole path is described by 123 80x64 images. During the autonomous run, the robot is expected to retrieve ("see") each image in the same sequence that it was stored during the learning stage. If at any point in the navigation it goes back in the sequence—say, from image im_i to im_{i-1}—, that is considered a navigation error.

The blue line shows the path followed during an autonomous run, in which the images were equalised and compared using the arithmetic distance. There were 11 errors during this run. The black line shows another path followed during an autonomous run, using the bitwise mode. There were 20 errors during this run. One reason for the poorer performance of the bitwise mode may be the fact that the binary code that represents the brightness intensity is not continuous. The distance between 127 and 128, for example, is 1 in the arithmetic mode, but 7 bits in binary code.

VI. Conclusions and Future Work

The scrimpy results of AI over the last decades are leading researchers to look for different approaches. One field that still requires further investigation is that of artificial memory systems, since there's evidence that the human brain may be working as a huge memory system, relying more on associations and analogy than on mathematical processing.

TABLE II

COMPARISON OF MATCHING ERRORS FOR DIFFERENT IMAGE PROCESSING ALGORITHMS. SL STANDS FOR STRONG LIGHT, DL FOR DIM LIGHT, CS FOR CONTRAST STRETCHING, EQ FOR EQUALISATION, AR FOR ARITHMETIC MODE AND BW FOR BITWISE MODE.

Lighting	Processing	Mode	Best match	Second match	Inc.	Average	Inc.	Remarks
SL	None	AR	240342	266458	10.87%	440792.10	83.40%	
SL	CS	AR	271550	300208	10.55%	539367.31	98.63%	
SL	Eq	AR	401707	419718	4.48%	709934.27	76.73%	
SL	CS + Eq	AR	400324	418955	4.65%	709363.06	77.20%	
SL	None	BW	36912	37469	1.51%	39758.20	7.71%	
SL	CS	BW	35890	36178	0.80%	39168.33	9.13%	
SL	Eq	BW	35369	35732	1.03%	39352.21	11.26%	
SL	CS + Eq	BW	35311	35723	1.17%	39367.54	11.49%	
DL	None	AR	662573	665912	0.50%	842004.94	27.08%	Detects image 10
DL	CS	AR	311795	326919	4.85%	577167.11	85.11%	
DL	Eq	AR	454176	461733	1.66%	712281.14	56.83%	
DL	CS + Eq	AR	452801	460476	1.70%	711719.65	57.18%	
DL	None	BW	39579	40224	1.63%	41448.90	4.72%	Detects image 53
DL	CS	BW	36436	36551	0.32%	39529.25	8.49%	Detects image 2
DL	Eq	BW	37358	37580	0.59%	39559.84	5.89%	
DL	CS + Eq	BW	37269	37276	0.02%	39584.21	6.21%	

It seems to *retrieve* solutions, more than computing new ones. Considering those facts, we implemented a Sparse Distributed Memory and equipped a small robot to navigate making use of it. The SDM is a kind of associative memory which exhibits typically human characteristics, such as storing and retrieval of sequences, natural learning and forgetting, as well as graceful degradation.

Lizard is able to learn and follow paths based on view sequences, captured by its onboard camera. During learning it stores images of unknown scenes. During navigation it compares images from its current view to the closest matching image, and decides its next action based on the action associated with the retrieved image. Small horizontal drifts are corrected based on the horizontal distance between the current view image and the retrieved view image. It is *naturally* immune to being kidnapped and transposed to another known location, for its navigation is solely based on vision.

The main limitation of this approach is that, being vision-based, the system is very sensitive to image noise and illumination (although other sensorial information may be used). Another drawback is that the SDM itself requires computational processing, and provides storage of about 0.1 bits per bit. It also cannot be selectively erased—data must be *forgotten* over time.

Future work includes the design and implementation of more sophisticated navigation algorithms, making use of the overhead information stored with each vector. The robot is currently only using the visual information stored in the image, but it may also use time and odometry information, thus being "aware" of time and detecting when it was kidnapped, or erroneously skipped from one sequence to another. Another subject of study may be the use of a different binary system to code the brightness intensities, with the goal of improving the results of bitwise operations.

REFERENCES

[1] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, New York, 2004.

[2] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, 1988.

[3] Joy Bose. A scalable sparse distributed neural memory model. Master's thesis, University of Manchester, Faculty of Science and Engineering, Manchester, UK, 2003.

[4] Dileep George and Jeff Hawkins. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks*, 2005.

[5] David Rogers. Predicting weather using a genetic memory: A combination of Kanerva's sparse distributed memory with Holland's genetic algorithms. In *NIPS*, 1989.

[6] Ashraf Anwar, Dipankar Dasgupta, and Stan Franklin. Using genetic algorithms for sparse distributed memory initialization. In *International Conference Genetic and Evolutionary Computation (GECCO)*, July 1999.

[7] Rajesh P. N. Rao and Dana H. Ballard. Object indexing using an iconic sparse distributed memory. Technical Report 559, The University of Rochester, Computer Science Department, Rochester, New York, July 1995.

[8] Stephen B. Furber, John Bainbridge, J. Mike Cumpstey, and Steve Temple. Sparse distributed memory using *n*-of-*m* codes. *Neural Networks*, 17(10):1437–1451, 2004.

[9] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. View-based approach to robot navigation. In *Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, 2000.

[10] Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *ECML*, 2004.

[11] Mateus Mendes, A. Paulo Coimbra, and Manuel Crisóstomo. AI and memory: Studies towards equipping a robot with a sparse distributed memory. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, Sanya, China, December 2007.

[12] James D. Keeler. Comparison between Kanerva's SDM and Hopfield-type neural networks. *Cognitive Science*, 12(3):299–329, 1988.

[13] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986.