AI and Memory: Studies Towards Equipping a Robot with 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  
mendes@estgoh.ipc.pt

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

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

Abstract—Traditional approaches to Artificial Intelligence (AI) and Robotics seem only to provide advances at a very slow pace. Many researchers agree that new, different approaches are needed to provide a breakthrough and allow the construction of robots with human-like capacities. Our approach consists in navigating a robot using vision a Sparse Distributed Memory (SDM), a kind of associative memory based on the properties of high dimensional binary spaces, which, in theory, exhibits some human-like behaviours. During learning the robot will store sequences of images in the SDM. During execution the robot will follow the sequence of images that is closest to its current view. Preliminary results show that the memory can store and predict sequences of images with a small error tolerance.

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

I. INTRODUCTION

It is often said that current Artificial Intelligence tools and approaches can’t provide the required framework to build truly intelligent machines, with human-like behaviours. Human intelligence relies mostly on our brain, whose working principles we don’t fully understand. Therefore, current AI models of the brain are, to a great extent, just speculative frameworks that need to be studied and tested, so that the flaws can be exposed and corrected, leading to more accurate models.

As for actual robots, they usually rely on limited amounts of memory and heavy processing. However, evidence seems to be that the human brain works exactly the other way: limited processing and huge amounts of memory, to store sequences of events [9], [10], [4] that will lead future actions. Therefore, it’s reasonable to assume that a human-like robot should rely, to a great extent, on an intelligent system with similar characteristics.

The Sparse Distributed Memory [10] is a mathematically sound model of a memory which intends to model some characteristics of the human brain. D. Rogers [17], A. Anwar et al [2], R. Rao and D. Ballard [15], Furber et al [7], [4], among others, have implemented SDMs and improved the original model, but the SDM has never been pushed too farther, despite some encouraging results.

II. THE BRAIN, INTELLIGENCE AND MEMORY

1) The brain as a memory system: The underlying reason for human intelligence may be the fact that we have a large neocortex, able to process large quantities of information up to a high level of abstraction. How those huge amounts of information are processed is still a mystery. The mystery is yet more intriguing as we find out that the brain performs incredibly complicated tasks at an incredibly fast speed. It is known that neurons take about 5ms to fire and reset. This means that our brain operates at about 200MHz—a frequency well below any average modern computer. One possible explanation for this awesome behaviour is that the brain performs many tasks in parallel. Many neurons working at the same time would contribute to the overall final result. This explanation, though, is not satisfactory for all the problems the brain seems able to solve in fractions of seconds, as J. Hawkins argues [9].

J. Hawkins proposes a simple thought experiment, which he calls the “one-hundred-step rule”. The problem is recognising if there is a cat in a picture. A human can do it in about half a second or less. Since the brain operates at about 200MHz, we can infer that it performed about 100 steps to recognise the cat, or traversed a chain of about 100 neurons. On the other hand, a computer would need to perform billions of steps to attempt to solve the same problem. Therefore, it is theorised, the brain must not work as a linear computer. It
must be operating like a vast amount of multi-dimensional computers working in parallel.

The theory of the brain working as a massive parallel super-computer, though attractive, is not likely to explain all the phenomena. This arises from the observation that many actions the human brain seems to perform in just fractions of a second cannot be done in parallel, for some steps of the overall process depend on the result of previous steps. A simple analogy, also from J. Hawkins, is that if one wants to carry one hundred stone blocks across a desert and it takes a million steps to cross the desert, one may hire one hundred workers to only cross the desert once, but it will, nonetheless, take one million steps to get the job done.

Based on the one-hundred step rule, J. Hawkins proposes that the human brain must not be a computer, but a memory system.

Additionally, although there’re noticeable physical differences between brain regions, those differences are only small. Based on these observations, V. Mountcastle [14] proposed that all the brain might be performing basically the same algorithm, the result being different depending only on the inputs. Even the physical differences could be a result of the brain wiring connections. Although this may seem an unrealistic proposal at first sight, many scientists currently endorse Mountcastle’s theory, as it can’t be proven wrong and explains phenomena which would be harder to explain assuming the brain is an enormous conglomerate of specialised neurons. One important observation is probably the fact that the brain is not static—it adapts to its environment and changes when necessary. People who are born deaf process visual information in areas where other people usually perform auditory functions, for example [9].

2) The memory-prediction framework: According to the previously explained principle that the brain is mostly a memory system, J. Hawkins [9] also proposes a new framework to explain brain’s behaviour. According to this framework, the neocortex is essentially a tool to store sequences of patterns and recall them auto-associatively to make predictions of what may happen or may have happened. Our ability to make those predictions is the basis of what we call intelligence.

According to the memory-prediction framework, our brain is essentially a large memory system, constantly predicting what will happen next and acting according to those predictions. In a known environment our actions are processed almost without us being aware of them. E.g., we drive everyday to work solely based on our old memories of what happened the days before, without paying attention to what we’re doing. When something unusual happens, such as a car accident, or someone crosses the street in an unexpected place, that violates our predictions and, in consequence, captures our attention. We stop acting based on our sequence of cases and start searching our memory for the most similar situation, in order to know how to behave in this new situation. If we’re in a new environment, we’re doing that all the time. Our senses keep feeding our brain patterns which keep it busy all the time. Those sequences of patterns are then stored as new sequences, which may later be retrieved—we learnt, so that we can later predict with better accuracy.

3) Invariant forms and HTMs: One of the most difficult tasks performed by the brain, according to the memory-prediction framework (MPF) explained in section II.-2, is that of creating invariant forms. Our computers can learn to do virtually anything, but they’re quite inefficient dealing with variations. Even the smallest variation can be troublesome for a modern computer, if not carefully programmed in advance. A modern computer is little more than a calculator, solving thousands of mathematical equations which have been programmed one by one. They have no understanding of what they are doing. All the models of the world which are the basis of their hard work had to be programmed in advance. Computers are unable to form their own models of the world, or to create invariant forms as our brains seem to do.

Jeff Hawkins has also been studying the problem of creating invariant representations, for they are the basis of their theory of the MPF. Dileep George and Jeff Hawkins [8] proposed the model of an Hierarchical Temporal Memory (HTM). This HTM, according to the theory, once exposed to some patterns of sensorial input, must be able to automatically create its own model of the world, much like what is believed the human brain can do. This internal model must guarantee the memory the ability to recognise familiar objects and predict the future in a very human-like way.

An HTM is composed of several nodes, organised in a hierarchy. Each node can be linked to several nodes below and one single node above, in a tree structure with one or more nodes on the top layer. The nodes at level one (lowest level) receive input from the outer world. The nodes at other levels receive input only from the level below—level two receives input from level one, level three from level two and so on.

Every node in the structure is able to learn to recognise a given pattern. When the overall system has learnt to recognise a given pattern, the tree can be seen as a Bayesian network of probabilities. The probabilities at the lowest level represent quantisations of the input patterns. The variables at intermediate levels represent object parts, while the ones at the top nodes represent complete objects. During the learning process the values stored at each node are changed according to the input pattern and the data fed back by the upper nodes. During recognition, the goal is to find the most likely set of stages at each module, so that the combination of all the states best explains the input pattern.

III. Sparse Distributed Memories

Years before Hawkins’ work, Pentti Kanerva [10] advocated the same principle as J. Hawkins: intelligence is probably the result of a memory of sequences and a little processing. Based on this assumption, Kanerva proposed the Sparse Distributed Memory (SDM) model, a kind of associative memory based on the properties of high-dimensional binary spaces. According to P. Denning [5], J. Albus [1] and D. Marr [12] independently developed similar theories, based on the observation of the structure of the human nervous system.
The underlying idea beyond 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 uniformly 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: 1) The space $2^n$, for $100 < n < 10^5$, exhibits properties which are similar to our intuitive notions of relationships between the concepts; 2) Neurons with $n$ inputs can be used as address decoders or a random-access memory; 3) Unifying principle: data stored in the memory can be used as addresses to the same memory; 4) Time can be traced in the memory as a function of where the data is stored.

A. The space $2^n$

As stated before, Kanerva’s Sparse Distributed Memory model is based on the properties of the space $\{0, 1\}^n$, which we will also refer to as $2^n$, or simply $N$, following the author’s notation. Below we summarise the most important concepts, relations and properties of $N$. A more detailed analysis can be found in [10] or [3].

1) Point and Vector: Any point in $N$ can be represented by an n-tuple of binary digits $< b_0, ..., b_{n-1} >$, or a binary number, $b_0...b_{n-1}$. Those binary numbers have no particular order. They are just points in the binary space, and it’s irrelevant in this analysis whether 001 comes before or after 000.

2) Difference: Difference between two points $x$ and $y$ is defined as the bits in which the points differ, or the exclusive or. It is commutative: $x - y = y - x = x \oplus y$

3) Distance: As for distance between two points $x$ and $y$, the Hamming distance is used. It is defined as the number of bits in which $x$ and $y$ differ, or the norm of the difference between $x$ and $y$: $d(x, y) = |x - y|$

Two points close to each other are considered similar. Distance, therefore, is a measure of similarity between two different patterns.

4) Betweenness: It is said point $y$ is between points $x$ and $z$ if and only if the distance from $x$ to $z$ is the sum of the distances from $x$ to $y$ and $y$ to $z$: $d(x, z) = d(x, y) + d(y, z)$

If point $y$ is between $x$ and $z$, it is represented as $x : y : z$. A consequence of the definition is that every single bit of $y$ coincides with the corresponding bit either in $x$ or $z$: $x : y : z$ if and only if $y_i = x_i \lor y_i = z_i, \ i = 0, ..., n - 1$

5) Orthogonality: Two points $x$ and $y$ are orthogonal (also said perpendicular or indifferent) if and only if the distance between $x$ and $y$ equals half the number of dimensions: $x \perp y$ if $d(x, y) = \frac{n}{2}$.

A noticeable property is that if $x$ is indifferent to $y$, it is also indifferent to its complement ‘$y$. $x$ is halfway between $y$ and its complement ‘$y$.

6) Circle: A circle of radius $r$ and centre $x$ is defined as the set of points less than $r$ bits distant from $x$: $C(r, x) = \{ y | d(x, y) \leq r \}$

B. Properties of $N$

1) $N$ as a spherical space: An $n$-dimensional unit cube can have its vertices placed into a sphere of diameter $\sqrt{n}$, the diameter being a Euclidean distance. Based on this observation, Kanerva establishes an analogy of the space $2^n$ as a spherical one. $N$ is, according to the analogy, the surface of a sphere with radius $\frac{\sqrt{n}}{2}$, exhibiting interesting properties:

- Any point of $N$ can be considered the origin of the space;
- A point and its complement are like two poles of the sphere, at distance $n$ and with the entire space in between them;
- Any points perpendicular to the poles and halfway between them, are like the equator.

2) Distribution of the space $2^n$: Since any point in the space can be considered the origin $O$, the number of points which are at distance $d$ from point $x$ is the number of combinations of $d$ of $n$ bits:

$$\binom{n}{d} = \binom{n}{d}$$ (1)

$N$ can, therefore, be described by the binomial distribution $B(n, p)$, where $p$ is the bit probability $p = \frac{1}{2}$ and $n$ is the number of dimensions of the space. Following Kanerva’s notation, we’ll denote this probability distribution function as $N(d)$. According to the probability theory, $N(d)$ has mean value $E(x) = np = \frac{n}{2}$, and variance $Var(x) = np(1 - p) = \frac{n}{4}$. Therefore, we can define a good normal approximation $F$ for $N(d)$, with mean $\mu = \frac{n}{2}$ and standard deviation $\sigma = \sqrt{\frac{n}{4}}$.

3) Tendency to orthogonality: Since the distribution of points in $N$ can be approximated by the Normal distribution, as stated above, an interesting property arises from that fact. 68% of the points lie within one standard deviation from the mean, in the interval $[\mu - \sigma, \mu + \sigma]$. 95% of the points are within two standard deviations, and 99.73% within 3 standard deviations. The interval $[\mu - 5\sigma, \mu + 5\sigma]$ accounts for almost all of the points in the space, namely 99.9999% of them. The consequence of this is that for any given point $x$, most of $N$ lies at approximately the mean distance $\frac{\sqrt{n}}{2}$. As Kanerva states, “most of the space is nearly orthogonal to any given point”. Considering $x$ and ‘$x$ the poles, almost all the points of $N$ lie close to the equator.

4) Distribution of a circle: Another interesting property is that of a circle $C(r, x)$ in the space $2^n$. $C(r, x)$ is the set of points no more than $r$ far from $x$. The area of the circle is defined as the number of points enclosed in the region, which can be determined as the sum of the first $r + 1$ binomial coefficients:
\[ |O(r, x)| = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r} \quad (2) \]

Kanerva shows the area also follows approximately a normal distribution. That means that the circle only includes a small portion of \( N \) if \( r \) is small enough. When \( r \) gets close to \( \frac{N}{2} \), it will enclose most of \( N \), and once \( r \) surpasses \( \frac{N}{2} \), the circle will contain almost all the space.

C. Memory items as points of \( N \)

In the implementation of a sparse distributed memory, each memory item is stored as an \( n \)-bit vector. Each vector can be considered a point in \( N \).

From the analysis of the properties of \( N \), important characteristics can be delineated for such a memory:

- Distance between two points is a measure of similarity between two memory items. The closer the points, the more similar the stored patterns.
- As described in section III-B.3, almost all of the space is indifferent to every single point \( x \). Most of the space lies about \( \frac{N}{2} \) bits away from \( x \), and only a small portion of \( N \) lies close to \( x \). This seems close to what happens in the human memory, where every concept seems close (related) to a small number of other concepts, and indifferent to the vast majority of the other concepts in our memory. Nonetheless, skipping from concept to concept (point to point), it is possible to reach any concept (point) from any other one.

Another interesting property of such a memory is that of concept boundaries. The idea is best explained using an example.

Consider a million random points of \( 2^{1000} \), representing a million memory items of 1000 bits each. A circle \( C \) of radius \( r = 425 \) bits will encircle \( \binom{1000}{425} + \binom{1000}{424} + \cdots + \binom{1000}{0} \) points, or less than \( 1.18 \cdot 10^{-8} \), which is about one millionth of the space. A memory item can be considered as the centre of \( C \). The points inside \( C \) can be considered as belonging to the same concept, while the ones outside can be considered as not belonging to it.

Kanerva suggests that a radius of 400 bits, which encircles about \( 10^{-10} \) of the space (more exactly, \( 1.36 \cdot 10^{-10} \)), should provide a satisfactory classification. This means that correctly extracting 600 out of 1000 bits should provide satisfactory results, or that 0.6 probability of being correct should suffice for normal operation.

Another conclusion of the 0.6 probability point explained above is that it should suffice to know only 200 out of the 1000 bits (20% of them) to, possibly, retrieve the correct datum. The other bits could be filled randomly. The probability of getting the correct datum is, therefore, \((0.2)(1) + (0.8)(0.5) = 0.6\). Querying the memory for this datum one should be able to retrieve the correct one.

Kanerva proposes that this property can mimic our ability to recognize an object in different contexts—with only 20% of the information accurate, we’re able to retrieve the correct entity from the memory.

D. A Sparse, Distributed Memory

The first practical problem one has to solve before implementing an SDM is the vastness of the space. Indeed, \( N \) can become extremely large as the number of dimensions \( n \) increases. For a robot dealing with state vectors of 1000 bits \( N = 2^{1000} \approx 10^{301} \). As a comparing measure, the number of neurons in the nervous system is about \( 2^{36} \approx 10^{10} \), or a fraction of \( 10^{-291} \) of \( N \). Another comparison, pointed by Kanerva, is that a century contains less than \( 2^{12} \) seconds, or a portion of \( 10^{-292} \) of \( N \).

A memory with \( 2^n \) locations may, therefore, be unfeasible to implement. To overcome this problem, Kanerva proposes that the memory be sparse: it must contain only a small portion of physical addresses. Those physical addresses are called hard locations, and they are just a small subset of all the addressable space. According to the original proposal, the hard locations are given right from the start. Their addresses are distributed randomly in the space \( 2^n \), and cannot be modified during normal operation of the memory—only the contents may be updated. Following Kanerva’s notation, we’ll denote as \( N' \) this subset of \( N \), and as \( x' \) a hard (physical) location.

Kanerva proposes that \( N' \) should be just a small portion of \( N \). His example is that \( 2^{20} \) (about one million) should suffice to represent a space of 1000 dimensions. \( N' \) is, therefore, a portion of about \( 2^{980} \approx 10^{-297} \) of \( N \), which makes the memory a very sparse one. Indeed, every hard location of such a memory represents, on average, \( 2^{980} \) addresses, or one millionth of the whole space \( N \). Due to the tendency to orthogonality (section III-B.3), a circle of radius 425 contains less than one millionth of \( N \).

In conclusion, any subset of hard locations \( N' \) should appropriately represent the whole space \( N \), if one accesses the memory looking in an appropriate radius. For \( n = 1000 \), a subset of \( 2^{20} \) hard locations and an access radius of 425 bits should produce satisfactory results.

1) Writing: Since every hard location \( x' \) accounts for a small amount of \( N \), Kanerva proposes the concept of distributed storage. Writing a datum \( x \) may affect more than a single hard location. \( x \) must be written in all the hard locations within an access circle. When a write operation is performed, the contents of all the hard locations at distance \( d < r \) are updated using some algorithm. The new datum may replace the previous ones, or be summed, for example. Storage, in consequence, is distributed in an access circle.

2) Reading: Reading back datum \( x \) means taking the representative of \( x \) from \( N \). This can be done, again, pooling all the hard locations within the access circle and using some algorithm to recover the desired information.

The best method, Kanerva proposes, is to compute the average of the words in the access circle. The \( i^{th} \) bit of the average value can be computed by summing all the \( i^{th} \) bits of all the words, and thresholding with half the number of words.
E. SDM using linked lists

The SDM model is suitable to be implemented using several different techniques, including linked lists and neural networks (NN). We will be using linked lists.

One possible model for an SDM is an array of addresses which may or may not be activated by the reference address, in function of the distance between the reference address and the very memory address, an array of bit counters which store the data and columnwise adders and thresholders that calculate the output datum when reading. Figure 1 illustrates the concept.

1) Addressing: The reference address is the given address, where data is to be stored, or to be pooled. In traditional memories this address would only activate a single memory location. In an SDM, though, it will activate all the hard locations within the activation radius. To build the set of active hard locations, the Hamming distance between the reference address and every other physical address is computed. If this distance is less than the activation radius, the location is added to the set of active locations, otherwise it is ignored.

2) Hard locations: In a traditional memory every location consists of a memory element able to store a set of bits. In an SDM, every location consists of a set of counters, one per bit, which are incremented or decremented as ones or zeros are stored. Those counters are usually referred to as “bit counters”, or “data counters”. Initially, all the bit counters are set to zero, as the memory stores no data. Then, the counters are pooled in every read operation, and incremented or decremented in every write operation.

One drawback of SDMs now becomes 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 [11].

Analysing the way data is stored, we can understand, from the point of view of implementation, how rewriting the same datum many times will make it less likely to be forgotten. The counters affected by that given datum will be incremented or decremented in every single write operation, reducing the probability of suffering interference with other data at each pass. Every rewrite reinforces something the memory must not forget.

3) Data reading: Since the datum was stored in various hard locations, reading will be done by averaging the active hard locations, as explained in section III-D.2.

For every single bit, we first sum all the bits columnwise. If the sum is above a given threshold, then the corresponding bit is considered 1, otherwise it is considered 0. Zero is a good threshold if, when writing, one sums -1 to store zero, and 1 to store one.

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.

F. Characteristics

Being implemented based on the properties of the space $2^n$, a Sparse Distributed Memory constructed following the models described in section III-E should exhibit characteristics and behaviours, to a great extent, similar to human memory [5]. Namely, the following characteristics can be pointed out:

- High dimensionality — SDMs are suitable for high-dimensional systems, due to the large number of address and input lines.
- Associativity — it’s possible to associate data patterns with other patterns and/or use data vectors as address vectors and the contrary.
- Error tolerance — good tolerance to noisy data and errors, due to distributed storage and thresholding techniques.
- Natural learning and reinforcement — An SDM must be able to learn in one pass, and successive writing of the same datum will make it less likely to be forgotten.
- Natural forgetting — Data in an SDM cannot be erased. It must be “forgotten”, as time goes by and the memory becomes full.
- Graceful degradation — In case of malfunction, an SDM must behave as a typical neural network. The output doesn’t collapse all of a sudden, but becomes less accurate as more and more neurons fail. Output becomes noisy until it eventually becomes only garbage.
- Incomplete data — When about 20% of the bits accurate and the remaining 80% are set at random, it may be enough to retrieve the correct datum.
- Parallel processing — An SDM can operate in parallel, just like it is thought the human brain operates, eventually achieving high speeds of operation.
- Knowing that one knows — From incomplete data one can retrieve a pattern that should be close to the one desired. With the retrieved pattern one can query the memory again, and eventually retrieve a pattern closer to the one desired. Repeating this procedure one can either reduce the error and converge to the desired pattern, or, if the error increases between successive iterations, infer that one doesn’t know.
- Discover new relations — An SDM is a sophisticated tool to match and relate patterns. It can be used to
discover relations between previously unrelated ideas.
- Sequence storing — One can store a pattern and associate it with another pattern, and a second with a third pattern, and so on. Starting from any but the last pattern, one can retrieve the remainder of the sequence. The memory can, therefore, be used to store sequences and retrieve the continuation of a sequence of events when given a cue.

G. SDM Advantages for robotics
The SDM model exhibits some characteristics which make it look attractive for application in robotics, namely:
- According to Rao and Ballard [15], SDMs can be used in pattern (image) recognition, where they have shown tolerance to occlusion, illumination changes, scale changes and rotations in 3D.
- They’re immune to noise up to a high threshold. Using coding schemes such as n-of-m codes, their immunity is even increased [7], at the cost of reducing the addressable space.
- Robust to failure of individual locations, just like neural networks.
- They learn in a single pass.
- Unlike traditional neural networks, SDMs can be “open” and subject to analysis of individual locations.
- It’s possible to change a memory’s structure without retraining all the memory [16].
- 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.

The main drawbacks are:
- Once a datum is written, it cannot be erased, only forgotten.
- Storage is only about 0.1 bits per bit or traditional computer memory.
- If implemented in software, a lot of computer processing is required to run the memory alone.

IV. FROM KANERV A TO HAWKINS, BACK TO KAN ERVA
Hawkins’ work is undoubtly a new approach to renew AI. Nonetheless, his framework is currently not more than a single idea with little support for how things may be done. The underlying brain algorithm remains undiscovered, and there’s little evidence on how to implement the memory-prediction framework. On the other hand, Kanerva proposed a sound theory and offered the mathematical support to implement it. Our work aims, therefore, to explore Kanerva’s proposal.

V. BUILDING A SPARSE DISTRIBUTED MEMORY
Our approach to an SDM implementation is based on Linked Lists. 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 bits integer.

The composition of the input vectors is as follows:

\[ x_i = < im_{i-1}, im_i, seq \cdot d, i, timestamp, motion > \]  

where \( im_i \) is the last image. \( seq \cdot d \) 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. \( motion \) is a single character, identifying the movement the robot was performing when the image was taken. It can be ‘s’, if the robot was stopped, ‘f’ if it was moving forward, ‘b’ if it was moving backwards, ‘r’ if it was turning right or ‘l’ if it was turning left.

Our implementation of the memory is ready to work with any dimension of the vector, but images 80x64 will be used. Since every pixel is stored as an 8 bit integer, the image alone needs \( 80 \times 64 = 5120 \) bytes. The overhead information comprises 13 additional bytes, meaning the input vector is as summarised in table 1.

<table>
<thead>
<tr>
<th>Image Resolution</th>
<th>Image Bytes</th>
<th>Overhead</th>
<th>Total Bytes</th>
<th>Total Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>80x64</td>
<td>5120</td>
<td>13</td>
<td>5133</td>
<td>82024</td>
</tr>
</tbody>
</table>

\[ 2^{40960} \] is not computable with ordinary computer software. Programs such as Aribas [6] can do it: http://www.mathematik.uni-muenchen.de/~forster/sw/aribas.html
The control software developed supports a variable radius for the access circle, and also another algorithm to compute similarity between two addresses. The latter approach is inspired in Ratitch et al’s method [16]: since all the elements in the input vector can be read as 8 bit integers, the absolute sum of the arithmetic differences of those integers is used as a similarity measure.

B. Memory capacity

According to what was explained in section III-D, the memory must not grow beyond its limits. To prevent saturation, the memory must not grow beyond its limits. Each hard location must represent about the same number of locations as the number of locations in the access circle. In the case of an access radius of 19500 bits, it’s necessary to ensure:

\[
2^x = A_c \Rightarrow x = \log_2 A_c \approx 40887
\]  

(5)

Therefore, each hard location represents about \(2^{40887}\) points of the space \(2^{40960}\), or about \(1.06 \times 10^{-20}\%\) of it. To enforce this condition, the number of hard locations \(y\) should be about \(1.05 \times 10^{-22}\) of the total number of locations:

\[
\frac{y}{2^{40960}} = 2^{-40887} \Rightarrow y = 2^{73}
\]  

(6)

As equation 6 shows, the memory can have up to \(y = 2^{73}\) hard locations, which can account for several gigabytes of information. This number could even grow, if a smaller radius was used for the access circle, or shrink, if the access radius was increased. However, the addressable space seems already rich enough for normal operation under a variety of scenarios.

Another difference in our implementation, relative to Kanerva’s proposal, is that we don’t fill the virtual space placing hard locations at random in the beginning of the operation. Instead, we use Ratitch et al’s Randomised Reallocation algorithm [16]: start with an empty memory, and allocate new hard locations when there’s a new datum which cannot be stored in enough existing locations.

C. Hard locations and addressing

A significant difference between Kanerva’s proposal and our implementation is that we are not using bit counters to store data. Furber et al [7] 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. Considering that we’re interested in an SDM intended to operate in real time, this simplification greatly reduces the need for processing power. Additionally, since we’re not using bit counters and our data can only be 0 or 1, the average value of the hard locations can only be a real number in the interval \([0, 1]\). Therefore, our threshold for bitwise operation is at 0.5.

During the autonomous run, the robot will predict \(i_{m_{t-1}}\) from \(i_{m_{t}}\). Therefore, our address is \(i_{m_{t-1}}\) (and the associated overhead bits). According to Kanerva’s suggestion, \(i_{m_{t-1}}\) is enough to retrieve all the vector \(x_t\), providing it accounts for more than 20% of the bits coincident with the corresponding bits in the desired image. We could fill the query vector with \(i_{m_{t-1}}\) and all the remaining bits generated at random. This approach, however, requires unnecessary processing to generate the random bits and compute the similarity between that random part of the vector and the corresponding part of all the hard locations. Therefore, our implementation of the model 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.

Summary of the working principles for this memory:

1) Addressing is done by measuring the similarity between the input vector and the stored vectors. During bitwise operation, the Hamming distance is used. During arithmetic operation, the sum of the absolute differences between 8 bit integers is used.

2) Since we’ll be interested in using an image to retrieve the next in the sequence, ignoring the overhead bits, we can only use part of the vectors to address the space.

D. Implementation details

Since we are using the Random Reallocation algorithm, it is necessary to decide when to introduce new hard locations. The introduction should occur when a new datum to be learnt activates only a small number of hard locations, below a given threshold or even zero. In that situation, the difference between the number of active hard locations and the number of required hard locations is the number of new hard locations to be added. That threshold is a variable parameter of our memory, and we tested it with values from 1 to 20. Obviously, the higher this threshold the slower the memory operation and the faster it saturates.

Another implementation detail is the learning rate. When using the Hamming distance to compute the similarity between memory items, learning is done by setting each bit to 0 or 1, replacing the old datum. When operating with integer values, though, a different approach is necessary to update the active hard locations. We’re updating them using the following equation, where \(h_i^k\) is the \(k^{th}\) 8-bits integer of the hard location, at time \(t\), \(x^k\) the corresponding integer in the input vector \(x\) and \(\alpha\) the learning rate:

\[
h_i^t = h_i^{t-1} + \alpha \cdot (x^k - h_i^{t-1})
\]  

(7)

VI. TESTS AND PRELIMINARY RESULTS

We tested the memory with several pictures, taken at short intervals and under different lighting conditions. All the images were pre-processed before storage. This processing included conversion from JPEG to 8-bits PGM and brightness equalisation.

The memory was loaded with vectors as the one represented in equation 3. During the tests we found out that the noise level can be up to 7000 bits per image, after equalisation and without changing lighting conditions. This value is acceptable, since 7000 bits is only about 17% of the whole input vector. This number also means that for robot navigation using a view sequence we won’t store images
which differ less than 7000 bits to its predecessor. Using the arithmetic metric, noise level can be up to an absolute value of 40000, after equalisation and without changing lighting conditions.

Since we’re only using about half the vector to address the memory, and 7000 bits is just about the noise level, an access radius of about 19500 bits was used for bitwise operation. It showed to be enough to retrieve the correct images with 100% accuracy. During arithmetic operation, a radius of 80000 was used, and the results were the same. The memory was not tested close to saturation.

Preliminary tests were performed on a Pentium IV 1.8GHz. The memory was loaded with various sequences of images, taken at 10cm intervals, and it was able to learn and correctly retrieve all of them, within reasonable computational time. As an example, the processing time was measured when the memory was loaded with 100 images, stored in 3 separate hard locations each (a total of 300 hard locations). Using the bitwise method, the search of an image takes about 190ms. Using the arithmetic method, the search takes less than 80ms. It should be noted, however, that this happens because all the operations are simulated in software. In the arithmetic mode, the difference between each pair of bytes is calculated as a single difference. During bitwise operation, each pair of bytes is shifted 8 times to sum difference between the least significant bits. While in specifically designed hardware the Hamming distance would be faster than the arithmetic difference, in software this approach is not as efficient as the first one. However, from the theoretical point of view, it is still the most interesting approach, for it is the basis of the memory’s expected human-like behaviour.

VII. CONCLUSIONS AND FUTURE WORK

Considering the difficulties AI and its application to robotics are facing, we decided to look for different approaches to solve the old problem of robot navigation. Considering that research is leading to the probable conclusion that human intelligence is, to a great extent, the result of a huge and sophisticated memory, our approach was to study and implement a biologically inspired memory, to be at the core of the system. The memory model is a Sparse Distributed Memory, as proposed by P. Kanerva, and is based on the properties of high-dimensional binary spaces. The ultimate goal is to use this memory to store sequences of images, and guide the robot using those sequences. Preliminary results show that the memory works as the theory predicts, and is suitable to our needs of storing and retrieving view sequences. Its main drawback is the low storage provided, which can be as low as 0.1 bits per bit of ordinary memory in some implementations. Another drawback is the additional processing required to access the memory. That is particularly important when it’s modelled in software.

Future work includes additional tests of the capacity of the memory, and the set up of an experimental robotic platform, equipped with a camera and a computer able to process the images in real time. That should provide the necessary testbed to the next stage: design and implementation of navigation algorithms.

VIII. ACKNOWLEDGEMENTS

This work is supported in part by grant SFRH/BD/44006/2008 from Fundação para a Ciência e Tecnologia, Portugal.

REFERENCES