Robotica (2012) volume 30, pp. 571–581. © Cambridge University Press 2011
doi:10.1017/S0263574711000828

Robot navigation based on view sequences stored in a sparse distributed memory

Mateus Mendes†,*, A. Paulo Coimbra‡ and Manuel M. Crisóstomo‡

†*ESTGOH, Polytechnic Institute of Coimbra, R. General Santos Costa, 3400-124 Oliveira do Hospital, Portugal*
‡*Institute of Systems and Robotics, Pólo II, Pinhal de Marrocos, 3030-290 Coimbra, Portugal*

(Accepted June 15, 2011. First published online: July 26, 2011)

SUMMARY

Robot navigation is a large area of research, where many different approaches have already been tried, including navigation based on visual memories. The Sparse Distributed Memory (SDM) is a kind of associative memory based on the properties of high-dimensional binary spaces. It exhibits characteristics, such as tolerance to noise and incomplete data, ability to work with sequences and the possibility of one-shot learning. Those characteristics make it appealing to use for robot navigation. The approach followed here was to navigate a robot using sequences of visual memories stored into a SDM. The robot makes intelligent decisions, such as selecting only relevant images to store, adjusting memory parameters to the level of noise and inferring new paths from the learnt trajectories. The method of encoding the information may influence the tolerance of the SDM to noise and saturation. This paper reports novel results of the limits of the model under different typical navigation problems. The SDM showed to be very robust to illumination and scenario changes, occlusion and saturation. An algorithm to build a topological map of the environment based on the visual memories is also described.

KEYWORDS: Sparse distributed memory; SDM, Robot navigation, Data encoding, Vision-based navigation.

1. Introduction

Humans are highly motivated by visual memories. Visual information is stored in sequences every time they see something new and recalled whenever they think of a particular item or situation. Humans do it *naturally*, quickly and without effort. But robots, and computational models in general, are still far from achieving the standards of human beings, despite the significant progress of computer vision.

Present robots usually rely on heavy processing and limited amounts of memory. However, evidence indicates that the human brain functions in the opposite manner: limited processing and huge amounts of memory, to store sequences of events that will lead future actions and analysis.[1,2]

The brain takes about 1/10th of a second to perform tasks, such as language understanding or visual recognition. Considering that neurons need about 1/1000th of a second to send a signal, this means that, on average, those tasks cannot take more than 100 serial steps.[3] One possible explanation for this remarkable behaviour is that the brain performs many tasks in parallel. However, there are tasks, which cannot be performed in parallel. Hawkins[1] proposes a thought experiment that can be summarised as follows: if it takes 100 steps to cross a river, it does not matter how many people there are to cross it—it always takes 100 steps to get to the other side. Therefore, most of the time, the brain has to retrieve solutions by analogy with past situations. It operates more as a sophisticated memory bank than a high-speed processing unit.

Based on this idea, Kanerva[2] proposed the Sparse Distributed Memory (SDM) model, a kind of associative memory sharing the properties of high-dimensional binary spaces. Kanerva presents thorough demonstrations of how the binary space 2^n, for $100 < n < 10^5$, exhibits properties that are similar to human intuitive notions of relationships between concepts. These properties are inherited by the SDM.

The concept of the SDM is very attractive for robot navigation. It will confer on the robot the ability to learn and follow paths, the same way humans do. Some researchers have already explored this idea. Rao and Fuentes[4] simulated the use of a SDM to store position information from optical sensors, associated with motor controls, during a learning stage. Using that information, the robot was later able to follow the same paths. The authors presented only simulation results. Watanabe *et al.*[5] used a SDM for the task of scene recognition in a factory environment, where the robots had to move autonomously from one place to another. The SDM, however, was just used as an auxiliary method of scene recognition.

The present work uses a SDM to implement vision-based navigation. The memory is used to store sequences of images and navigation data. Using that information, the robot is able to follow learnt paths. Since the SDM can also work as a pattern matching tool, an algorithm is also proposed to build a partial topological representation of the environment, which can be used for path planning. The system was tested with different methods of encoding the information, and in some typically challenging situations. The results were obtained using a real robot, in different reconstitutions of urban and countryside scenarios. Among other features, the SDM-based navigation showed to be highly robust to illumination and scenario changes and to memory overflows. The limits of the memory under different typical robot navigation problems were tested.

* Corresponding author. E-mail: mmendes@estgoh.ipc.pt

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

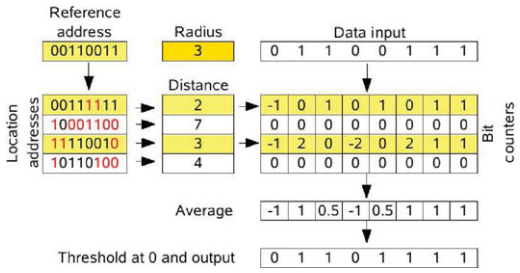


Fig. 1. (Colour online) Diagram of a SDM, according to the original model, showing an array of bit counters to store data and an array of addresses.

The idea of navigating robots based on visual memories is not new itself. The present approach is similar to Matsumoto *et al.*: the robot grabs images during a supervised learning stage and uses them later for autonomous navigation. During the autonomous run, small drifts are detected using a block matching technique, to compare the robot's view with the stored view, and corrected.[6] Jones *et al.* also attempted a similar approach, using low-resolution wide angle images and correcting small drifts adjusting the trajectory of the robot in order to maximise the Zero Mean Normalised Cross Correlation between the robot's current view and the images stored.[7] Other authors have tried to improve the performance of the method. Winters and Santos-Victor compress the images using Principal Component Analysis.[8] Franz *et al.* aim to select only relevant images.[9] Jensfelt *et al.* extract image features to improve the speed and performance of the process.[10] Differently from other models, the SDM naturally provides a framework for manipulating sequences of high-dimensional Boolean vectors. That is the reason why it was selected for the present implementation, in which its properties are tested under extreme conditions, in order to assess the robustness of the model and its adequacy to the task of navigating robots.

Section 2 briefly describes the SDM theory. Section 3 describes the experimental platform. Section 4 describes the problems found with the method of encoding the information, and the variations of the SDM models implemented. Section 5 presents a method of building a topological representation and path planning using the SDM as a pattern matching tool. Section 6 describes the experiments performed and the results obtained. In Section 8, some conclusions are drawn.

2. Sparse Distributed Memories

The underlying idea behind the SDM is the mapping of a huge binary memory onto a smaller set of physical locations, so-called *Hard Locations* (HLs). As a general guideline, those HLs should be uniformly distributed throughout the virtual space, to *mimic* the existence of the larger virtual space as accurately as possible. Every datum is stored by distribution to a set of HLs, and retrieved by *averaging* those locations.

Fig. 1 shows a model of a SDM. 'Address' is the reference address where the datum is to be stored or read from. It will activate all the HLs in a given access radius, which is predefined. Kanerva proposes that the Hamming distance (HD), that is the number of bits in which two binary vectors are different, be used as the measure of distance between the addresses. All the locations that differ less than a predefined number of bits from the input address are selected for the read or write operation.

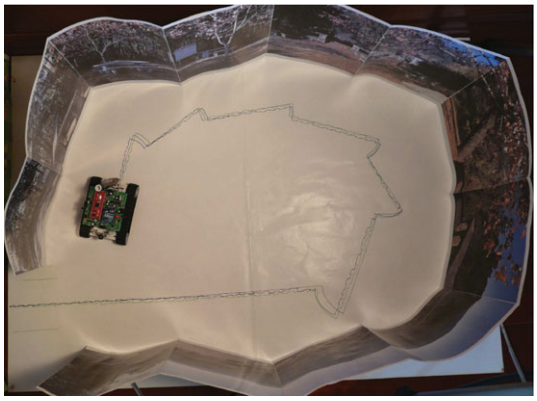


Fig. 2. (Colour online) Surveyor robot in one of the testbeds, surrounded by a realistic countryside scenario representing mountain views.

Data are stored in arrays of counters, one counter for every bit of every location. Writing is done by incrementing or decrementing the bit counters at the selected addresses. To store 0 at a given position, the corresponding counter is decremented. To store 1, it is incremented. Reading is done by averaging the values of all the counters columnwise and thresholding at a predefined value. If the value of the sum is below the threshold, the bit is zero; otherwise, it is one.

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 uniformly distributed throughout the addressing space.

There is no guarantee that the data retrieved is exactly the same that was written. It should be, providing that the HLs are correctly distributed over the binary space and the memory has not reached saturation.

3. Experimental Platform

The SDM was tested driving a small robot. Due to its small size and limited processing power, the robot was controlled in real time from a laptop and moved inside a special arena where various realistic scenarios were reconstructed.

3.1. Hardware and software

The robot used was a Surveyor[1] SRV-1, a small robot as can be seen in Fig. 2. Among other features, it has a built-in video camera and a radio communication module. Control is exerted in real time from a laptop with a 1.8 GHz processor and 1 Gb RAM. The overall software architecture is as shown in Fig. 3. It contains three basic modules: the SDM, where the information is stored; the Focus (following Kanerva's terminology), where navigation algorithms are run; and an operational layer, responsible for tasks, such as motor control, collision avoidance and image equalisation. Navigation is based on vision, following Matsumoto *et al.*'s proposal.[6] It has two modes: supervised learning and autonomous run. In the learning mode, the

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

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

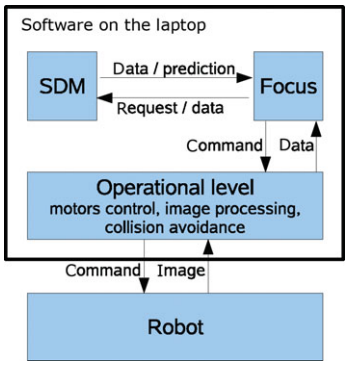


Fig. 3. (Colour online) Overview of the architecture of the system.

system is manually guided and stores images and navigation information, one sequence per path. But only images that present a distance to their predecessors superior to the activation radius are stored, in order to save memory and avoid mix up of consecutive images. In the autonomous run mode, the robot grabs images in real time and searches the SDM for the closest matching image. If it recognises the view, it performs the same action that is associated with the image retrieved from the SDM, thus following the same path it has learnt. When it stops recognising the images or detects an obstacle, it stops until further orders.[11]

Fig. 2 shows the robot in one of the used testbeds. For practical constraints, the tests were performed in a controlled environment. The scenario surrounding the robot is a typical mountain environment, built with high-quality printed pictures of a Portuguese mountain. The picture also shows the robot with a pen attached to its rear so that it leaves a printed trail of the path. The pen marks the motion of the rear of the robot. That is the reason for the arcs that appear when the robot changes direction.

3.2. *SDM input*

The composition of the input vectors is

$$x_i =< im_i, seq_id, i, timestamp, motion > \qquad (1)$$

In those vectors, im_i is the image i, in Portable Grey Map (PGM) format and 80×64 resolution. In PGM images, every pixel is represented by an 8-bit integer. The value 0 is black, while 255 is white. seq_id is an auto-incremented, 4-byte integer, unique for each sequence. It is used to identify which sequence the vector belongs to. The i is an auto-incremented, 4-byte integer, unique for every vector in the sequence, used to quickly identify every image in the sequence. The $timestamp$ is a 4-byte integer, storing Unix timestamp. It is not being used so far for navigation purposes. Finally, the single character $motion$ identifies the type of movement the robot performed after the image was grabbed.

The image alone uses 5120 bytes. The overhead information comprises 13 additional bytes. Hence, the input vector contains 5133 bytes.

4. SDM Implementations

In the present work, auto-associative versions of the SDM were implemented, implying that the same vectors are simultaneously used as addresses and data. Datum ζ is stored at location ζ, where ζ is address and datum at the same time. Hence, the array of addresses becomes unnecessary, and memory size is cut to about one half, without loss of performance or generality. The software was also adjusted so that addressing is done using just the part of the input vector corresponding to im_i, disregarding the overhead information.

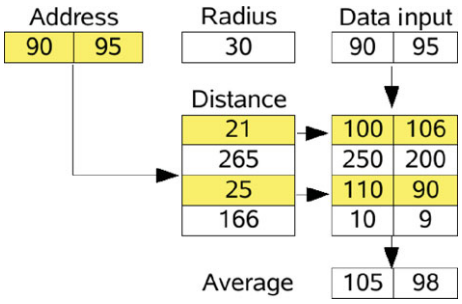


Fig. 4. (Colour online) Auto-associative arithmetic SDM. The data are simultaneously addresses, and the distances are arithmetic.

Another difference, relative to Kanerva's proposal, is that the virtual space is not filled placing HLs randomly at the addressing space in the beginning of the operation. Instead, Ratitch and Precup's Randomised Reallocation algorithm is used:[12] start with an empty memory, and allocate new HLs when there is a new datum that cannot be stored in enough existing locations.

Different models of the SDM were also implemented: three that use the HD (bitwise implementations), and another one that uses an Euclidean distance (arithmetic implementation), as described below.

4.1. *Bitwise implementation*

Kanerva's model has a small handicap: the arrays of counters are hard to implement in practice and require a lot of processing, thus increasing the processing time in the models simulated using traditional computers. Furber *et al.*[13] claim their results show that the memory's performance is not significantly affected if a single bit is used to store 1-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. Therefore, in the present implementation, the bit counters were replaced by a single bit each.

Writing in this model is simply to replace the old datum with the new datum. Additionally, when reading, the average value of the HLs can only be a real number in the interval [0, 1]. Hence, the threshold for bitwise operation is at 0.5.

4.2. *Arithmetic implementation*

The arithmetic model is suggested by Ratitch and Precup.[12] In this model, the bits are grouped as byte integers, as shown in Fig. 4. Addressing is done using an Euclidean distance, instead of the HD, and writing is done by applying to each byte the following equation:

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

In the equation, h_t^k is the kth 8-bit integer of the hard location, at time t. The x^k is the corresponding integer in the input

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

vector x and α is the learning rate. Learning rate $\alpha = 0$ means to keep the previous values unchanged. Learning rate $\alpha = 1$ implies that the previous value of the hard location is overwritten (one-shot learning), and that is the value used.

4.3. Determination of the access radius

An important parameter of the SDM is the activation radius—it determines that memory locations are selected to read or write. In the presented implementations, it is calculated dynamically at the beginning of each sequence, as a function of the noise levels in the images. Noise is defined as the distance between two consecutive pictures taken without any environmental change (i.e. the lighting and scenario are the same). Once the system is turned on, the robot captures three consecutive images and computes the distance between the first and the second, and between the second and the third. The average of the two distances is taken as a good approximation to the actual noise level. The access radius is set to the noise level increased by 40% for bitwise operation, and 70% for arithmetic operation.[14]

4.4. Encoding problem

In Natural Binary Code (NBC), the value of each bit depends on its position. 0001_2 (1_{10}) is different from 1000_2 (8_{10}) although the number of ones and zeros in each number is exactly the same. The Arithmetic Distance (AD) between those numbers is $8 - 1 = 7$, and the HD is 2. Considering the numbers 0111_2 (7_{10}) and 1000_2 (8_{10}), the AD is 1 and the HD is 4.

These examples show that the HD is not proportional to the AD. There are some *undesirable transitions*, i.e. situations where the HD decreases and the AD increases, as shown in Table I. Those transitions happen in possibly all the codes where the position of the bits influences their value, including Gray code and other popular codes. However, sensorial data, such as PGM images, are usually encoded using the NBC, to take advantage of the positional values of the bits. Hence, it is hardly random. The performance of the SDM, therefore, might be affected because of these different criteria being used to encode the information and to process it.[15]

One possible approach to solve the problem is to sort the bytes, producing a code whose HD matrix does not exhibit undesirable transitions. This sorting can be accomplished by trying different permutations of the numbers and computing the matrix of HDs, thus producing an Optimised Code (OC).[15] However, the only way to avoid undesirable transitions at all is to reduce the number of different grey levels to the number of bits +1 and use a kind of sum code. Using, for example 4 bits, it is only possible to represent five different grey levels: {0000, 0001, 0011, 0111, 1111}. Using 8 bits, nine grey levels can be used, and so on. This is the only way to work with a HD that is proportional to the Euclidean distance. The disadvantage, however, is that either the quality of the image is much poorer, or the dimension of the stored vectors has to be extended to accommodate additional bits.

Table I. Matrix of Hamming distances for 3-bit natural binary code.

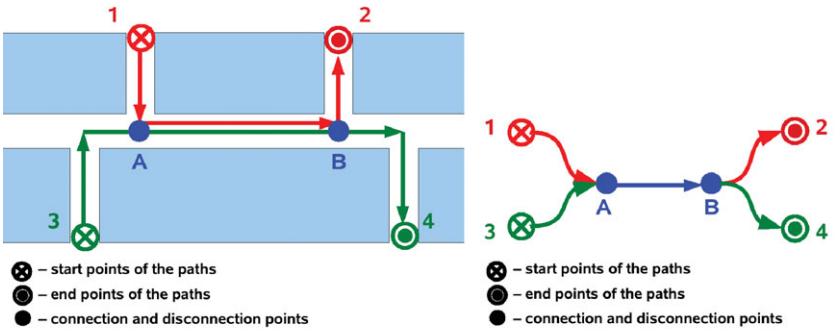
	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001		0	2	1	2	1	3	2
010			0	1	2	3	1	2
011				0	3	2	2	1
100					0	1	1	2
101						0	2	1
110							0	1
111								0

5. Mapping and Planning

The 'teach and follow' approach *per se* is very simple and powerful. But for robust navigation and route planning, it is necessary to extend the basic algorithm to perform tasks, such as detection of connection points between the paths learnt and disambiguation when there are similar images or divergent paths.

5.1. Building a topological representation

In situations, such as the one depicted in Fig. 5, the robot does not need to learn the segment AB twice. The figure shows two different paths, connecting four different laboratories. The green path (1-2) goes from Lab 1 to Lab 2, the red path (3-4) goes from Lab 3 to Lab 4. The segment AB is a portion of the corridor that is common to both paths. If the robot learns segment AB for the green path, for example then it does not



(a) Paths connecting four different laboratories. (b) Topological representation of the paths.

Fig. 5. (Colour online) Example of paths that have a common segment. The robot does not need to learn AB twice when mapping the building. (a) Paths connecting four different laboratories; (b) Topological representation of the paths.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

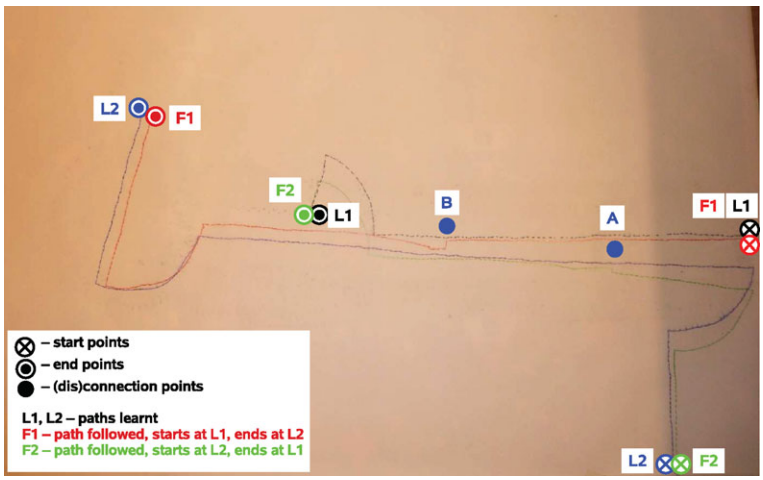


Fig. 6. (Colour online) Paths learnt (L_i) and followed (F_i). The robot learnt segment AB only once and at point B chose the path according to the destination.

need to learn it again for the red path. When learning the red path, it only needs to learn it until point A. Then, it can store an association between the green and red paths at point A and skip all the images until point B. At point B, it should again record a connection between the red and green paths. This way, it builds a map of the connection points between the known paths. That is a kind of topological representation of the environment, sufficient for path planning and robot navigation.

The main problem with this approach is to detect the connection points. The points where the paths come together (point A in Fig. 5) can be detected after a *reasonable* number of images of the green path have been retrieved, when the robot is learning the red path. When that happens, the robot stores the connection in its working memory and stops learning the red path. From that point onwards, it keeps monitoring if it is following the same green path. After a *reasonable* number of predictions have failed, it adds another connection point to the graph and resumes learning the new path. In the tests with the SDM, a number of three to five consecutive images within the access radius usually sufficed to establish a connection point, and three to five images out of the access radius were a good indicator that the paths were diverging again.

5.2. Tests in an empty arena simulating a country environment

The mapping and planning algorithm was tested in the arena surrounded by a realistic countryside scenario, as described in Section 3, and later in an arena filled with objects simulating a urban environment. All the results shown, unless stated otherwise, were obtained when the memory was loaded with three copies of each image, in the arithmetic mode.

The first test performed consisted in analysing the behaviour of the navigation algorithm in the empty arena. The surrounding wall was printed with a composition of images of mountain views, as shown in Fig. 2.

Fig. 6 shows an example of the results obtained. In the example, the robot was first taught paths L1 and L2. Then, the memory was loaded with both sequences, establishing connection points A and B. The minimum *overlapping* images required for establishing a connection point was set to three consecutive images. The minimum number of different images necessary for splitting the paths at point B was also set to three consecutive images out of the access radius. As the picture shows, the robot was able to start at the beginning of sequence L1 and finish at the end of sequence L2, and vice versa. Regardless of its starting point, at point A it always defaulted to the only known path L1. This explains the small arc that appears at point A in path F2. This arc represents an adjustment of the heading when the robot defaulted to path L1.

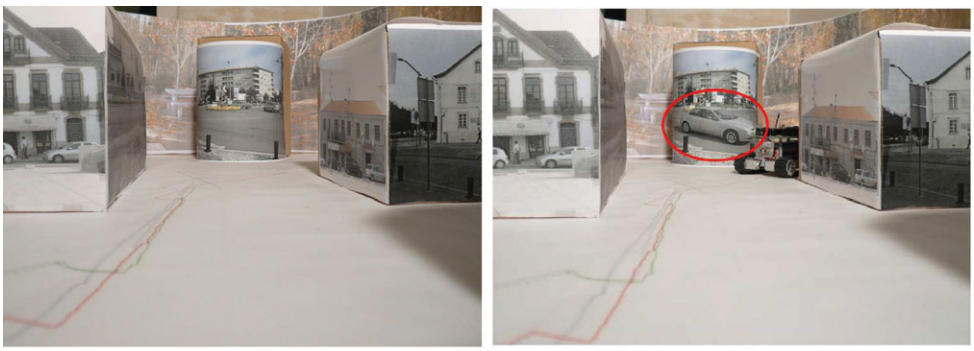
The direction the robot takes at point B depends on its goal. If the goal is to follow path L1, it continues along that path. If the goal is to follow path L2, it will disambiguate the predictions to retrieve only images from path L2. This behaviour explains the changes in direction that appear in the red line (F1) at point B. The arcs were drawn when the robot started at path L1, but with the goal of reaching the end of path L2.

5.3. Tests in a simulated urban environment

In a second experiment, the scenario was filled with images mimicking a typical city environment. Urban environments change very often. Ideally, the robot should learn one path in a urban environment but still be able to follow it in case there are small changes, up to an *acceptable* level. For example, Fig. 7 shows two pictures of a traffic turn, taken only a few seconds apart. Although the remaining scenario holds, one picture captures only the back of a car in background. The other picture captures a side view of another car in foreground.

Due to the small dimensions of the robot, it was not tested in a real city environment, but in a reconstitution of it. Fig. 8 shows the results. Fig. 8(a) shows the first scenario, where the robot was taught. In this scenario, the robot, during segment AB, is guided essentially by the image of the traffic turn without the car. In a second part of this experiment, the picture of the traffic turn was replaced by the one with the car in foreground, and the robot was made to follow the same paths. Again, it had to start at path L1 and finish at path L2, and vice versa. As Fig. 8(b) shows, it was able to successfully

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press



(a) (b)

Fig. 7. (Colour online) Typical city view, where the traffic turn is temporarily occluded by a passing car in the autonomous run (notice the red circle). The robot overcomes interferences like this.

complete the tasks. The results shown in this section were obtained using the arithmetic implementation of the SDM with three copies of each image stored, but they are equivalent to the results obtained using other operation modes under the same circumstances.

6. Tests Under Different Typical Problems

In order to assess the navigational performance of the robot with the SDM at its helm, it was tested in several different situations and facing typical navigation problems. The results are presented here for different implementations, corresponding to four different operation modes: (1) Arithmetic (Arit); (2) Bitwise implementation using the NBC; (3) OC and (4) 8-bit sum-code (S09). As for the minimum number of copies of each datum to be stored in the memory, tests were performed with one and three copies, in order to have an insight into how the distribution of the data might affect the performance of the robot.

6.1. Different illumination conditions

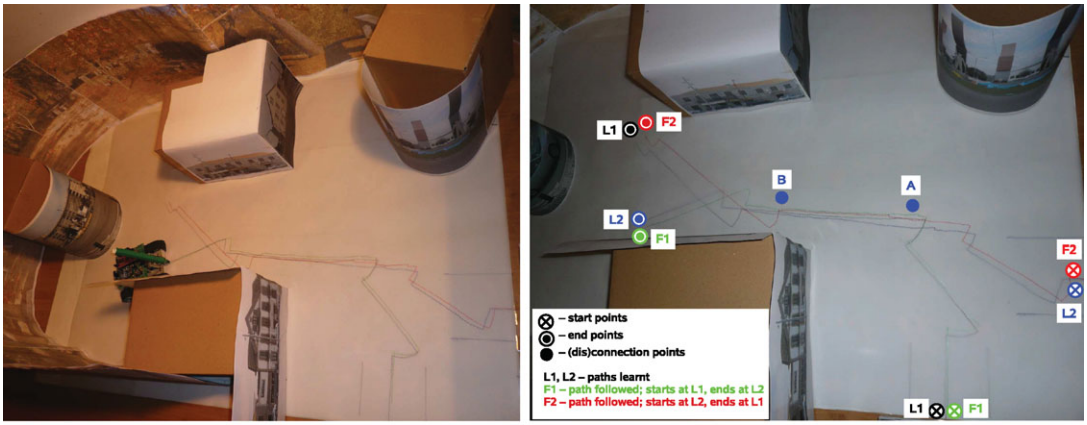
One of the most challenging problems of computer vision is that of illumination. The same scene may look very different if under dim or strong light, or if the relative position of the light sources is changed.

In order to assess the performance of the system under different illumination conditions, the robot was taught one path under a strong light source and tried to follow it in two different situations:

(1) With only ambient light, where an 11 W compact fluorescent bulb was the only light source to illuminate a $4 \times 7\,m^2$ room. This bulb was suspended from the ceiling, while the experimental setup was located at a corner.
(2) Including ambient light and the fill light of a 300 W projector, using a halogen bulb. This projector was directed towards the scenario at the height of 1.2 m, from one side of the arena.

Fig. 9 shows the testbed under these two different illumination conditions. The light sources are placed so that they cause shadows in opposite directions. That is the most challenging situation to the robot.

Table II summarises the results obtained in the experiments, for a sequence of 55 images. The first column is the minimum number of copies that is guaranteed to be stored in the memory for each image. The second column is the qualitative intensity of the light. The third is the operation mode. The fourth column is the average number of Momentaneous Localisation Errors (MLEs). The results shown are the average of five runs. The rows for NBC and



(a) The traffic turn on the left is not occluded. (b) The traffic turn on the left is partially occluded by a passing car

Fig. 8. (Colour online) Paths learnt (L1 and L2) and followed. The small scenario changes did not affect the performance of the robot. (a) The traffic turn on the left is not occluded; (b) The traffic turn on the left is partially occluded by a passing car.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

Table II. Results obtained under different illumination conditions. The arithmetic and sum code modes are more robust to illumination changes.

Copies stored	Light intensity	Operation mode	MLE (errors)
3	Strong	Arit	16.4
		NBC	20.6
		OC	23.0
		S09	14.4
	Dim	Arit	62.6
		NBC	∞
		OC	∞
		S09	61.2
1	Strong	Arit	17.2
		NBC	24.8
		OC	23.6
		S09	14.6
	Dim	Arit	63.6
		NBC	∞
		OC	∞
		S09	64.0

OC operation mode under dim light were suppressed because under this operation mode and dim light the robot is not able to complete the task.

A MLE is counted when the system retrieves image im_{i-j}, for $i, j \in N$, a step after retrieving im_i. When this happens, either im_{i-j} or im_i were incorrectly retrieved because the robot must never get back in the sequence. Although a MLE is not a serious threat to the robot's ability to complete the task, it is a good estimator of the performance of the system. It should be noticed that during the autonomous run the robot moves in steps 1/16th the length of those of the learning mode. This means that to complete a sequence described by 55 images, in the autonomous run the robot will possibly perform more than 1000 steps: at least 55×16 steps are required, but at each step it will check for any horizontal drifts and correct them if necessary. Therefore, even 64 MLEs actually represent errors in about 6% of the total number of steps, and only pose a hardly noticeable delay in the completion of the task.

In previous works,[15] the authors also presented the distance (error in similarity) between the image currently viewed by the robot (input vector) and the closest matching image in the memory (data predicted). This distance was compared to the distance to the second best prediction and the average distance to all the images stored in the memory. This analysis allows a better insight into *how successful* the memory was in separating the possibly right prediction from the pool of information stored. However, the number of MLEs, as could be expected, shows a strong negative correlation with these measures: the more successful the system is in *separating* the right image from the pool of information, the less MLEs occur. Therefore, this paper only presents the number of MLEs as a measure of performance.

Analysing Table II, it can be reasoned out that NBC and OC are much worse than the arithmetic and the sum-code modes. The arithmetic and the sum-code modes perform almost the same, with a small advantage to the sum code. As for distribution of the data, the results using three copies of the images are, in general, better than the results obtained using a single copy.

6.2. *Scenario changes*

Partial scenario changes occur in common environments all the time. For instance, a door can be either open or closed, people can pass by or come into the scene, objects can be moved from one position to another, rotated or shifted, etc. All these changes can pose serious problems to an autonomous robot, eventually making it impossible for it to recognise a scene based on visual information only, even if it has already been there.

Scenario changes are just a particular kind of noise for the SDM. Up to a certain level they should not pose a serious threat to the performance of the system. How much of this noise is tolerable depends on other environmental conditions and on the parameters of the SDM. To simulate a scenario change in software, part of the image was erased, setting some pixels to zero (black) before sending it to the SDM, during the autonomous run. The effect is the same as if there was a real object in the scene, but it is possible to have total control over the characteristics of the interference. Besides, setting some pixels to zero is a kind of 'worst-case' scenario:



(a) Ambient light only. (b) Including fill light of a 300 W projector.

Fig. 9. (Colour online) Scenario under dim and strong lights, with shadows pointing in different directions. In the arithmetic and sum-code modes, the robot successfully navigates in both cases. (a) Ambient light only; (b) Including fill light of a 300 W projector.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

Table III. Impact of different scenario changes with one and three copies stored. The robot is able to navigate with interferences of up to 12 lines plus noise.

Erased lines	Operation mode	MLE (three copies)	MLE (one copy)
2	Arit	14.4	15.8
	NBC	21.6	25.6
	OC	22.8	22.8
	S09	14.4	14.6
4	Arit	18.6	16.6
	NBC	25.2	24.2
	OC	22.8	23.6
	S09	16.2	16.2
6	Arit	18.4	16.0
	NBC	26.4	27.0
	OC	23.6	23.8
	S09	17.0	15.0
8	Arit	17.6	16.6
	NBC	25.0	24.0
	OC	24.0	26.0
	S09	16.0	14.0
10	Arit	20.4	16.6
	NBC	25.6	26.8
	OC	29.0	26.8
	S09	19.8	14.2
12	Arit	∞	∞
	NBC	29.8	29.2
	OC	25.0	30.4
	S09	∞	∞

under normal conditions, the interference is expected to be better distributed in the [0, 255] codomain.

Table III shows the results obtained. The first column shows the number of lines erased in the image. Starting with two vertical lines and increasing in steps of two, the performance of the system was measured in each case. The number of MLEs steadily increases, as the right images get harder to retrieve from the pool of images in the SDM. Distribution of the data seems to have little influence on the results.

Although with a large number of MLEs, the NBC and OC modes support an interference of 12 lines, or 15% of the image. The arithmetic and sum-code modes cause less MLEs, but fail more abruptly and support an interference of just 10 lines, or 12.5% of the image. It should be stressed, however, that these 12.5% are a worst-case scenario, and additional to the 'normal' image noise. Fig. 10 shows what an image with 12.5% of the pixels erased looks like.

6.3. *Sequence disambiguation*

One problem that arises when using navigation based on sequences is that of sequence disambiguation. Under normal circumstances, the occurrence of sequences, such as (1) ABC; (2) XBZ or (3) DEFEG, is possible (each letter represents a random input vector). There are two different problems with these three sequences: (1) and (2) both share one common element (B); and one element (E) occurs in two different positions of sequence (3). In the first case, the successor of B can be either C or Z. In the second case, the successor of E can be either F or G. The correct answer depends on the history of the system. One possible solution relies on using a kind of *short-term* memory.



Fig. 10. (Colour online) Image, from the testbed, 12.5% erased. This amount of damage, additional to the normal noise, is well tolerated by the robot in all operation modes.

Kanerva proposes a solution in which the input to the SDM is not the last input D_t, but the juxtaposition of the last k inputs $\{D_t, D_{t-1} \ldots D_{t-k}\}$. This technique is called *folding*, and k is the number of $folds$. The disadvantage is that it greatly increases the dimensionality of the input vector. Bose[16] uses an additional neural network, to store a measure of the *context*, instead of adding folds to the memory.

In the present work, a solution inspired by Jaeckel and Karlsson's proposal of segmenting the addressing space[17,18] seemed more appropriate. Jaeckel and Karlsson propose to fix a certain number of coordinates when addressing, thus reducing the number of HLs that can be selected. Since, when following a given sequence, the robot is looking for an image just within the sequence that is being followed, this idea can be applied here. The number of the sequence can be *fixed*, thus truncating the addressing space.

A second navigation algorithm was written, which makes use of a *context memory*, where it stores the sequence number of the last k predictions. The sequence s_n that has more entries in the context is considered the best prediction for the sequence that is being followed. If the SDM predicts an image of a sequence s_j different from s_n, the memory is queried again for the best prediction within s_n, and this is considered the right prediction. The sequence number s_j is stored in the context anyway so that the robot is still able to localise itself correctly if kidnapped from one sequence to another, after a number of steps.

To compare the performance of this new algorithm with the old one, the memory was loaded with three different paths. Fig. 11 shows an example of the results obtained without and with context. The black lines (1-4, 2-5 and 3-6) are the paths that were taught. The green and red lines were drawn, respectively, in the arithmetic and S09 modes. The lines were drawn by pens attached to the rear of the robot, so they represent the motion of its back, not its centre. The results shown were obtained in the arithmetic mode, enforcing distribution to a minimum of three locations, but they are similar for the other operation modes or other numbers of copies.

It is clear that without context the robot sometimes skips from one sequence to another when paths cross or come very close, thus not being able to complete the path. A context of just three images solves this particular problem although the results shown were obtained with a context of 15.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

Table IV. Results obtained under overflow situations of increasing severity. The robot 'forgets' the older paths first, and with data distribution it tolerates overflows of above 20% the maximum capacity.

Min. copies	Limit (%)	Limit (HL)	Arit 1	Arit 2	Arit 3	NBC 1	NBC 2	NBC 3	OC 1	OC 2	OC 3	S09 1	S09 2	S09 3
1	85	186	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	80	175		Y	Y	Y	C	Y		Y	Y	C		Y
	75	164			Y		C	Y			Y	Y		
	70	153			Y			C						
	65	142			Y			C						
	60	131			Y									
3	85	558	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	80	526		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	75	493		Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
	70	460		Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
	65	427		Y	Y		Y	Y	Y	C	Y			Y
	60	394		Y	Y		C	Y	Y		Y			Y
	55	361		C	Y		C	Y			Y			
	50	329			Y			C			Y			
	45	296			Y			Y			Y			
	40	263			Y			C			Y			
	35	230			Y						Y			

6.4. Memory overflow

In the original Kanerva model, the memory contained a fixed number of HLs. Those HLs were randomly placed in the addressing space before operation of the memory. Therefore, memory overflows could not occur. However, many authors prefer to use scalable algorithms, such as Ratitch *et al.*'s Randomised Reallocation Algorithm (the one used here).[12] In this algorithm, new HLs are added when a datum cannot be stored in enough existing locations. If the memory already contains the maximum set number of locations, some existing locations are deleted. Various approaches can be followed to select locations for deletion. However, simple random selection is appropriate because it is simple and makes older locations statistically more vulnerable. This means that the memory is expected to exhibit the human characteristic of most likely forgetting the older information.

To test this behaviour, the memory was loaded with three sequences (72, 61 and 85 images, respectively), in memory overflow situations of increasing severity. The maximum number of HLs for the first experiment was the exact number of locations required. In subsequent experiments, it was decreased in steps of 5%, until the system was not able to follow any of the sequences. In all the situations, the context was set to 15 steps.

Table IV shows the results obtained. The first column indicates the minimum number of copies of each image. The second and third specify the maximum number of existing HLs, in percentage of the number of HLs needed to store the complete sequences and in absolute value. The next three columns indicate if the robot was actually able to successfully follow, respectively, paths 1, 2 or 3, in the arithmetic mode. 'Y' means that it was able to follow the path with just a minimum error. 'C' means that it was very close, but the drift was above its normal performance. An empty cell means that the robot got lost or jumped to another path, thus not being able to complete the task. The other nine columns show the same, for the other operation modes. The results are not shown for the limits of 90 and 95%, because they are similar to the limit of 85%.

Table IV shows that the system 'forgets' the old paths as the new ones require more memory. Nonetheless, an overload of 15–20% was well tolerated, before the robot was unable

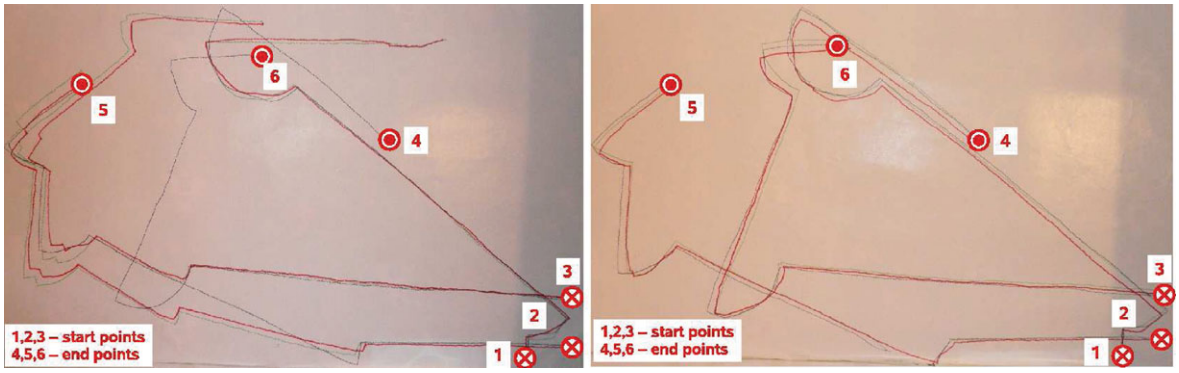


Fig. 11. (Colour online) Navigation results obtained without and with context. The use of a short-term memory prevents the robot from getting lost.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

to follow the first path that it had learnt. If the sequences are stored in reverse order (i.e. sequence 3 is the first to be learnt and sequence 1 is the last), the result is similar: the first to be learnt is the first to be forgotten.

7. Discussion

The results presented here show the feasibility of the approach proposed: navigation of a robot using sequences of images stored in a SDM. This approach is highly tolerant to severe situations, such as a significant change in illumination conditions, occlusion or memory overflow. Moreover, this robustness arises from the natural characteristics of the memory. Some other common navigation problems, such as the kidnapped robot problem, are also naturally solved in a vision-based navigation approach with limited context information, such as the one proposed. Another interesting characteristic is that backwards navigation is also possible: the SDM makes it possible to retrieve sequences in either direction.

The characteristics of the SDM make it suitable to determine overlapping sequences based on similarity of consecutive images. This way node points can be established when two paths come together or split, building a partial topological representation of the space. However, using similarities between images, the connection points can only be detected if the robot approaches a known path in a very close heading. This drawback could only be solved using metric information—but this problem has to be left as future work.

Compared to more *conventional* approaches, such as Support Vector Machines, Hidden Markov Models or even Neural Networks, the SDM has the advantage of exhibiting the most interesting characteristics of all of them, and only a few disadvantages. It is biologically inspired, easy to plan, implement and train. It is also easily scalable and can be implemented using parallel or distributed processing.

It can be noted that the SDM, if configured to store just one copy of each datum, actually works as a k-nearest neighbour model with sophisticated learning and forgetting algorithms. Data distribution makes the model more robust to noise and incomplete data. Compared to neural networks, the SDM has the advantage of being a more straightforward approach, easier to set up and scale. Compared to Support Vector Machines and Hidden Markov Models, the SDM is easier to train and deals better with associativity and forgetting, thus being more versatile.

The SDM simulated in software is very computer intensive because it has to be implemented based on linked lists or neural networks. Additionally, storage can be as low as 0.1 bits per bit of traditional memory. However, for production systems, it should be implemented in hardware, thus achieving much better performance at lower cost.[16]

8. Conclusions

To the best of the authors' knowledge, this is the first time a robot has actually been navigated by a system with a SDM at its helm. It was guided using view sequences stored into the SDM. The behaviour of the system was tested using different forms of encoding the information, with and without data distribution. The system proved to be quite robust to illumination changes as well as in the presence of interferences and overflow. Using only visual information, a partial topological representation of the environment was also built, making it possible to plan paths using A*. It is a viable and interesting model, making it possible to mimic behaviours similar to the human brain. It was proven that the way information is encoded, as well as distribution of the data, may have a significant impact on the performance of the system.

The advantages of using the SDM are immense. It is a very sound model, with a solid mathematical foundation. It *naturally* provides characteristics that otherwise may have to be hard coded, such as the ability to deal with sequences, backwards motion, immunity to noisy data and natural memory decay. Drawbacks include the problem that, when implemented in non-specific hardware, it requires a lot of processing and provides storage of only 0.1 bits per bit of traditional computer memory.

References

1. J. Hawkins and S. Blakeslee, *On Intelligence* (Times Books, New York, 2004). USA.
2. P. Kanerva, *Sparse Distributed Memory* (MIT Press, Cambridge, 1988). Massachusetts, USA.
3. R. M. Harnish, *Minds, Brains, Computers: An Historical Introduction to the Foundations of Cognitive Science* (Wiley-Blackwell, 2002). Massachusetts, USA.
4. R. P. N. Rao and O. Fuentes, "Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory," *Mach. Learn.* **31**(1–3), 87–113 (Apr. 1998).
5. M. Watanabe, M. Furukawa and Y. Kakazu, "Intelligent agv driving toward an autonomous decentralized manufacturing system," *Robot. Comput.-Integr. Manuf.* **17**(1–2), 57–64 (Feb.–Apr. 2001).
6. Y. Matsumoto, M. Inaba and H. Inoue, "View-Based Approach to Robot Navigation," *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)* (2000). Takamatsu, Japan.
7. S. D. Jones, C. Andresen and J. L. Crawley. "Appearance Based Processes for Visual Navigation," *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Grenoble, France (Sep. 1997).
8. N. Winters and J. Santos-Victor, "Mobile Robot Navigation Using Omni-Directional Vision," *Proceedings of the 3rd Irish Machine Vision and Image Processing Conference (IMVIP '99)* (1999) pp. 151–166. Dublin, Ireland.
9. M. O. Franz, B. Schölkopf, H. A. Mallot and H. H. Bülthoff, "Learning View Graphs for Robot Navigation," *Auton. Robots* **5**(1), 111–125 (Mar. 1998).
10. P. Jensfelt, D. Kragic, J. Folkesson and M. Björkman, "A Framework for Vision Based Bearing only 3D SLAM," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'06)*, Orlando, FL, USA (2006).
11. M. Mendes, M. Crisóstomo and A. P. Coimbra, "Robot Navigation Using a Sparse Distributed Memory," *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA (May 2008).
12. B. Ratitch and D. Precup, "Sparse Distributed Memories for On-Line Value-Based Reinforcement Learning," *Proceedings of the European Conference on Machine Learning (ECML)* (2004). Pisa, Italy.
13. S. B. Furber, J. Bainbridge, J. M. Cumpstey and S. Temple, "Sparse Distributed Memory Using *n*-of-*m* Codes," *Neural Netw.* **17**(10), 1437–1451 (2004).

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press

14. M. Mendes, A. P. Coimbra and M. Crisóstomo, "AI and Memory: Studies Towards Equipping a Robot with a Sparse Distributed Memory," *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, Sanya, China (Dec. 2007) pp. 1743–1750.
15. M. Mendes, M. Crisóstomo and A. P. Coimbra, "Assessing a Sparse Distributed Memory Using Different Encoding Methods," *Proceedings of the 2009 International Conference of Computational Intelligence and Intelligent Systems*, London, UK (Jul. 2009) pp. 37–42.
16. J. Bose, "A Scalable Sparse Distributed Neural Memory Model," *Master's Thesis* (Manchester, UK: University of Manchester, Faculty of Science and Engineering, 2003).
17. L. A. Jaeckel, "An Alternative Design for a Sparse Distributed Memory," *Technical Report*. Research Institute for Advanced Computer Science, NASA Ames Research Center (Jul. 1989).
18. R. Karlsson, "A fast activation mechanism for the kanerva SDM memory." Proceedings of the 95 RWC Symposium, pp. 69–70, Tokyo, Japan, June 1995.

https://doi.org/10.1017/S0263574711000828 Published online by Cambridge University Press