Robot Navigation and Manipulation based on a Predictive Associative Memory

Sascha Jockel, Mateus Mendes, Jianwei Zhang, A. Paulo Coimbra and Manuel Crisóstomo

***Abstract*—Proposed in the 1980s, the Sparse Distributed Memory (SDM) is a model of an associative memory based on the properties of a high dimensional binary space. This model has received some attention from researchers of different areas and has been improved over time. However, a few problems have to be solved when using it in practice, due to the non-randomness characteristics of the actual data. We tested an SDM using different forms of encoding the information, and in two different domains: robot navigation and manipulation. Our results show that the performance of the SDM in the two domains is affected by the way the information is actually encoded, and may be improved by some small changes in the model.**

***Index Terms*—Sparse distributed memory (SDM), EPIROME, episodic memory, associative memory, navigation, manipulation, robotics.**

I. Introduction

Since the inception of Artificial Intelligence, an aim of research has been to build machines able to exhibit human-like behaviours. Current evidence [1], [2] supports the idea that intelligence is, to a great extent, more the result of using a large specialised memory than the result of heavy computation.

In the 1980s, Pentti Kanerva proposed the model of a Sparse Distributed Memory [1]. The properties of this model are derived from the mathematical properties of high dimensional binary spaces and thus provide a solid ground for the theory. Such a memory must be very tolerant to noisy data, learn in at the first attempt, forget in a natural way and exhibit other human long-term memory characteristics. Kanerva's original model has received considerable attention from the research community. The first major implementation of the SDM was probably the Stanford Prototype, a huge hardware implementation made at Stanford University in 1989 [3]. But the SDM has also been subject to several suggestions for improvements. L. Jaeckel [4] and R. Karlsson [5], e.g., propose an alternative addressing mechanism that may improve the performance of the model at the cost of reducing the addressing space. D. Hely et al. [6] propose a signal model which permits the memory to adjust its structure according to the stored data, thus improving its performance with non-random data. B. Ratitch et al. [7] propose an alternative architecture which might perform better with nonrandom data, at the cost of possibly degrading some characteristics of the original model.

Related applications of the SDM can be found in the Learning Intelligent Distribution Agent (LIDA) architecture, a cognitive architecture that comprises modules for perception, various types of memory, "consciousness," action selection, deliberation, and violation [8]. LIDA employs an SDM as its major episodic memory [9]–[11]. Also in neuroscience, the *sparse coding* strategy is an appropriate theory on the neural coding of sensory inputs [12]. Several theoretical, computational and experimental studies suggest that neurons encode sensory information using a small number of simultaneously active neurons out of a large population at any given point in time. For each stimulus to be encoded, there is a different subset of all available neurons. This is the case in an SDM.

This memory, if applied to robotics, could provide robots with behaviours which are otherwise difficult to implement, such as the ability to learn in a natural way, reinforce important ideas and forget unused ones. And such robots would also be very tolerant to noisy data with graceful degradation.

Rao and Fuentes [13] were probably the first to use an SDM in the robotics domain. Their model was a hierarchical system in which the lower levels were responsible for tasks such as collision detection and obstacle avoidance. Simulations were then made on the use of an SDM, implemented as a Neural Network, to navigate a robot. The SDM was trained by the user who manually guided it through the desired path. During learning, the sensorial information from optical sensors was recorded, and the system was expected to later follow the same path by using this information. The authors report simulation results only.

We implemented a versatile version of the SDM, which was tested in two different tasks: navigation based on a view sequence, and manipulation based on a sequence of joint angles of a motion trajectory. The original SDM model was proposed to deal with random information, but information in these domains is hardly purely random. Therefore, the performance of the system might suffer because of the nature of the information it is dealing with. To overcome this problem, we studied different forms of encoding the information: the use of natural binary code; using a different sorting of the bytes; grouping the bits as integers or floating point numbers; and using a sum-code. Our results show that the architecture of the memory and the way information is

S. Jockel and J. Zhang are with CINACS International Research Training Group, Technical Aspects of Multimodal Systems (TAMS), Department of Informatics, University of Hamburg, Hamburg, Germany. Email: {*jockel,zhang*}@*informatik.uni-hamburg.de*

M. Mendes is with Escola Superior de Tecnologia e Gestão de Oliveira do Hospital (ESTGOH), Instituto Politécnico de Coimbra (IPC), Coimbra, Portugal. Email: *mmendes@estgoh.ipc.pt*

M. Mendes, A. P. Coimbra and M. Crisóstomo are with ISR - Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra, Portugal. Email: {*acoimbra,mcris*}@*isr.uc.pt*

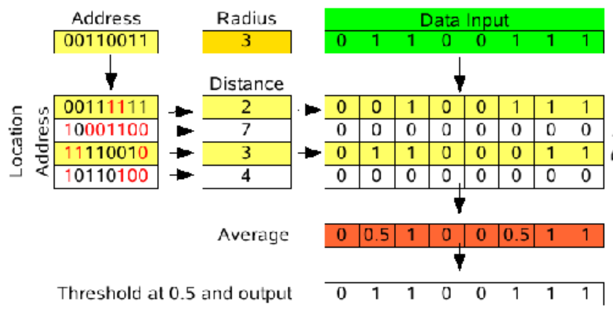

Fig. 1. One model of an SDM.

encoded is not irrelevant—they may have a significant impact on its performance and the use of a sum-code, for example, might significantly improve the performance of the SDM.

In Section II we give a brief overview of Kanerva's SDM. In Section III we depict effects that might be caused by different methods of encoding the information. Section V outlines the use of an SDM for vision-based robot navigation and robot manipulation. Results are presented and discussed in Section VI, and conclusions in Section VII.

II. 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* (HL). As a general guideline, those hard locations should be uniformly distributed in the virtual space, to *mimic* a larger virtual space as accurately as possible. Every datum is stored in a distributed manner in a set of hard locations, and retrieved by *averaging* those locations within a certain radius. Therefore, recall may not be perfect—accuracy depends on the saturation of the memory.

SDMs can be used as associative memories. According to the theory, knowing only 20% of the bits and setting the remaining 80% at random should be enough to statistically retrieve the right datum with a 0.6 probability [1]. Other characteristics of the SDMs are also significant, such as high tolerance to noise [14], [15], robustness to failure of individual locations and graceful degradation, one-shot learning, suitability to work with sequences [1], [16] and phenomena typical of human memory, such as *knowing that one knows* or *tip of the tongue*. Detailed descriptions and demonstrations of the mathematical properties of the memory can be found in [1] and [17], [18].

Figure 1 shows a model of an SDM. In this model, the main modules are an array of addresses, an array of bits, another module that sums and averages the bits, and a fourth module where the sums are thresholded to produce the output vector.

"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, it will activate all the addresses within a predefined access radius. For every read or write operation, the distance between the input address and every other address in the memory has to be calculated. Different metrics can be used to compute the distance. Kanerva proposes that the Hamming distance, which is the number of bits by which two binary vectors differ, can be used. However, the most appropriate metric to be used in distance calculations depends on the codification of the data, as we will show later.

In this model, data are stored at the arrays of bits. In the original proposal by Kanerva, those were arrays of bit counters, and writing was performed by incrementing a bit counter to store 1, or decrementing to store 0. Unfortunately, that approach is not practical—it increases the system complexity, requires a lot of processing and increments the response time significantly. Furber et al. [19] 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. In this case, writing to a hard location simply means to replace its old contents, greatly simplifying the system and improving its response time.

Reading is done by summing up the bits (or bit counters, in the original version) column-wise, averaging and thresholding at a predefined value. If the value of the sum is below the threshold, the bit shall be considered zero, otherwise it shall be considered one. For a memory where arrays of bits are used, 0.5 is an appropriate threshold value.

Initially, all the bit arrays must be set to zero, so that the memory stores no data. As for the bits of the address locations, Kanerva proposes to set them randomly, so that the addresses are uniformly distributed in the addressing space. In our implementation, we fill the virtual space according to Ratitch et al's Randomised Reallocation algorithm [7], instead of placing hard locations randomly in the addressing space during an initialisation phase. The algorithm starts with an empty memory and allocates new hard locations when there is 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 [7], [17].

III. Practical Problems

According to its theory, the properties of the SDM should hold if the information is purely random. In robotics and many other domains, though, the information is hardly purely random. Sensorial data is more likely to be represented using the *Natural Binary Code* (NBC), and the values hardly use all the available representation range uniformly.

In NBC, the value of each bit depends on its position. `01` is different from `10`, although the number of ones and zeros in each number is exactly the same. Additionally, the number of ones can increase and decrease as the represented value of the number grows. `0100` contains less ones than `0011`, but its binary value is larger than `0011`.

Table I shows the Hamming distances between all the 3-digit binary numbers. As can be seen, this distance is not proportional to the arithmetic distance. The Hamming distance sometimes even decreases when the arithmetic distance increases. One example is the case of `001` to `010`, where the arithmetic distance is 1 and the Hamming distance is 2. If we compare `001` to `011`, the arithmetic distance increases to 2 and the Hamming distance decreases to 1. In total, there are 9 *undesirable* transitions for the 3-bit sequence, where

TABLE I
HAMMING DISTANCES BETWEEN 3-BIT NUMBERS.

	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

the Hamming distance decreases while it should increase or, at least, maintain its previous value. Therefore, if NBC code with Hamming distances is used, the performance of the SDM is affected. The Hamming distance is not appropriate for use with positional codes, as in the NBC code. It would be appropriate for bit-sized or random data.

IV. SDM IMPLEMENTATIONS

In order to find a better solution to the problem of non-random data encoding and its distance calculation, four different solutions have been investigated: 1) NBC code and Hamming distance (bitwise mode); 2) Optimised binary coding and Hamming distance; 3) Sum-code and Hamming distance; and 4) NBC code and arithmetic distance. NBC and the Hamming distance were described above; the other alternative solutions are described in the following subsections.

A. Optimised binary coding for the Hamming distance

One possible solution to the problem of using the Hamming distance with non-random NBC data is to use a coding so that the Hamming distance becomes proportional to the arithmetic distance—or, at least, does not exhibit so many undesirable transitions.

This coding can be accomplished by simply trying different permutations of the numbers and computing the matrix of Hamming distances. For 3-bit numbers, there are 8 different numbers and $8! = 40320$ permutations, which can be easily computed in a reasonable time. Unfortunately, for large numbers of bits the processing time increases exponentially.

Despite the possibly long computation time, for a finite domain the numbers can simply be sorted one time and then the input and output vectors can be translated using a simple look-up table. Anyway, despite this approach of reordering the bytes being straightforward if NBC is used to represent integers, it does not hold if the information to store in the SDM is composed of floating point numbers. Floating point numbers are manipulated in scientific notation, which means they are described by an exponent and a mantissa. This exponent and mantissa are stored as two different sets of bits, making the sorting impractical, if not completely impossible. Therefore, this solution was dropped in our work.

TABLE II
SUM-CODE TO REPRESENT 5 DIFFERENT VALUES.

0	0000
1	0001
2	0011
3	0111
4	1111

B. Use of a Sum-code and Hamming Distance

With the hamming distance calculation, one way to completely avoid undesirable transitions is to reduce the number of values represented to the number of bits + 1. Therefore, when using 4 bits we can only use 5 different numbers, as shown in Table II. Using 8 bits, we can use 9 values and so on. This is the only way to work with a Hamming distance that is proportional to the arithmetic distance. This approach is similar in many aspects to Vanhala's [20] idea of considering only the most significant bit. However, while Vanhala was more concerned about reducing noise, we are considering the similarity measure of items in the SDM—and in this aspect Vanhala's approach is very inadequate.

Our proposal is to use a sum-code, which consists of expressing a number y within an interval $[a, b] = \{y \in \mathbb{Z} | a \leq y \leq b\}$ with the number of 1-bits according to Equation 1. This representation is known as the *sum-code*, a derivate of *1-from-n* code.

$$sc(y) = y - a \tag{1}$$

The sum-code may lead to a significant extension of the dimensionality of the input vector. Since an SDM is particularly designed to cope with high-dimensional input vectors, it should not be a problem.

Note that in an unsorted sum-code there may be many different possibilities to represent the same number by shifting the bits. For example, the number 3 in a 4-bit code may have four different representations: `0111;1011;1101;1110`. Thus, similar distances to hard locations may occur.

C. Arithmetic Mode

Inspired by Ratitch et al's work [7], we also tested the SDM with arithmetic distance calculations and NBC coding (arithmetic mode). In this variation of the model, the bits are grouped as numbers, either integers or reals, as shown in Figure 2. 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:

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

h_t^k is the k^{th} number of the hard location, at time t. x^k is the corresponding number in the input vector x and α the learning rate. $\alpha = 0$ will 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 is the value that we use. However, this means the memory may loose its one-shot learning ability.

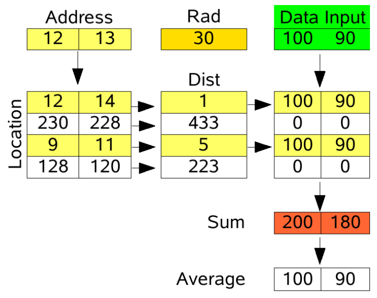

Fig. 2. Arithmetic SDM, which works with byte integers and arithmetic distances.

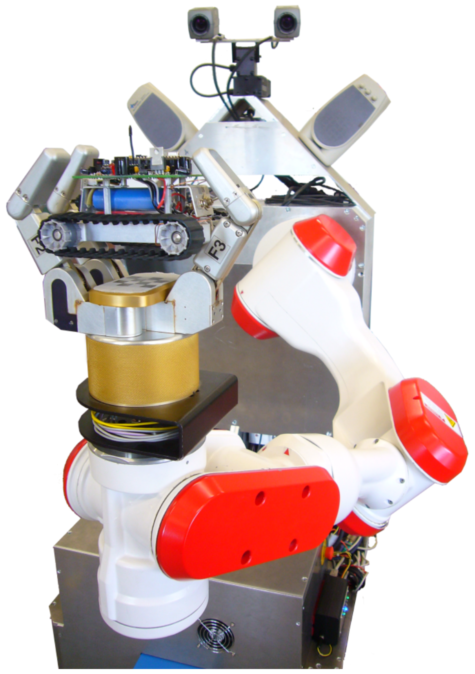

Fig. 3. Platforms used in the experiments: TASER's robotic arm is grasping small robot Lizard.

V. Test Platforms

Our tests were performed on two different platforms, designed for different purposes: one for navigation (Lizard), the other for manipulation (TASER), as shown in Figure 3. In both cases the memory was used to store sequences of events which were later to be repeated.

A. Lizard and SDM for Navigation

Lizard is a small robot as described in [15]. It is equipped with a camera, and navigation is based on a sequence of images. During a supervised learning stage, images and additional information (such as the motion that leads the robot to the next image position) are stored in the SDM. During the autonomous run, the robot uses its camera view as an address to retrieve the learnt path from its memory.

Input and output vectors consist of arrays of bytes, meaning that each individual value must fit in the range [0, 255]. Every individual value is, therefore, suitable to store the gray level value of an image pixel or an 8-bit integer. The composition of the input vectors is as summarised in Table III and Equation 3:

TABLE III
Summary of the total dimensions of the input vector.

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

$$\vec{x}_i = < im_i, seq\_id, i, timestamp, motion > \qquad (3)$$

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

The memory is used to store vectors as explained, but addressing is done using just one image. During the autonomous run, the robot grabs an image, identifies its position by recognising the closest image of the learnt sequence (im_i) and performs the motion associated with that image. Then it grabs a new image and so on. If the robot detects a horizontal shift between the grabbed image and the stored one, it tries to correct it iteratively by turning left or right until the displacement is below 5 pixels.

Addressing is done using only im_{i-1}, not the whole vector. The remaining 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 remaining bits. This saves computational power and reduces the probability of false positives being detected. As for the activation radius, it is automatically adjusted based on the noise value detected in the images, as described in [17].

B. TASER and SDM for Manipulation

TASER is a service robot of human height (see Fig. 3). The robot system consists of a mobile platform with a differential drive and wheel encoders, two laser range finders, a Pentium IV 2.4 GHz industrial computer as central control unit, a PA10-6C manipulator, a three-finger robotic hand and several IEEE1394-cameras (stereo head with a pan-tilt-unit and omnidirectional vision system). In this work SDM learning is constrained to the 6-degree-of-freedom robot arm trajectory for manipulation tasks.

Manipulation is based on a sequence of 3-D coordinates and roll, pitch, and yaw angles of the manipulation tool. The inverse kinematics to route from one point to another is computed by the Robot Control C Library (RCCL). During a learning phase, the 6 joint angles[1] of the robot arm as well as the tool centre point (TCP) and the tool orientation are stored

[1]Each joint has a software limited operation range up to $\pm 255°$.

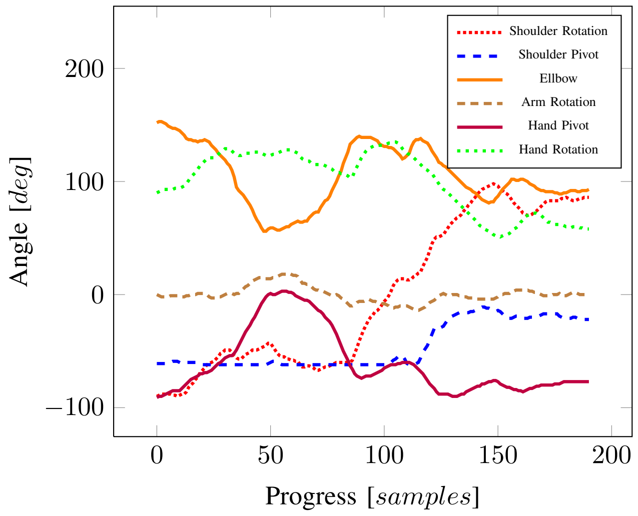

Fig. 4. A single sample trajectory of the 6-DoF robot arm stored to the SDM. The line graph only represents the joint angles that correspond to $j_1, \ldots, j_6$ of $\vec{x}_i$ (see Equation 4). Further information of $\vec{x}_i$, also stored to SDM, is omitted for the sake of clarity.

to the SDM with a sampling rate of 6-10Hz[2]. Although we can operate our robot arm either by joint angles or TCP and tool orientation, we only use the latter parameters during the learning phase for robot arm control but store joint angles in our memory, too. During an autonomous task execution, the robot is supposed to recognise its current joint constellation and follows the learnt trajectory from its memory.

The input and output vectors consist of arrays of doubles. The contents of a vector in our SDM implementation is shown in Equation 4.

$$\vec{x}_i =< j_1, j_2, j_3, j_4, j_5, j_6, x, y, z, \chi, \psi, \omega, seq\_id, i >, \quad (4)$$

where each j_n is the angle of the corresponding arm joint. The 3D coordinates of the tool centre mounted at the end of the robot arm in relation to the robot coordinate system are depicted with x, y, z, and χ, ψ, ω describe the roll, pitch, and yaw tool orientation. Each of the above-mentioned variables is an 8-byte double precision value. Finally, the seq_id is a unique 4-byte id for each sequence, and i is unique for each vector.

Addressing the memory is done only by presenting a single arm position by its corresponding 6-joint angles. During an autonomous execution phase, the SDM will predict j_n from the corresponding j_{n-1}.

VI. Main Results, Comparison and Discussion

Different tests were performed in order to assess the behaviour of the two systems using the approaches described in Section IV. The results are summarised in Table IV. The upper half of the table shows results obtained while the memory contained only one copy of each datum (i.e., there is no distribution of the data). The bottom half contains the results of the same tests, but this time obtained with 5 copies of each datum stored, i.e., distribution of the data is enforced. The values shown are: the distance from the input address to the closest hard location HL (the most similar image); the distance from the input address to the second closest hard location; and the average distance from the input address to all the hard locations in the memory. There is also a measure of the increases, in percentage, which somehow expresses how successful the system is in separating the desired datum from the pool of information in the SDM (columns 6 and 8).

The table also shows the number of prediction errors. We propose that a prediction error occurs every time there is a step back in the sequence. For instance, if at time t and $t+1$ the predictions are, respectively, x and $x-1$, that is a prediction error, as the robots are not expected to get back in a sequence.

[2]Due to problems in coordinated timing at sampling phase, the rate ranges from 6-10Hz.

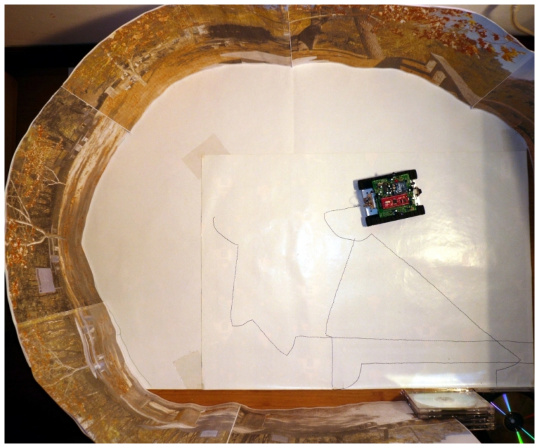

Fig. 5. Lizard in the testbed.

A. Navigation

The testbed used for navigation is shown in Fig. 5. It consists of a large white sheet of paper, circumvented by an artificial scenario. The black lines are examples of paths that the robot learnt.

The results described here were obtained using a sequence of 55 images, which were equalised before processing. The table shows the average result of 30 tests, except for the navigation errors, which are the average of six. The tests were performed using the arithmetic mode and the bitwise mode (8 bits per pixel, 256 gray levels represented using the natural binary code and the Hamming distance); and the sum-code mode using 16 bits per pixel[3] (17 gray levels). In each case, the access radius was computed dynamically ("Dyn." in Table IV) in the beginning of the sequence, and set to 20 % above the noise level [15].

B. Manipulation

The results for SDM-based manipulation were obtained using a sequence consisting of 256 arm configurations. The memory was just triggered by a single initial arm position that was presented to the memory. Figure 4 shows at least the

[3]Note that in this mode, the dimensionality of the vector almost doubles.

TABLE IV
COMPARISON OF THE PERFORMANCE OF THE SDM IN TWO DIFFERENT DOMAINS: NAVIGATION AND MANIPULATION.

Domain	Operation mode	Distr. Copies	Dist. to closest HL	Dist. to 2^{nd} closest HL	Inc. %	Aver. dist. to all HLs	Inc. %	Errors	Access Radius	Vector Bits
Manipulation	Arithmetic	1	86.40	103.97	20.33	1797.54	1980.49	0	10	832
	Bitwise		14.17	33.07	133.41	63.52	348.37	7		832
	Sum-code		13.10	82.40	529.01	1274.55	9629.36	0		22384
Navigation	Arithmetic	1	18282	118892	550.32	166406.53	810.22	13.2	Dyn.	41064
	Bitwise		6653	9186	38.07	9724.80	46.17	15.3		41064
	Sum-Code		569	3791	566.26	5257.01	823.90	13.8		82024
Manipulation	Arithmetic	5	12.35	14.4	17.32	1275.30	10227.33	0	15	832
	Bitwise		2.87	6.03	110.47	62.83	2091.90	0		832
	Sum-code		12.67	12.67	0.0	1271.31	9936.67	0		22384
Navigation	Arithmetic	5	33349	33503	0.46	145765.32	337.09	15.3	Dyn.	41064
	Bitwise		7167	7298	1.83	9359.14	30.59	16.9		41064
	Sum-code		2214	2346	5.97	9184.19	314.84	15.2		82024

joint angles $j_1, \ldots, j_6$ of $\vec{x}_i$ (cf. Equation 4) of an example trajectory stored to the memory.

Table IV shows the average of 30 predictions. The tests were performed using the arithmetic mode, the bitwise mode and the sum-code mode. The latter uses as many bits as for the range for the particular values in our input vector, in our case 22384 in total (cf. Equation 1). An access radius was chosen experimentally by the users.

C. Discussion

The randomness characteristic of the SDM means that the results obtained with it may be different even in very close circumstances. However, we can emphasise the robustness of the models we implemented, in face of the consistent results we obtained in two completely different domains: the domain of vision-based navigation, in which the data is of very high dimensionality and contains a high amount of noise, and in the domain of robot manipulation, in which the data contains small amounts of noise and the dimensionality is relatively small. In both cases, the models proved adequate and behaved as the SDM theory predicts.

In the case of vision-based navigation, the results show a higher number of sequence prediction errors, which is probably due to the presence of noise in the images, but in no way seemed to compromise the ability of the robot to successfully follow the sequences in which the images were rich enough.

In the case of manipulation, the high precision of the robot arm shows—if at all—only a very small level of noise. Robot arm trajectories are successfully learnt and followed. One interesting characteristic of this approach is that regardless of the initial position, the robot is able to converge to the closest point in the sequence, thus always executing the assigned task with success. This happens in manipulation, because we are using absolute positioning of the joints, but not in navigation, where the robot has no information on absolute positioning.

VII. CONCLUSIONS AND FUTURE WORK

Recent evidence shows that what we recognise as intelligent behaviour is often the result of using a sophisticated memory, rather than the result of heavy computation. The SDM is one model of associative memory that is able to mimic many characteristics of the human long-term memory, such as the ability to learn in a single pass, work with high-dimensional features and the ability to work with incomplete or noisy data.

We described and compared different models of the SDM, applied to two distinct subfields of robotics: vision-based navigation and robot manipulation. Despite those being two very different problems, the approach was the same: use supervised learning to teach the robot sequences of steps to follow a path (navigation), or to interact with an object (manipulation), and later retrieve those sequences from the memory. During learning, the robots store sensorial readings into their memories, and those readings shall guide them later to repeat the tasks during the autonomous runs.

One of the most difficult problems one faces is that the SDM theory uses random data, and real-world sensorial inputs are hardly random. Therefore, the performance of the memory is significantly affected. To overcome this problem, we studied four different forms of encoding the information: using the natural binary code and the Hamming distance; using an optimised sorting of the numbers; using an arithmetic distance; and using a sum-code and the Hamming distance.

Our results show that the same model of SDM can successfully accomplish the tasks of navigating the robot and manipulating objects. The arithmetic distance and the sum-code exhibit a better performance in both domains of application.

As drawbacks of this approach we can point out that the arithmetic mode may reduce some of the original characteristics of the model, while the natural binary code exhibits the worst performance and the other methods require additional processing.

Future work may include the study of the impact of each different encoding method on the natural characteristics of the SDM, as well as refining the navigation and manipulation algorithms used.

References

[1] P. Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, USA, 1988.

[2] J. Hawkins. *On Intelligence*. Times Books, New York, 2004.

[3] M. J. Flynn, P. Kanerva and N. Bhadkamkar. *Sparse Distributed Memory: Principles and Operation*. Tech. Report, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, California, USA, 1989.

[4] L. A. Jaeckel. *An Alternative Design for a Sparse Distributed Memory*. Tech. Report, Research Institute for Advanced Computer Science, NASA Ames Research Center, USA, 1989.

[5] R. Karlsson. *A Fast Activation Mechanism for the Kanerva SDM Memory*. In Proc. of the 95 RWC Symp., Tokyo, Japan, 1995.

[6] T. A. Hely, D. J. Willshaw and G. M. Hayes. *A New approach to kanerva's sparse distributed memories*. In IEEE Trans. on Neural Networks, pp. 101–106, 1999.

[7] B. Ratitch and D. Precup. *Sparse Distributed Memories for On-Line Value-Based Reinforcement Learning*. In ECML, 2004.

[8] S. Franklin and M. Ferkin. An ontology for comparative cognition: A functional approach. *In Comparative Cognition & Behavior Reviews*, 1:36–52, 2006.

[9] S. Franklin. *Perceptual memory and learning: Recognizing, categorizing, and relating*. In Symp. on Developmental Robotics, American Association for Artifical Intelligence (AAAI), Stanford University, Palo Alto CA, USA, 2005.

[10] A. Anwar and S. Franklin. *Sparse distributed memory for conscious software agents*. In Cognitive Systems Research, 4(4):339–354, 2003.

[11] S. Jockel, M. Weser, D. Westhoff and J. Zhang. *Towards an Episodic Memory for Cognitive Robots*. In Proc. of 6^{th} Cognitive Robotics workshop at 18^{th} European Conf. on Artificial Intelligence (ECAI), pp. 68–74, Patras, Greece, 2008, IOS Press.

[12] B. A. Olshausen and D. J. Field. Sparse coding of sensory inputs. *Current Opinion in Neurobiology*, 14:481–487, 2004.

[13] R. P. N. Rao and O. Fuentes. *Hierarchical Learning of Navigational Behavious in an Autonomous Robot Using a Predictive Sparse Distributed Memory*. In Machine Learning, pp. 87–113, Kluwer Academic Publishers, Boston, USA, 1998.

[14] R. P. N. Rao and D. H. Ballard. *Object Indexing using an Iconic Sparse Distributed Memory*. The University of Rochester, Computer Science Department, Rochester, New York, USA, 1988.

[15] M. Mendes, M. Crisóstomo, and A. Paulo Coimbra. *Robot navigation using a sparse distributed memory*. In IEEE Intl. Conf. on Robotics and Automation (ICRA), Pasadene, CA, USA, 2008.

[16] J. Bose, S. B. Furber and J. L. Shapiro. *A spiking neural sparse distributed memory implementation for learning and predicting temporal sequences*. Intl. Conf. on Artificial Neural Networks (ICANN), Warsaw, Poland, 2005.

[17] M. Mendes, A. Paulo Coimbra, and M. Crisóstomo. *AI and memory: Studies towards equipping a robot with a sparse distributed memory*. In IEEE Intl. Conf. on Robotics and Biomimetics (ROBIO), pp. 1743–1750, Sanya, China, 2007.

[18] S. Jockel, F. Lindner and J. Zhang. *Sparse Distributed Memory for Experience-Based Robot Manipulation*. In Proc. of 2008 IEEE Intl. Conf. on Robotics and Biomimetics (ROBIO), pp. 1298-1303, Bangkok, Thailand, 2008.

[19] S. B. Furber, J. Bainbridge, J. Mike Cumpstey and S. Temple. *Sparse distributed memory using N-of-M codes*. In Neural Networks, 17(10):1437-1451, 2004.

[20] J. Vanhala, J. Saarinen and K. Kaski. *Sparse distributed memory for multivalued patterns*. In IEEE Intl. Conf. on Neural Networks, 1993.