Vision-based Navigation Using an Associative Memory

Mateus Mendes
ESTGOH, Polytechnic Institute of Coimbra
Institute of Systems and Robotics, University of Coimbra
Portugal

1. Introduction

One of the most challenging long-term goals of Robotics is to build robots with human-like intelligence and capabilities. Although the human brain and body are by no means perfect, they are the primary model for roboticists and robot users. Therefore, it is only natural that robots of the future share many key characteristics with humans. Among these characteristics, reliance on visual information and the use of an associative memory are two of the most important.

The information is stored in our brain in sequences of snapshots that we can later retrieve full or in part, starting at any random point. A single cue suffices to remind us of a past experience, such as our last holidays. Starting from this cue we can relive the most remarkable moments of the holidays, skipping from one snapshot to another. Our work is inspired by these ideas. Our robot is a small platform that is guided solely by visual information, which is stored in a Sparse Distributed Memory (SDM).

The SDM is a kind of associative memory proposed in the 1980s by Kanerva (1988). The underlying idea is the mapping of a huge binary memory onto a smaller set of physical locations, so-called *hard locations*. Every datum is stored distributed by a set of hard locations, and retrieved by *averaging* those locations. Kanerva proves that such a memory, for high dimensional binary vectors, exhibits properties similar to the human memory, such as ability to work with sequences, tolerance to incomplete and noisy data, and learning and forgetting in a *natural* way.

We used a SDM to navigate a robot, in order to test some of the theories in practice and assess the performance of the system. Navigation is based on images, and has two modes: one learning mode, in which the robot is manually guided and captures images to store for future reference; and an autonomous mode, in which it uses its previous knowledge to navigate autonomously, following any sequence previously learnt, either to the end or until it gets lost or interrupted.

We soon came to the conclusion that the way information is encoded into the memory influences the performance of the system. The SDM is prepared to work with random data, but robot sensorial information is hardly well distributed random data. Thus, we implemented four variations of the model, that deal with four different encoding methods. The performance of those variations was then assessed and compared.

Section 2 briefly describes some theories of what intelligence is. Section 3 describes the SDM. Section 4 presents an overview of various robot navigation techniques. In sections 5 and 6 the hardware and software implementation are described. In section 7 we describe the encoding problem and how it can be solved. Finally, in section 8 we describe some tests we performed and the results obtained, before drawing some conclusions in section 9.

2. Human and machine Intelligence

The biggest problem one faces when researching towards building *intelligent machines* is that of understanding what is intelligence. There are essentially three problems researchers have to face: 1) What is intelligence; 2) How can it be tested or measured; and 3) How can it be artificially simulated. We're not deeply concerned about these points in this study, but the very definition of intelligence deserves some attention, for it is the basis of this work—after all, the goal is to build a system able to perform intelligent vision-based navigation.

2.1 Definitions of intelligence

Until very recently, the most solid ground on this subject was a series of sparse and informal writings from psychologists and researchers from related areas—and though there seems to be a fairly large common ground, the boundaries of the concept are still very cloudy and roughly defined.

Moreover, it is in general accepted that there are several different "intelligences", responsible for several different abilities, such as linguistic, musical, logical-mathematical, spacial and other abilities. However, in many cases individuals' performance levels in all these different fields are strongly correlated. Spearman (1927) calls this positive correlation the *g*-factor. The *g*-factor shall, therefore, be a general measure of intelligence. The other intelligences are mostly specialisations of the general one, in function of the experience of the individual.

2.1.1 Gottfredson definition

Gottfredson (1997)[1] published an interesting and fairly complete review of the mainstream opinion in the field . Gottfredson wrote a summary of her personal definition of intelligence, and submitted it to half a dozen "leaders in the field" for review. The document was improved and then submitted to 131 experts in the field, who were then invited to endorse it and/or comment on it. 100 experts responded: 52 endorsed the document; 48 didn't endorse it for various reasons. Of those who didn't, only 7 stated that it did not represent the mainstream opinion about intelligence. Therefore, it is reasonable to assume that a representative number of experts agree with this very definition of intelligence:

> Intelligence is a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings—"catching on,", "making sense" of things, or "figuring out" what to do.

It is our understanding that Gottfredson emphasises some key aspects: **problem solving**, **learning** and **understanding**. It should be noted that there's little consideration with the performance of the intelligent agent. At most, that is part of the "problem solving" assessment.

[1] The reference Gottfredson (1997) states the article was first published in the *Wall Street Journal*, December 13, 1994.

On the contrary, this definition strongly depends on the ability to "understand". However, there's no definition of what is "understanding", meaning that this definition of intelligence is of little use for engineers in the task of building intelligent machines.

2.1.2 Legg's formal definition

Legg & Hutter (2007) also present a thorough compilation of interesting definitions, both from psychologists and AI researchers. And they end up with a shorter and very pragmatic definition:

> Intelligence measures an agent's ability to achieve goals in a wide range of environments.

This very definition has the merit of being much shorter and clearer from the point of view of an engineer, as it is very pragmatic. Legg starts from this informal definition towards a more formal one, and proposes what is probably one of the first formal definitions of intelligence.
According to Legg, an intelligent agent is the one who is able to perform *actions* that change the surrounding *environment* in which he exists, assess the *rewards* he receives and thus *learn* how to behave and profit from his actions. It must incorporate, therefore, some kind of reinforcement learning.
In a formal sense, the following variables and concepts can be defined:

o	observation of the environment
a	agent's action
r	reward received from the environment
π	an agent
μ	an environment
E	the space of all environments

The agent π is defined as a probability measure or its current action, given its complete history: $\pi(a_k|o_1r_1...o_{k-1}r_{k-1}), \forall \quad k \in \mathbf{N}$.
The environment μ is defined as a probability function of its history: $\mu(o_kr_k|o_1r_1a_1....o_{k-1})$
Legg also imposes the condition that the total reward is bounded to 1, to make the sum of all the rewards received in all the environments finite:

$$V_\mu^\pi := E\sum_{i=1}^{\infty} r_i \leq 1 \tag{1}$$

One important point to consider when evaluating the performance of the agent is also the complexity of the environment μ. On this point, Legg considers the Kolmogorov complexity, or the length of the shortest program that computes μ:

$$K(\mu) = \min_p\{l(p) : \mathcal{U}(p) = \mu\} \tag{2}$$

where $\mathcal{U}$ is the universal Turing Machine.
Additionally, each environment, in this case, is described by a string of binary values. As each binary value has two possible states, it must reduce the probability of the environment by $1/2$. Therefore, according to Legg, the probability of each environment must be well described by the *algorithmic probability distribution* over the space of environments: $2^{-K(\mu)}$.
From these assumptions and definitions, Legg proposes the following measure for the universal intelligence of an agent μ:

$$\Upsilon(\pi) = \sum_{\mu \in E} 2^{-K(\mu)} V_{\mu}^{\pi} \tag{3}$$

The intelligence Υ of an agent π is, therefore, a measure of the sum of the rewards it is able to receive in all the environments—a formal definition that is according to the informal ones described before. Unfortunately, the interest of this definition up to this point is most from a theoretical point of view, as this equation is not computable. It is, nonetheless, an interesting approach to formalise intelligence. And it is still interesting from a practical point of view, as a general demonstration that intelligent agents need to be versatile to perform well in a wide range of environments, as well as profit from past experience.

2.1.3 Discussion

So far, we have presented two mainstream definitions of Intelligence:

1. Gottfredson's definition of intelligence as an ability to learn, understand and solve problems;
2. Legg's formal definition of intelligence as a measure of success in an environment.

These definitions are not incompatible, but if the first is to be accepted as the standard, we need an additional definition of what is understanding. Searle (1980) proposed an interesting thought experiment which shows that performance is different from understanding. Imagine Searle in a room where he is asked some questions in Chinese. Searle knows nothing of Chinese, but he has a book with all the possible questions and the correct answers, all in Chinese. For every question, Searle searches the book and sends the correct answer. Therefore, Searle gives the correct answer 100 % of the times, without even knowing a single word of the language he's manipulating.
Anyway, the definitions seem to agree that successfully solving problems in a variety of environments is key to intelligence. Dobrev (2005) goes even further, proposing that an agent that correctly solves 70 % of the problems (i.e., takes the correct decision 7 out of every 10 times), should be considered an intelligent agent.

2.2 The brain as a memory system

From above, intelligence is solving problems and learning. But how does the brain do it? That is currently an active and open area of research. However, current evidence seems to point that the brain works more as a sophisticated memory than a high speed processing unit.

2.2.1 On the brain

The study of the brain functions *one by one* is a very complex task. There are too many brain functions, too many brain regions and too many connections between them. Additionally, although there're noticeable physical differences between brain regions, those differences are only but small. Based on these observations, V. Mountcastle (1978) proposed that *all* the brain might be performing basically the same algorithm, the result being different only depending 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. Some people who have special brain areas damaged can have other parts of the brain processing information which is usually processed in the damaged area in healthy people. Even more convincing, neuroscientists have surgically rewired the brains of newborn ferrets, so that their eyes could send signals to the areas of cortex that should process auditory information. In result, the ferrets developed visual pathways in the auditory portions of their brains (Hawkins & Blakeslee (2004)).
The brain is 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 *misery* is yet more intriguing as we find out that the brain performs incredibly *complicated* tasks at an incredibly fast speed. It is known neurons take about 5 ms to fire and reset. This means that our brain operates at about 200 Hz—a frequency fairly 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. Harnish (2002) proposes the 100 steps thought experiment to prove this. The brain takes about 1/10th of a second to perform tasks such as language understanding or visual recognition. Considering that neurons take about 1/1000 of a second to send a signal, this means that, on average, those tasks cannot take more than 100 serial steps. 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.

2.2.2 Intelligence as memory

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. An example from Hawkins & Blakeslee (2004) is the apparently simple task of catching a ball moving at some speed. The brain needs to process visual information to identify the ball, its speed and direction, and compute the motor information needed to move all the muscles which have to be stimulated in order to catch the ball. And more intriguing, the brain has to be repeating all those steps several times in a short time interval for better accuracy, while at the same time controlling basic impulses such as breathing and keeping a stable stance and equilibrium. To build a robot able to perform this apparently simple task is a nightmare, if not at all impossible, no matter how many processors can be used. The most difficult part of the problem is that motor information cannot be processed while sensory information is not available. No matter how many processors are used, there is always a number of steps which cannot be performed in parallel. 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. It doesn't compute solutions, but retrieves them based on analogies with learnt experiences from past situations. That also explains why practice and experience lead us closer to perfection—our *database* of cases, problems and solutions is

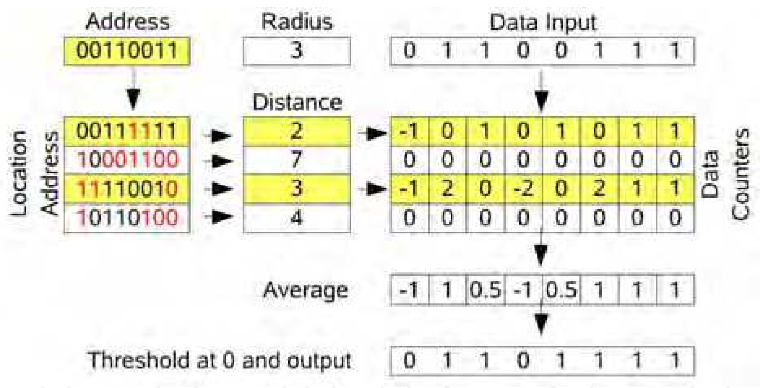

Fig. 1. One model of a SDM.

enriched, allowing us to retrieve better solutions to problems similar to the ones we've already captured.

Even before Hawkins' memory model, other researchers proposed models which somehow try to mimic human characteristics. Willshaw et al. (1969) and Hopfield (1982) propose two neural network models which are very interesting. A more promising proposal, however, was that of Pentti Kanerva. Kanerva (1988) proposes a complete model for the system, and not just a network model.

3. The Sparse Distributed Memory

Back in the 1980s, Pentti Kanerva advocated the same principle stated above: intelligence is probably the result of using a sophisticated memory and a little processing. Based on this assumption, Kanerva proposed the Sparse Distributed Memory model, a kind of associative memory based on the properties of high-dimensional binary spaces.

Kanerva's proposal is based on four basic ideas: the space 2^n, for $100 < n < 10^5$, exhibits properties which are similar to our intuitive notions of relationships between the concepts; neurons with n inputs can be used as address decoders of a random-access memory; unifying principle: data stored in the memory can be used as addresses to the same memory; and time can be traced in the memory as a function of where the data are stored. Kanerva presents thorough demonstrations of how those properties are guaranteed by the SDM. Therefore, we will only focus on the implementation details. Figure 1 shows a model of a SDM. The main modules are an array of addresses, an array of bit counters, a third module that computes the average of the bits of the active addresses, and a thresholder.

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

Writing is done by incrementing or decrementing the bit counters at the selected addresses. Data are stored in arrays of counters, one counter for every bit of every location. To store 0 at a given position, the corresponding counter is decremented. To store 1, it is incremented. The counters may, therefore, store either a positive or a negative value.

Reading is done by summing the values of all the counters columnwise and thresholding at a predefined value. If the value of the sum is below the threshold, the bit is zero, otherwise it is one. For a memory where the counters are incremented or decremented one by one, 0 is a good threshold value.
Initially, all the bit counters must be set to zero, for the memory stores no data. The bits of the address locations should be set randomly, so that the addresses would be uniformely distributed in the addressing space.
One drawback of SDMs becomes now clear: while in traditional memories we only need one bit per bit, in a 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 Keeler (1988).
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.

4. Robot navigation and mapping

To successfully navigate a robot, it must have some basic knowledge of the environment, or accurate exploring capacities. This means that the problems of navigation and mapping are closely related. Several approaches have been tried to overcome these problems, but they are still subject to heavy research. It is accepted (see e.g. Kuipers & Levitt (1988)) that robust mapping and navigation means that performance must be excellent when resources are plentiful and degradation graceful when resources are limited.
View based methods, most of the times, rely on the use of sequences of images, which confer the robot the ability to follow learnt paths. Topological maps may or may not be built. This is according to the human behaviour, for it is known that humans rely on sequences of images to navigate, and use higher level maps only for long distances or unknown areas.
However, despite the importance of vision for humans, view based methods are not among the most popular between researchers. One reason for this is that vision usually requires huge processing power. Other approaches include the use of Voronoi Diagrams and Potential Fields methods. Navigating through the use of View Sequences is not as common as other major approaches, but it's becoming increasingly popular as good quality cameras and fast processors become cheaper.

4.1 Some popular mapping and navigation methods

One popular approach is indeed very simplistic: the occupancy grid (OG). The grid is simply a matrix, where each element means the presence or absence of an obstacle. The robot must be able to position itself in the grid by scanning its surroundings and/or knowing its past history. Then it can move from one grid cell to another empty cell, updating the map accordingly. This method is often combined with a Potential Field algorithm. The robot's goal is to reach the centre of the potential field, to where it is being attracted. Every pixel in the matrix contains a number, representing the power of the potential field. The higher the potential, the closer the robot is to its goal. The robot must then try to find the path by following the positive gradient of the potential values. The disadvantages of the OG are obvious: huge memory requirements, and difficulty in scaling to large environments.
Another navigation method is the Voronoi Diagram (VD). The VD is a geometric structure which represents distance information of a set of points or objects. Each point in the VD is

equidistant to two or more points. While moving, the robot must sense the objects and walls and incrementally create the diagram. Its goal is to always be in the centre. It must then move between objects and walls, thus avoiding collisions.
Topological maps overcome many of the drawbacks of the grid-based methods. They consist of more sophisticated representations, which neglect details of the environment. The robot builds a graph of the routes available for the robot to travel. Then, algorithms such as Dijkstra (1959) or A* can be used to compute the possible paths and possibly choose the best alternative. Using these models, there is no need to record information of all the path that the robot follows. Instead, the robot only stores information about the points at which decisions are made.
Kuipers & Levitt (1988) proposes a semantic hierarchy of the descriptions of the interaction with the environment, consisting of four hierarchical levels:

1. *Sensorimotor*—This is the lower level of the hierarchy, represented by the sensor input and actuator output data.
2. *Procedural*—These are procedures the robot learns to accomplish particular instances of place finding or route following tasks. These procedures are acquired in terms of sensorimotor primitives, such as control strategies to cross a boundary defined by two landmarks. The basic element of procedural behaviours are represented in a production-like schema, $< goal, situation, action, result >$. The interpretation of the production is "When attempting to reach goal $goal$, if the current view is $situation$, perform action $action$ and expect the result to be $result$. At this level, the robot is able to perform basic navigation tasks and path planning.
3. *Topological*—While at the procedural level the environment is described in terms of the egocentric sensorimotor experience of the robot, at the topological level the world is described in terms of relations between other entities. A description of the environment is possible in terms of fixed entities, such as places, paths and landmarks, linked by topological relations, such as connectivity, containment and order. At this level, it is possible to perform navigation tasks and route planning more efficiently than at the procedural level. However, there is no information on the cost of alternative paths. In topological terms, it is only possible to describe actions such as "turn" or "travel", but there is no associated description of the magnitude required for the action.
4. *Metric*—This level adds to the topological level metric information, such as distance and angle information. At this level the robot has all the information it needs to plan routes and choose the quickest or shortest alternative path. It is possible to compute accurately the length of a path, as well as the number and magnitude of the turns that the robot has to perform.

Kuipers proposes three models which are based on the idea of a four levels semantic hierarchy: the Tour model, the Qualnav simulator and the NX Robot. All these models have been tested in software simulation Kuipers & Levitt (1988), to validate the four levels hierarchy described. The Tour model is a computational model of a robot more suitable for environments that have approximate network-like structures, such as urban environments or the interior of large buildings with corridors and walls. The Tour model robot is expected to explore the world and find places and paths that can be followed, in order to build a topological map with metric information of the environment. At the sensorimotor level, Tour has available a sequence of views and actions. At the procedural level, it has procedures built from the sensorimotor schemas. This means that Tour is expected to grab a sequence of views $V_0, ...V_n$, and for each

view V_j, it must perform action A_j. At the topological level, Tour builds place-path networks, boundary relations, regions and skeletal networks. At the metric level, Tour keeps information about local geometry of places and paths. This confers it the ability to perform actions where the action can be $Travel(\delta)$ to move forward or backward, and $Turn(\alpha)$ to change direction.
Kuipers simulated robots obtain the topological models without great difficulty, in the simulation environment. In a virtual environment, sensorimotor readings can be assumed to be correct. Kuipers used these readings to feed the topological level of the semantic hierarchy and build the topological representation of the environment. In the real world, however, sensorial readings are subject to a multitude of interferences that make the task of building a topological map much more difficult than in a controlled environment. Variable amounts of noise from the sensors, scenario changes and other phenomena interfere with the performance of the system and make the mapping task a very challenging one. Many authors implemented topological navigation, based on Kuipers' simulated models, although the practical problems have to be addressed to achieve a robust navigation model.

4.2 View based navigation

Many approaches consist of teaching the robot some sequence of images and then make it follow the same path by capturing an image, retrieve the closest image from its memory and execute the same motion that is associated with it.
View sequence navigation is extremely attractive, for it is very simple to implement, and a single sensor, i.e., the camera, provides a lot of information. Humans are able to navigate quite well based only on visual information. But robots are usually guided with many other sensors, such as sonars, infrared or other distance sensors. Those sensors provide additional information that has to be processed in real time, and the robots usually have to develop internal models of the world based on this sensorial information. "Teach and follow", is a much simpler and biologically inspired approach for basic navigation purpose. The disadvantage is that images have to be grabbed and stored at regular intervals during learning, and matched against a large database of images during the autonomous run. Good resolution images take a lot of storage space and processing time. To overcome this problem, some authors choose to extract some features from the images, in order to save processing needs in real time.
One possible approach for navigation based on a view sequence is that of using landmarks. Those landmarks may be artificially placed on the scene (a barcode or data matrix, for example), or selected from the actual images or the real scene in runtime. Selecting a few landmarks which are easy to identify, the navigation task is greatly facilitated. Rasmussen & Hager (1996) follow such an approach. They store panoramic images at regular intervals during learning, and select salient *markers* from them. *Markers* are characteristics considered invariant over time, such as colour, shape or texture. By sticking to these *markers*, the system saves considerable processing power, for there is no need to process the whole image to detect the best match.

4.2.1 Matsumoto's approach

Matsumoto et al. (1999; 2000) propose a vision-based approach for robot navigation, based on the concept of a "view-sequence" and a look-up table of controlling commands. Their approach requires a learning stage, during which the robot must be guided. While being guided, the robot memorises a sequence of views automatically. While autonomously running, the robot performs automatic localisation and obstacle detection, taking action in real-time.

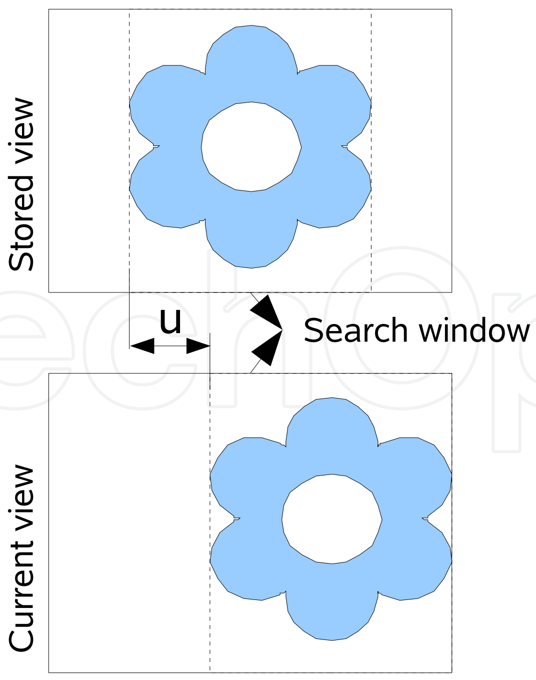

Fig. 2. Matching of two images, using a search window.

Localisation is calculated based on the similarity of two views: one stored during the learning stage and another grabbed in real-time. The robot tries to find matching areas between those two images, and calculates the distance between them in order to infer *how far* it is from the correct path. That distance is then used to extract stored motor controls from a table, thus guiding the robot through the same path it has been taught before.
To calculate the drift in each moment, Matsumoto uses a block matching process, as Figure 2 illustrates. A search window taken from the memorised image is matched against an equivalent window in the current view, calculating the horizontal displacement that results in the smallest error e. Considering two images I_1 and I_2, Matsumoto defines the matching error between them as:

$$e(u) = \sum_{x=s}^{w-s} \sum_{y=0}^{h} |I_1(x,y) - I_2(x+u,y)| \tag{4}$$

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

$$e = min(e(u)), -s \leq u < s \tag{5}$$

If the robot is following a previously learnt image sequence, its view in each moment must match some stored image. If there is a horizontal drift, then e will be minimum for some horizontal displacement $u \neq 0$. Equation 4 is just the sum or the errors between the pixels in the image, for some value u of the horizontal search range. Equation 5 determines the horizontal displacement u that gives the minimum error e.
Matsumoto uses 32×32 images, a search window 16×32 and a horizontal search range of 8 pixels, so that $-8 \leq u < 8$. The original images, captured by the robot, are 480×480, but

the author claims that there is no significant loss of essential information when the images are subsampled up to the 32×32 thumbnails that are being used.
In a later stage of the same work, Matsumoto et al. (2003) equipped the robot to use omniview images. Omniview images represent all the surroundings of the robot. This is achieved through the use of a spherical mirror, that is placed above the robot and reflects the scenario 360° wide onto a conventional camera. Omniviews are neither easy to recognise for humans, who have a narrow field of view, nor to process using conventional image processing algorithms. However, they allow the robots to overcome the limitation of the narrow field of view of traditional cameras. One camera can be used to sense all the directions at the same time. Forward and backward motion and collision avoidance may be planned using the same single sensor. To facilitate image processing, the omniview image can then be transformed into a cylindrical image that represents the surroundings of the robot. This cylindrical image can then be processed as if it was a common image, although it is necessary to pay attention to the fact that it represents a field of view of 360°. Its left and right borders are actually images of the back of the robot, while the middle of the image represents its front.
Matsumoto extended the navigation algorithms and sequence representations. Junctions, such as corridor intersections, were detected, based on the optical flow differences. Assuming the optical flow is just caused by the motion of the robot, closer objects generate larger optical flows in the image. This simple principle makes it possible to detect junctions and construct a topological map of the building. The images were stored with additional information of neighbouring images, so that it was possible to follow a corridor and, at the appropriate junction, skip to another. Route planning was done using the Dijskstra (1959) algorithm, so that the shortest path was followed. Matsumoto's approach, however, has the disadvantages of being too computationally intensive and suitable only for indoors navigation.

4.2.2 Ishiguro's approach

Ishiguro & Tsuji (1996) used an approach slightly different from Matsumoto. Ishiguro uses omni-directional views, but in order to speed up processing, the images are replaced by their Fourier transforms. The omni-directional views are periodic along the azimuth axis. Thus, the Fourier transform is applied to each row of an image, and the image is then replaced by the set of magnitude and phase Fourier coefficients. This technique reduces both the memory and the computational requirements.
In order model the geometry of the environment, Ishiguro defines a set of observation points. From each observation point, the robot captures a view I_i of the environment. Then it compares this view with all the views I_j that it has learnt before, computing a measure of similarity, using the formula:

$$Sim(I_i, I_j) = \sum_{y=0}^{l-1} \sum_{k=0}^{m-1} |F_{iy}(k) - F_{jy}(k)| \tag{6}$$

where $F_{iy}(k)$ and $F_{jy}(k)$ are the Fourier coefficients of the frequency k of the row y of images I_i and I_j. The similarity is interpreted as a measure of the distance—the larger the similarity measure, the farther apart the images must be. The relation holds up to a threshold value. Above this threshold, the similarity measure is meaningless, meaning that the images are probably unrelated at all.
Based on these ideas, Ishiguro proposes a navigation strategy. To start, the agent explores the unknown environment, capturing and storing images at a number of reference points. These images are then processed and organised according to the similarities computed, leading to

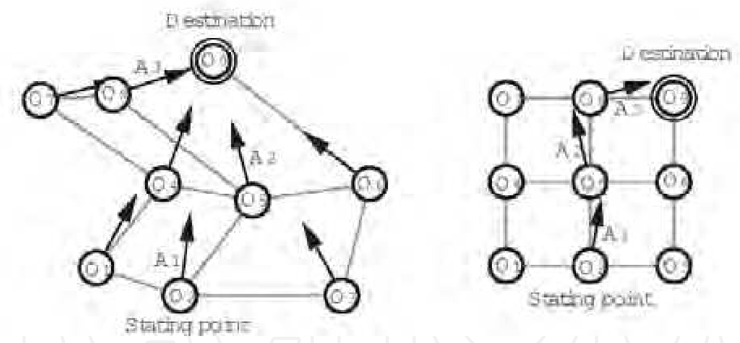

Fig. 3. Geometry of a recovered and real environment. Source: Ishiguro & Tsuji (1996).

a geometrical representation that must describe the environment accurately in terms of paths that can be followed, as illustrated in Figure 3. The robot is then able to navigate from one reference point to another, choosing the shortest path according to the perceived distance between images.

4.2.3 The T-Net

One related work is the Target Network (T-Net), implemented by Ishiguro et al. (1995). A T-Net is similar to the topological mapping model, but it relies on some key "feature points", paths and intersections between these paths, as shown in Figure 4. Ishiguro uses a robot with a panoramic vision system. The robot travels along the paths by tracking the feature points of those paths, which are their starting and ending points. The angular relations between the paths are also calculated, based on the panoramic views. As the robot moves, it picks candidates for intersections between paths and explores the environment to verify if the intersections are possible. If the intersections are possible, they are stored, as passage points between different paths. The T-Net is, therefore, constituted by the set of feature points, paths and intersections, thus being a solid basis for quick and efficient route planning and navigation.
Similarity between the images of the feature points is computed by a special algorithm, known as the Sequential Similarity Detection Algorithm (SSDA). However, to make tracking of the features points possible, it is necessary that the images input to the SSDA algorithm do not present significant changes in size. Ishiguro solves this problem using cameras with zoom. The magnification factor is decreased as the robot approaches the feature point and increased as it moves away from it.
However robust, navigation based on a T-Net is clearly a messy and expensive approach, for it requires expensive hardware (a mobile robot with at least two cameras, although Ishiguro reports to have used four) and a lot of computer power to process the images and compute the intersections between paths in real time. Besides, it is not clear how the feature points have to be defined.

4.2.4 Other view-based approaches

Jones et al. (1997) also attempted view-based navigation, in an approach very similar to Matsumoto's idea described above. The images used in his system are not panoramic, but low resolution wide angle images. Correlation between the images is computed using a method of Zero mean Normalised Cross Correlation (ZNCC). The trajectory of the robot is adjusted in order to maximise the correlation between the images grabbed by the robot and the ones

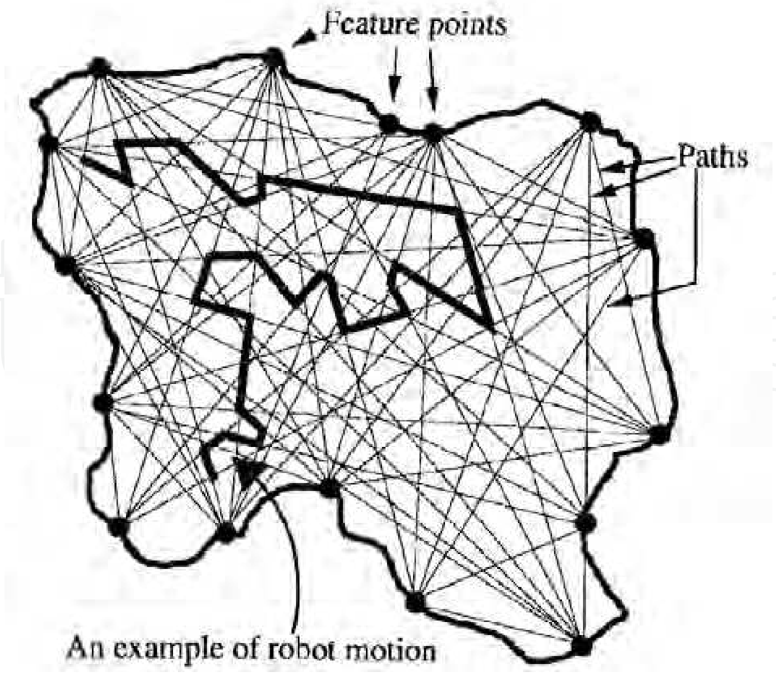

Fig. 4. Feature points and paths of a T-Net. Source: Ishiguro et al. (1995).

stored in the database. Odometric information is also used to correct the estimated position of the robot.
Winters & Santos-Victor (1999) use yet another different approach. The authors also propose a view based navigation method, based on omni-directional images. The images are compressed using Principal Component Analysis (PCA), thus producing a low dimensional eigenspace. The eigenvalues of the images are projected into the eigenspace and connected by a linear interpolation model. This forms a manifold in the eigenspace, which represents the path that the robot must follow. The manifold is a kind of topological map of the environment. The robot is then guided with the goal of minimising the angle between the path followed and the desired path.
Winter and Santos-Victor approach is in many aspects similar to Franz et al. (1998)'s approach, although the latter do not use compressed images. The manifold is constituted by the images themselves. To save computational requirements, only images that are relevant to the navigation task are stored, which means that only "snapshot images" taken at decision points are retained. The authors also consider the problem of similar views, which is solved using the past history of the robot, assuming that the robot moves continuously in the navigation space.
Table 1 summarises the approaches that have been described above, and Table 2 compares the described mapping methods.

4.3 Previous work using SDMs

4.3.1 Rao and Fuentes Goal-Directed navigation

Rao & Fuentes (1998) were probably the first to use a SDM in the robotics domain. They used a SDM to store perception (sensorial) information, which was later used to navigate the robot. Rao and Fuentes' model was a hierarchical system, inspired by Brooks (1986) subsumption architecture, in which the lower levels were responsible for lower level tasks, such as collision

Approach	Images	Landmarks	Compression	Similarity mode
Matsumoto	omniview	intersections	–	block match
Iconic memory	omniview	user defined points	Fourier trans.	Fourier coef.
T-net	4 cameras	beg. & end of paths	–	SSDA
Stephen Jones	wide angle	–	–	ZNCC
Winters	omniview	–	PCA	eigenvalues
Franz	omniview	intersections	–	neural net

Table 1. Summary of the analysed view-based navigation approaches.

Method	Sensor requirements	Memory requirements	Processing requirements	Scalability
Occupancy Grids	various	huge	high	low
Voronoi Diagrams	various	large	large	medium
Topological Maps	various	low	low	good
View Sequence	camera	large	large	good
Landmark based	landmark	low	low	low

Table 2. Summary of the analysed mapping approaches.

detection and obstacle avoidance. At this level the behaviour of the system was essentially reactive, and a simple hill-climbing learning algorithm was implemented.
The SDM used was implemented as a Neural Network, as shown in Figure 5. This Neural Network is used at the upper level, to provide goal-directed navigation. It was trained by the user, who manually guided the robot through the desired path, during a learning stage. While in the learning mode, the sensorial information from optical sensors (Perception) was recorded and stored into the SDM, along with the motor control vectors. During the autonomous run, similar perceptions (readings from the optical sensors), are expected to retrieve the correct actions (motor commands), making the system follow the same path by using this information, thus repeating the same trajectory. The authors report simulation results only, and they mention nothing about drift corrections or other error control strategies.
The SDM used in this system was also a variant of the original Kanerva's proposal. The initial addresses are picked at random, as Kanerva suggests, and represent the first layer of the Neural Network. This means that before operation, the weights w of the neural connections are assigned random values. But, contrary to the original idea, in this implementation the values

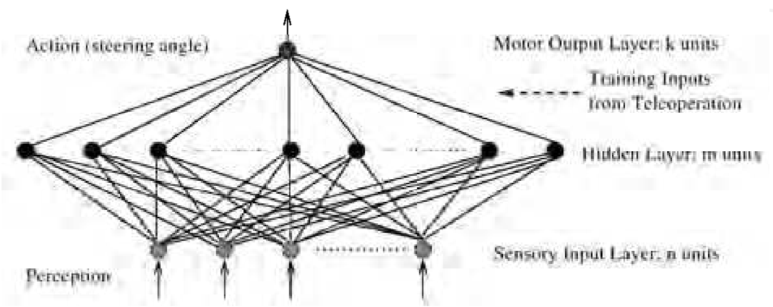
Action (steering angle)
Motor Output Layer: k units
Training Inputs from Teleoperation
Hidden Layer: m units
Sensory Input Layer: n units
Perception

Fig. 5. Rao and Fuentes implementation of a SDM.

of these neurons can be changed, adjusting the addresses towards the most active regions of the address space. Rao and Fuentes call this implementation a "Self-Organizing SDM".
Learning in this memory, both for the address and the data spaces, occurs using a soft competitive learning rule. For each perception (input) vector p, at time t, the Euclidian distance between p and the corresponding neural weight w_j is computed: $d_j = \| w_j^t - p \|$. Then, the weight vector is adjusted as follows:

$$w_j^{t+1} \leftarrow w_j^t + g_j(t) \cdot P_t(d_j) \cdot (p - w_j^t) \tag{7}$$

P is the equivalent of the probability of the prototype vector w_j winning the current competition for perception p, which, for m neurons in a layer, is computed as follows:

$$P_t(d_j) = \frac{e^{\frac{-d_j^2}{\lambda_j(t)}}}{\sum_{i=1}^{m} e^{\frac{-d_j^2}{\lambda_j(t)}}} \tag{8}$$

$\lambda_j(t)$ is the equivalent of the "temperature" of the system, thus representing its susceptibility to change. The temperature is higher at the beginning and lowers as the system stabilises over time. $g_j(t)$ is also a stabiliser parameter, given by:

$$g_j(t) = \frac{1}{n_j(t)}, where \quad n_j(t+1) = n_j(t) + P_t(d_j) \tag{9}$$

4.3.2 Watanabe et al AGV

Watanabe et al. (2001) also applied a SDM in robotics, though just for the small task of scene recognition in a factory environment. The goal is to build intelligent vehicles that can move around autonomously—Autonomously Guided Vehicles (AGV)—in industrial environments. Those vehicles are usually required to run the same routes in a repetitive manner, transporting tools or other materials, from one point to another of the production line. But under normal circumstances it is required that they must be able to identify small changes in the environment, such as the presense of other AGVs, people, tools or other entities that may get into the route of the AGV. Even in the presence of those obstacles, they must avoid collision with them and successfully complete their goals.
Watanabe et al. use a Q-learning technique to teach the robots the paths they must perform, as well as learn the behaviour to negotiate with other AGVs in the environment. Scene recognition is performed based on the use of a SDM. The SDM is, therefore, used as a mere pattern recognition tool.
The factory plan is input to the AGV in the form of an occupancy grid. When moving, for each grid position, the system computes in real time the correct motion to execute, in function of its goal. But if the scene its sensorial readings at its current position differ from the expected sensorial readings, which are stored in the SDM, then an obstacle is probably in the way of the AGV. In this case, the system has to replan its trajectory and find an alternative way to bypass the obstacle and still make its way to the target point.
To the best of our knowledge, the authors report simulation results only, for a virtual AGV that can sense obstacles around itself, capture the scenes to store into the SDM, and knows every time its current location, as well as its goal location. The SDM used was implemented as a Neural Network.

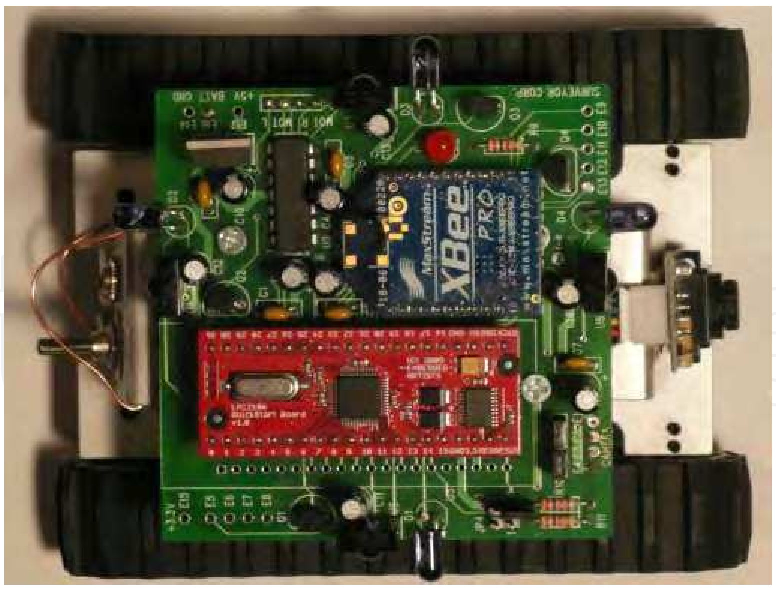
Fig. 6. Robot used.

5. Experimental Platform

We used a SDM to navigate a robot, in order to test some of the theories in practice and assess the performance of the system. The robot used was a Surveyor[2] SRV-1, a small robot with tank-style treads and differential drive via two precision DC gearmotors (see Figure 6). Among other features, it has a built in digital video camera with 640x480 resolution, four infrared sensors and a Zigbee 802.15.4 radio communication module. This robot was controlled in real time from a laptop with an Intel 1.8 GHz Pentium IV processor and 1 Gb RAM.
The overall software architecture is as shown in Figure 7. It contains three basic modules:

1. The SDM, where the information is stored.
2. The Focus (following Kanerva's terminology), where the navigation algorithms are run.
3. A low level layer, responsible for interfacing the hardware and some tasks such as motor control, collision avoidance and image equalisation.

Navigation is based on vision, and has two modes: one learning mode, in which the robot is manually guided and captures images to store for future reference; and an autonomous mode, in which it uses its previous knowledge to navigate autonomously, following any sequence previously learnt, either to the end or until it gets lost (when it doesn't recognise the current view).
Level 1 is working as a finite state machine with two states: one where it executes all the orders from above, and the other where it is blocked awaiting orders. The robot is normally operating in the first state. When an image is out of focus or the infrared proximity detectors detect an obstacle close to the robot, Lizard stops an autonomous run and waits until further order is received from the user. In historical terms, Level 1 can be considered as inspired by the first level of competence of the Brooks' subsumption architecture (Brooks (1986)). In biological terms, it can be considered as inspired by the "primitive brain", which controls basic body functions such as breathing and other instinctive behaviours. This layer is also responsible for

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

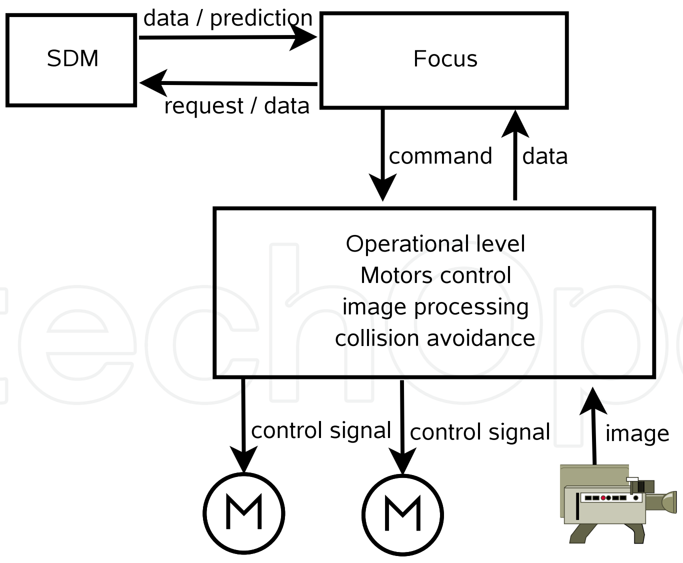

Fig. 7. Architecture of the implemented software.

Image resolution	Image bytes	Overhead	Total bytes	Total bits
80×64	5120	13	5133	41064

Table 3. Summary of the dimensions of the input vector.

converting the images to 8 bit grayscale and equalise the brightness to improve quality and comparability.
Navigation using a view sequence is based on the proposal of Matsumoto et al. (2000). This approach requires a learning stage, during which the robot must be manually guided. While being guided, the robot memorises a sequence of views automatically. While autonomously running, the robot performs automatic image based localisation and obstacle detection, taking action in real-time.
Localisation is accomplished with basis on the similarity of two views: one stored during the learning stage and another grabbed in real-time. During autonomous navigation, the robot tries to find matching areas between those two images, and calculates the horizontal distance between them in order to infer *how far* it is from the correct path. That distance is then used to correct eventual drifts, until the error is smaller than 5 pixels or cannot be made smaller after 5 iterations.

6. Our implementations of the SDM

In our implementation, 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 graylevel value of an image pixel or an 8-bit integer.
The composition of the input vectors is as summarised in Table 3 and Equation 10:

$$x_i =< im_i, seq\_id, i, timestamp, motion > \qquad (10)$$

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

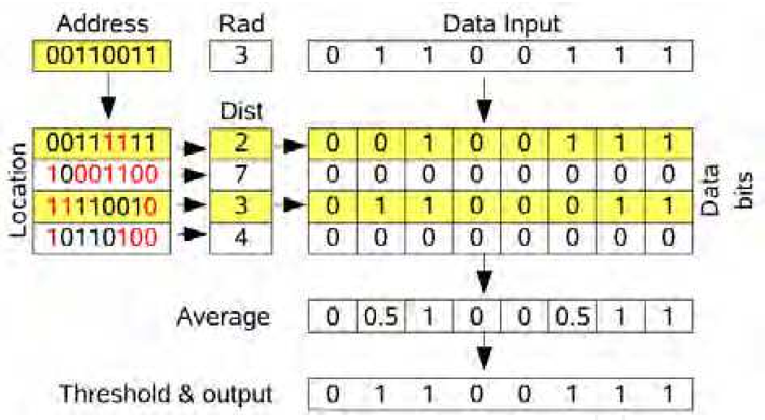

Fig. 8. Bitwise SDM, not using bit counters.

image in the sequence. *timestamp* is a 4-byte integer, storing Unix timestamp. It is read from the operating system, but not being used so far for navigation purposes. *motion* is a single character, identifying the type of movement the robot was performing when the image was grabbed.
We are using Portable Gray Map (PGM) images, in which each pixel is an 8-bit integer, representing a gray level between 0 (black) and 255 (white). Resolution is 80x64, implying that the image alone needs $80 \times 64 = 5120$ bytes The overhead information comprises 13 additional bytes, meaning the input vector contains 5133 bytes.
The memory is used to store vectors as explained, but addressing is done using just the part of the vector corresponding to one image. During the autonomous run, the robot will predict im_i from im_{i-1}. Therefore, the address is im_{i-1}, not the whole vector. The remainder bits could be set at random, as Kanerva suggests. According to the SDM theory, 20 % of the bits correct and the remainder bits set at random should suffice to retrieve the right datum with a probability of 0.6. But it was considered preferable to set up the software so that it is able to calculate similarity between just part of two vectors, ignoring the remainder bits. This saves computational power and reduces the probability of false positives being detected. Since we're not using the *overhead* to address, the tolerance to noise increases a little bit, resulting in less possible errors during normal operation.
Another difference in our implementation, relative to Kanerva's proposal, is that we don't fill the virtual space placing hard locations randomly in the addressing space in the beginning of the operation. Instead, we use Ratitch et al's Randomised Reallocation algorithm Ratitch & Precup (2004): start with an empty memory, and allocate new hard locations when there's a new datum which cannot be stored in enough existing locations. The new locations are allocated *randomly* in the neighbourhood of the new datum address.

6.1 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, which increases the processing time. Furber et al. (2004) claim their results show that the memory's performance is not significantly affected if a single bit is used to store one bit, instead of a bit counter, under normal circumstances. For real time operation, this simplification greatly reduces the need for processing power and memory size. In our case, the original model was not implemented, and the system's performance was ac-

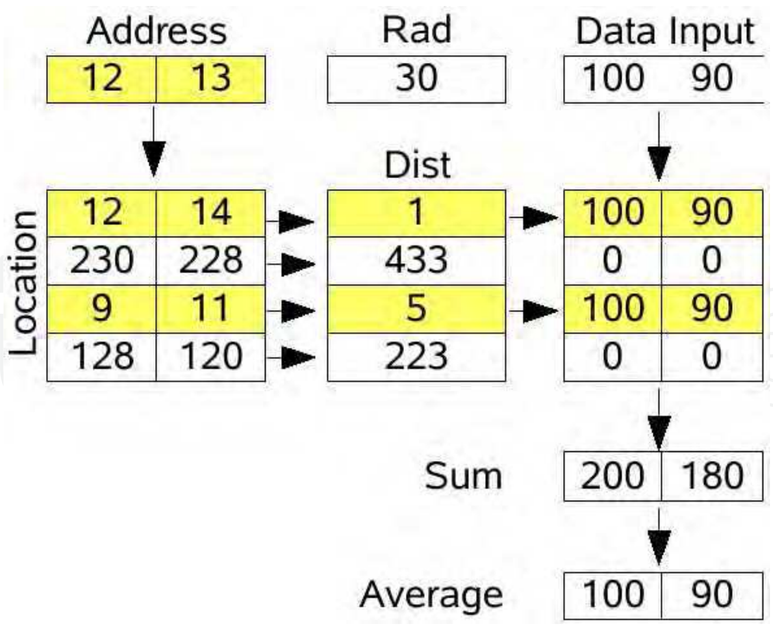

Fig. 9. Arithmetic SDM, which works with byte integers.

ceptable using this implementation where the bit counters are replaced by a single bit each, as shown in Figure 8.

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

6.2 Arithmetic implementation

Although the bitwise implementation works, we also implemented another version of the SDM, inspired by Ratitch & Precup (2004). In this variation of the model, the bits are grouped as byte integers, as shown in Figure 9. Addressing is done using an arithmetic distance, instead of the Hamming distance, and writing is done by applying to each byte value the following equation:

$$h_t^k = h_{t-1}^k + \alpha \cdot (x^k - h_{t-1}^k), \quad \alpha \in \mathbb{R} \land 0 \leq \alpha \leq 1 \tag{11}$$

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

6.3 Determination of the access radius

To calibrate the system, it is necessary to previously estimate noise levels. We consider noise the distance between two consecutive pictures taken without any environmental change (i.e., the lighting and scenario are the same).

To make navigation a little more robust, a dynamic algorithm was implemented, to automatically adjust the system to the noise level before learning. Once the system is turned on, the robot captures three consecutive images and computes the differences between the first and

the second, and between the second and the third. The average of the two values is taken as a good approximation to the actual noise level.
As proposed in Mendes et al. (2007), to make sure the images are retrieved from the memory when following the sequence, the access radius is set as a function of the noise level. It is set to the average value of the noise increased 40 % for bitwise operation, and 70 % for arithmetic operation.

7. The 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 Hamming Distance (HD) is 2. If we consider 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 4. However, sensorial data is usually encoded using the natural binary code and is hardly random. This means that the data might not be appropriate to make the most of the SDM.
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 twice the value of `0011`. This characteristic means that the Hamming Distance (HD) is not proportional to the binary difference of the 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

Table 4. Hamming distances for 3-bit numbers.

Table 4 shows the HDs between all the 3-bit binary numbers. As it shows, this distance is not proportional to the Arithmetic Distance (AD). The HD sometimes even decreases when the arithmetic difference increases. One example is the case of `001` to `010`, where the AD is 1 and the HD is 2. And if we compare `001` to `011`, the AD increases to 2 and the HD decreases to 1. In total, there are 9 undesirable situations in the table, where the HD decreases while it should increase or, at least, maintain its previous value. In the case of PGM images, they are encoded using the natural binary code, which takes advantage of the position of the bits to represent different values. But the HD does not consider positional values. The performance of the SDM, therefore, might be affected because of these different criteria being used to encode the information and to process it inside the memory.
These characteristics of the NBC and the HD may be neglectable when operating with random data, but in the specific problem of storing and retrieving graylevel images, they may pose serious problems. Suppose, for instance, two different copies, im_i and im_j, of the same image. Let's assume a given pixel P has graylevel 127 (`01111111`) in im_i. But due to noise,

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

Table 5. Hamming distances for 3-bit Gray Code.

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

Table 6. Hamming distances for a 3-bit optimised code.

P has graylevel 128 (`10000000`) in im_j. Although the value is *almost* the same, the Hamming distance between the value of P in each image is the maximum it can be—8 bits.
A solution to this problem could rely on the use of the Gray Code (GC), where only one bit changes at a time as the numbers increase. This would ensure that transitions such as the one from 7 to 8 have only a difference of one, while in NBC all the bits differ.
The GC, however, also exhibits many undesirable transitions, as Table 5 shows. It may solve some particular problems of adjacent bytes, but it's not a general solution. Besides, it is circular, meaning that a white image can easily be confused with a black image.
Another approach is simply to sort the bytes in a more convenient way, so that the HD becomes proportional to the AD—or, at least, does not exhibit so many undesirable transitions. This sorting can be accomplished by trying different permutations of the numbers and computing the matrix of Hamming distances. For 3-bit numbers, there are 8 different numbers and $8! = 40,320$ permutations. This can easily be computed using a modern personal computer, in a reasonable amount of time. After an exhaustive search, different sortings are found, but none of them ideal. Table 6 shows a different sorting, better than the NBC shown in Table 4. This code shows only 7 undesirable transitions, while the NBC contains 9. Therefore, it should perform better with the SDM. It should also be mentioned that there are several sortings with similar performance. There are 2,783 other sortings that also have seven undesirable transitions. The one shown is the first that our software found. In our case, we are looking for the best code to compute the similarity distance between images. If those images are equalised, then the distribution of all the brightness values is such that all the values are approximately equally probable. This means that it is irrelevant which sorting is chosen, among those with the same number of undesirable transitions. Yet another approach is to use a Sum-Code (SC). As previously written, using 256 graylevels it's not possible to find a suitable binary code with minimum undesirable transitions, so that one can take advantage of the representativity of the NBC and the properties of the SDM. The only way to avoid undesirable transitions at all is to reduce the number of different gray levels to the number of bits + 1 and use a kind of

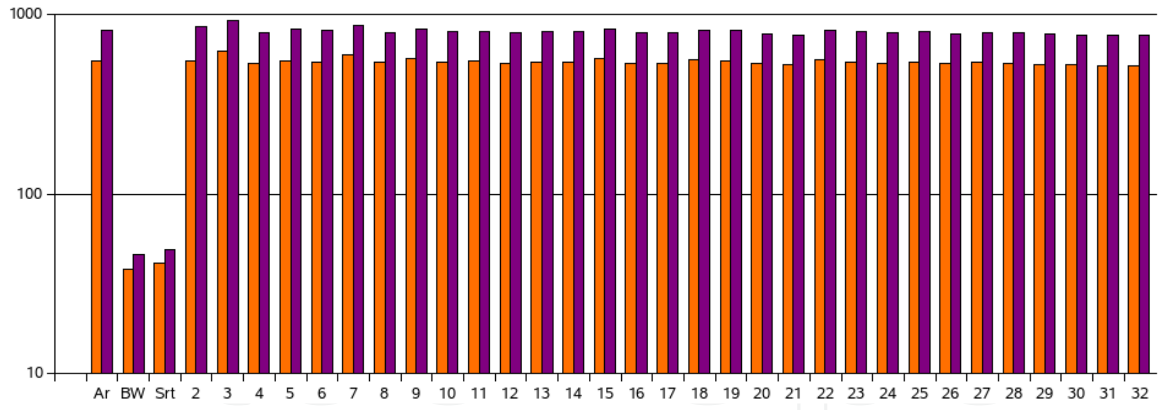

Fig. 10. Error increments. Light colour: to the second best image. Darker: to the average.

0	0000
1	0001
2	0011
3	0111
4	1111

Table 7. Code to represent 5 graylevels.

sum-code. Therefore, using 4 bits we can only use 5 different gray levels, as shown in Table 7. Using 8 bits, we can use 9 gray levels and so on. This is the only way to work with a Hamming distance that is proportional to the arithmetic distance.
The disadvantage of this approach, however, is obvious: either the quality of the image is much poorer, or the dimension of the stored vectors has to be extended to accommodate additional bits. In Mendes et al. (2009) this encoding problem is discussed in more detail.

8. Tests and Results

Different tests were performed in order to assess the behaviour of the system using each of the approaches described in the previous sections. The results were obtained using a sequence of 55 images. The images were equalised, and the memory was loaded with a single copy of each.

8.1 Results

Table 8 shows the average of 30 operations. The tests were performed using the arithmetic distance; using the NBC and the Hamming distance (8 bits, 256 graylevels, represented using the natural binary code); using the the Hamming distance and a partially optimised sorting of the bytes; and bitwise modes in which the graylevels were reduced to the number of bits + 1. We tested using from 1 to 32 bits, which means from 2 to 33 graylevels, in order to experimentally get a better insight on how the number of bits and graylevels might influence the performance of the system.
The table shows the distance (error in similarity) from the input image to the closest image in the SDM; the distance to the second closest image; and the average of the distances to all the images. It also shows, in percentage, the increase from the closest prediction to the second and to the average—this is a measure of how *successful* the memory is in separating the desired datum from the pool of information in the SDM. We also show the average processing time for each method. Processing time is only the memory prediction time, it does not include the

image capture and transmission times. The clock is started as soon as the command is issued to the SDM and stopped as soon as the prediction result is returned.
For clarity, chart 10 shows, using a logarithmic scale, the increments of the distance from the closest image to the second closest one (lighter colour) and to the average of all the images (darker colour).

	Dist. to best	Dist. to 2^{nd}	inc. (%)	Dist. to Average	inc. (%)	Time (ms)
Ar.	18282	118892	550.32	166406.53	810.22	241.29
NBC	6653	9186	38.07	9724.80	46.17	231.35
Sort.	6507	9181	41.09	9720.56	49.39	240.45
B2	101	653	546.53	961.25	851.73	979.31
B3	144	1034	618.06	1465.57	917.76	983.02
B4	232	1459	528.88	2069.46	792.01	964.34
B5	291	1893	550.52	2689.06	824.08	970.88
B6	365	2349	543.56	3308.30	806.38	974.53
B7	412	2849	591.50	3964.05	862.15	963.56
B8	517	3312	540.62	4605.01	790.72	968.84
B9	569	3791	566.26	5257.01	823.90	996.56
B10	654	4214	544.34	5897.50	801.76	981.74
B11	724	4706	550.00	6546.08	804.15	968.81
B12	810	5142	534.81	7183.31	786.83	969.92
B13	871	5608	543.86	7817.77	797.56	971.62
B14	944	6084	544.49	8469.16	797.16	983.49
B15	986	6555	564.81	9126.96	825.66	992.54
B16	1098	6963	534.15	9750.75	788.05	977.52
B17	1180	7487	534.49	10424.05	783.39	967.14
B18	1208	7938	557.12	11040.56	813.95	965.06
B19	1290	8410	551.94	11729.28	809.25	968.77
B20	1406	8843	528.95	12377.95	780.37	975.30
B21	1498	9298	520.69	13015.70	768.87	996.89
B22	1494	9794	555.56	13680.24	815.68	978.63
B23	1591	10230	542.99	14290.35	798.20	968.75
B24	1687	10679	533.02	14934.10	785.25	977.01
B25	1744	11178	540.94	15616.34	795.43	971.71
B26	1850	11646	529.51	16277.14	779.85	974.81
B27	1898	12086	536.78	16880.55	789.39	999.59
B28	1988	12533	530.43	17558.76	783.24	965.80
B29	2083	13000	524.10	18178.87	772.73	965.99
B30	2175	13512	521.24	18878.92	768.00	968.89
B31	2263	13936	515.82	19489.28	761.21	981.75
B32	2336	14433	517.85	20163.64	763.17	967.21
B33	2372	14900	528.16	20796.13	776.73	1012.32

Table 8. Experimental results using different metrics.

Some results are also plotted in charts 10 and 11. The first shows, using a logarithmic scale, the increments of the distance from the closest image to the second closest one (lighter colour)

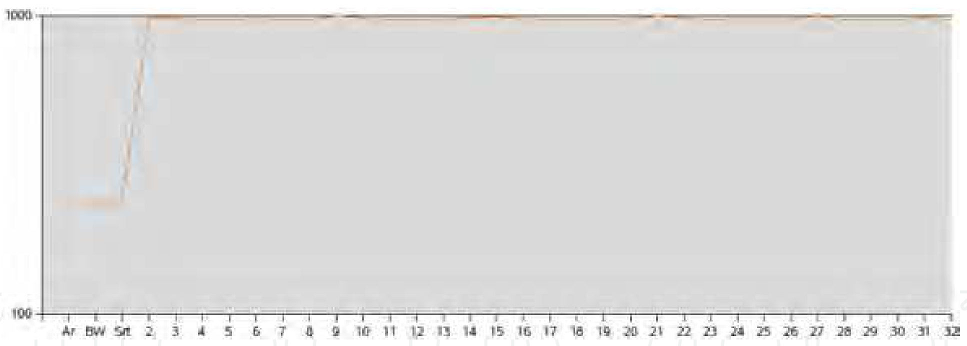

Fig. 11. Processing time.

and to the average of all the images (darker colour). The second chart shows the processing time, again using a logarithmic scale.

8.2 Analysis of the results

It can be confirmed that the bitwise mode using the NBC seems to be remarkably worse than the other methods, which seem to show similar results. Sorting the bytes results in a small, but not significant, improvement. Another interesting point is that the number of graylevels seems to have little impact on the selectivity of the image, for images of this size and resolution.
The processing time exhibits a great variation, due to the fact that the tests were run on a computer using Linux (OpenSuSE 10.2), a *best effort* operating system. Even with the number of processes running down to the minimum, there were very disparate processing times. For better precision and real time operation, a real time operating system would be recommended. The average processing times for the arithmetic and bitwise mode are about 240 ms for the complete cycle to fetch the closest matching image. Using the NBC with the HD, the time is a little shorter, and using a different sorting of the bytes the time increased a little. This was expectable, since the only variation in this method was implemented using an indexed table, where each position held the sorted byte. Therefore, to compute the similarity between two pixels, two accesses had to be done to the indexed table, which considerably increases the total memory access time. A more efficient approach would be to make the conversion as soon as the images were grabbed from the camera. This is undesirable in our case, though, as we're also testing other approaches.
As for the other approaches using different graylevels, the processing times are all similar and about 4 times larger than the time of processing one image using the arithmetic mode. The reason for this is that, again, an indexed table is used to address the binary code used. And in this case there's the additional workload of processing the conversion into the desired number of gray values. In a production system, obviously, the conversion would only need to be done once, just as the images were grabbed from the camera.

9. Conclusions

To the best of our knowledge, this is the first time a robot has actually been guided by a system with a SDM at its helm. Our implementation consisted in guiding the robot using a view sequence stored in the SDM.
The first problem noticed was that of encoding the information, because sensorial information is hardly random, as the SDM theory considers. Thus, our tests used four different operational

modes: one based upon the natural binary code; another based on an optimised code; a third one based on an arithmetic mode; and the last using a sum-code.
In the original SDM model Kanerva proposes that the Hamming distance be used to compute the similarity between two memory items. Unfortunately, this method exhibits a poor performance if the data are not random. The NBC with the Hamming distance shows the worst performance. By sorting some bytes the performance is slightly improved. If the bits are grouped as bytes and an arithmetic distance is used, the memory shows an excellent performance, but this can fade some characteristics of the original model, which is based on the properties of a binary space. If the number of graylevels is reduced and a sum-code is used, the performance is close to that of the arithmetic mode and the characteristics of the memory must still hold..
Although our work was performed using images as data, our results should still be valid for all non-random data, as is usually the case of robotic sensorial data.
Future work includes the study of the impact of using different encoding methods on the performance of the SDM itself, in order to infer which characteristics shall still hold or fade.
Implemented in software the SDM is very computer intensive and storage can be as low as 0.1 bits per bit. However, it is a viable and interesting model, making it possible to implement models and behaviours similar to the human brain, based on pattern recognition, such as the case of navigation based on visual sequences.

10. References

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

Dijskstra, E. W. (1959). A note on two problems in connexion with graphs, *Numerische Mathematik* **1**(1): 269–271.

Dobrev, D. (2005). Formal definition of artificial intelligence, *Information Theories and Applications* **12**(3): 277–285.

Franz, M. O., SchÃűlkopf, B., Mallot, H. A., BÃijlthoff, H. H. & Olkopf, B. S. (1998). Learning view graphs for robot navigation, *Autonomous Robots* **5**(1): 111–125.

Furber, S. B., Bainbridge, J., Cumpstey, J. M. & Temple, S. (2004). Sparse distributed memory using *n*-of-*m* codes., *Neural Networks* **17**(10): 1437–1451.

Gottfredson, L. S. (1997). Mainstream science on intelligence: An editorial with 52 signatories, history and bibliography, *Intelligence* **24**(1): 13–23.

Harnish, R. M. (2002). *Minds, brains, computers: an historical introduction to the foundations of cognitive science*, Wiley-Blackwell.

Hawkins, J. & Blakeslee, S. (2004). *On Intelligence*, Times Books, New York.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities., *Proceedings of the National Academy of Science*, Vol. 79, pp. 2554–2558.

Ishiguro, H., Miyashita, T. & Tsuji, S. (1995). T-net for navigating a vision-guided robot in real world, *ICRA*, pp. 1068–1074.

Ishiguro, H. & Tsuji, S. (1996). Image-based memory of environment, *in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pp. 634–639.

Jones, S. D., Andresen, C. & Crawley, J. L. (1997). Appearance based processes for visual navigation, *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Grenoble, France.

Kanerva, P. (1988). *Sparse Distributed Memory*, MIT Press, Cambridge.

Keeler, J. D. (1988). Comparison between kanerva's sdm and hopfield-type neural networks, *Cognitive Science* **12**(3): 299–329.
URL: *http://dx.doi.org/10.1016/0364-0213(88)90026-2*

Kuipers, B. J. & Levitt, T. S. (1988). Navigation and mapping in large-scale space, *International Journal of Artificial Intelligence* **9**(2): 25–43.

Legg, S. & Hutter, M. (2007). Universal intelligence: A definition of machine intelligence, *Minds and Machines* **17**(4): 391–444. http://www.vetta.org/about-me/publications/.

Matsumoto, Y., Ikeda, K., Inaba, M. & Inoue, H. (1999). Exploration and map acquisition for view-based navigation in corridor environment, *Proceedings of the International Conference on Field and Service Robotics*, pp. 341–346.

Matsumoto, Y., Inaba, M. & Inoue, H. (2000). View-based approach to robot navigation, *Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000).*

Matsumoto, Y., Inaba, M. & Inoue, H. (2003). View-based navigation using an omniview sequence in a corridor environment, *Machine Vision and Applications*.

Mendes, M., Coimbra, A. P. & Crisóstomo, M. (2007). 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.

Mendes, M., Crisóstomo, M. & Coimbra, A. P. (2009). Assessing a sparse distributed memory using different encoding methods, *Proceedings of the 2009 International Conference of Computational Intelligence and Intelligent Systems*, London, UK.

Mountcastle, V. (1978). An organizing principle for cerebral function: the unit model and the distributed system, *in* G. M. Edelman & V. Mountcastle (eds), *The mindful brain*, MIT Press, Cambridge, Mass.

Rao, R. & Fuentes, O. (1998). Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory, *Machine Learning* **31**(1-3): 87–113.

Rasmussen, C. & Hager, G. D. (1996). Robot navigation using image sequences, *In Proc. AAAI*, pp. 938–943.

Ratitch, B. & Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning., *ECML*.

Searle, J. (1980). Minds, brains, and programs, *Behavioral and Brain Sciences* (3): 417–424.

Spearman, C. E. (1927). *The abilities of man, their nature and measurement*, Macmillan, New York.

Watanabe, M., Furukawa, M. & Kakazu, Y. (2001). Intelligent agv driving toward an autonomous decentralized manufacturing system, *Robotics and computer-integrated manufacturing* **17**(1-2): 57–64.
URL: *citeseer.ist.psu.edu/274654.html*

Willshaw, D. J., Buneman, O. P. & Longuet-Higgins, H. C. (1969). Non-holographic associative memory, *Nature* **222**(5197): 960–962.

Winters, N. & Santos-Victor, J. (1999). Mobile robot navigation using omni-directional vision, *In Proc. 3rd Irish Machine Vision and Image Processing Conference (IMVIP'99)*, pp. 151–166.