Improved Vision-based Robot Navigation Using a SDM and Sliding Window Search

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

Abstract—**Robust and fast vision-based robot navigation is a long sought goal, which requires comparing the robot's current view with a database of visual memories. The technique described in the present paper uses a Sparse Distributed Memory (SDM) to store paths described by sequences of images, and a sliding window to narrow the search space for real-time operation. The use of the sliding window greatly reduces the processing time and the number of prediction errors. The use of a short-term memory confers on the robot the ability to still solve the kidnapped robot problem. The Sparse Distributed Memory is a kind of associative memory suitable to work with high-dimensional binary vectors, thus being appropriate to store long sequences of images.**

Index Terms—**Robot Navigation, View-based Navigation, Sliding Window, SDM, Sparse Distributed Memory**

1 Introduction

Among all the techniques that may be used for robot localisation, vision-based approaches are probably the most cherished, for they are biologically inspired. Humans strongly depend on visual memories: approximately 80% of the sensorial information processed by an average person is visual [1]. Additionally, the sensors required for vision-based robot navigation are inexpensive. An average quality video camera may be enough for the simpler applications.

The main drawback of vision-based approaches is that the processing power needed is huge. Every single image is usually described by several hundreds or thousands of pixels, and every path that the robot learns is described by tens, hundreds or even thousands of images, depending on the path length and frame rate used. That makes the technique less appealing, because real-time operation may be compromised for large databases, due to increasing needs of processing power and error probability. As the robot learns more and more paths, the number of images it has to store grows continually. The memory requirements and the processing time increase in proportion to the database size. It should be noted, however, that modern evidence indicates that the human brain functions in a similar manner: it is a huge amount of memory, used to store sequences of events that will lead future analysis and actions [2, 3].

The images alone are a means for instantaneous localisation. View-based navigation is almost always based on the same idea: during a learning stage the robot learns a sequence of views and motor commands that, if followed with minimum drift, will lead it to a target location. The robot is later able to follow the learnt path by following the sequence of commands, possibly correcting the small drifts that may occur.

In previous work the authors presented a system to navigate a robot using images stored into a Sparse Distributed Memory (SDM) [4]. The SDM is a kind of associative memory based on the properties of high-dimensional boolean spaces, and thus suitable to work with large binary vectors such as images [3]. The method was efficient even under difficult conditions [5], but the processing requirements were very demanding. The present paper, which is an extended version of [6], describes an improvement to the system, in which a search sliding window truncates the search space and thus considerably reduces the time and processing requirements, as well as the number of robot localisation errors.

Section 2 reviews some popular navigation techniques and explains navigation based on view sequences in more detail. Section 3 briefly describes how the SDM works. In Section 4 the experimental platform used is described. Section 5 explains the problems encountered with the original navigation method, and how the application of a sliding window contributes to solve many of them. Section 6 shows and discusses the results obtained, and Section 8 draws some conclusions.

2 Robot Navigation

There are many different approaches for robot navigation. Vision-based approaches have been extensively used, although other techniques may be better for some types of common environments.

*ISR - Institute of Systems and Robotics, Dept. of Electrical and Computer Engineering, University of Coimbra, Portugal. E-mail: acoimbra@deec.uc.pt, mcris@isr.uc.pt. †ESTGOH, Polytechnic Institute of Coimbra, Portugal. E-mail: mmendes@estgoh.ipc.pt.

(Advance online publication: 12 November 2011)

2.1 Popular techniques

Besides vision-based techniques, many different approaches have been tried to localise and navigate robots in a safe and robust way. Some of those approaches work only in structured environments, since they are based on the recognition of artificial landmarks, beacons, indoor/outdoor GPS or similar strategies [7]. Those strategies greatly improve the accuracy of the system. They trim the complexity of the environment and narrow the robot search space. The problem of robot localisation becomes much simpler, for the robot only has to look for selected signals and cut off all the remaining data that is unnecessary. The disadvantages, however, are obvious: such approaches are suitable only for structured environments. They are not a general solution to the problem of robot localisation.

More generic strategies that work in unstructured environments include mapping and localisation using laser range finders, as well as sonars or cameras for vision-based approaches. In this case the robot is equipped and programmed in such a way that it is expected to be able to succeed in localising itself in a wide range of environments, and those environments do not have to be intentionally structured in order for the robot to succeed. It should be able to map the environment, regardless of its characteristics, and later use that map to localise itself autonomously.

Popular sensors used for robot localisation are sonars, infrared and laser range finders. Sonars are cheap but in general offer poor precision. Infrared sensors offer better precision, but they hardly work on clear daylight. Lasers are expensive and may also have problems working in daylight. On the other hand, an average quality video-camera is nowadays cheap and can work in a wide range of environments with different illumination levels [8].

2.2 Navigation using view sequences

There are two popular approaches for vision-based navigation: one that uses plain images [9], the other that uses omnidirectional images [10]. Omnidirectional images offer a 360° view, which is richer than a plain front or rear view. However, that richness comes at the cost of even additional processing power requirements. Some authors have also proposed techniques to speed up processing and/or reduce memory needs. Matsumoto [11] used images as small as 32×32 pixels. Ishiguro replaced the images by their Fourier transforms [12]. Winters compressed the images using Principal Component Analysis [13].

In the present work, the approach followed to navigate the robot is based on using visual memories stored into a Sparse Distributed Memory, as described in [4]. It requires a supervised learning stage, in which the robot is manually guided. While being guided, the robot memorises a sequence of views automatically. It stores a sequence of views for each path. Images that are very similar to previously stored images are discarded, because they would, with high probability, not add any relevant information to the known information. They might even disturb important information already stored into the SDM, as explained further in Section 3.

While running autonomously, the robot performs automatic image-based localisation and obstacle detection. Localisation is estimated based on the similarity of two views: one stored during the supervised learning stage and another grabbed in real-time. To minimise possible drifts to the left or to the right, 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, being able to reduce the drift iteratively over time. There is no need to process the vertical distance, since the camera is fixed and vertical shifts are not expected. The technique is described in more detail in [4].

3 Sparse Distributed Memories

The Sparse Distributed Memory is an associative memory model proposed by Kanerva in the 1980s [3]. It is suitable to work with high-dimensional binary vectors. In the proposed approach, an image is regarded as a high-dimensional vector, and the SDM is used simultaneously as a sophisticated storage and retrieval mechanism and a pattern recognition tool.

3.1 Previous use of the SDM for robot navigation

The concept of the SDM is very attractive for robot navigation. It confers on the robot the ability to learn and follow paths the same way humans do. Some researchers have already explored the idea to some extent. Rao and Fuentes [14] simulated the use of a SDM to store position information from optical sensors, associated with motor controls, during a learning stage. Using that information, the robot was later able to follow the same paths. The authors presented only simulation results. Watanabe et al. [15] used a SDM for the task of scene recognition in a factory environment, where the robots had to move autonomously from one place to another. The SDM, however, was just used as an auxiliary method of scene recognition. It was not used for robot navigation purposes.

3.2 The original model

The underlying idea behind the SDM is the mapping of a huge binary memory onto a smaller set of physical locations, called *hard locations*. That way it is possible to mimic the existence of a much larger space, taking ad-

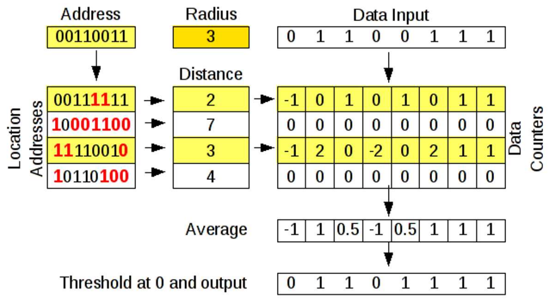

Figure 1: One model of a SDM, using bit counters. The address bits highlighted are different in the address vector, compared to the corresponding bits in the input address.

vantage of its inherent properties. As a general guideline, those hard locations should be uniformely distributed in the virtual space, to *mimic* the existence of the larger virtual space as accurately as possible. Every datum may be stored into various hard locations. Storage is performed into a set of hard locations that exist within a given radius. Retrieval is performed by *averaging* those locations and comparing the result to a given threshold. 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. It will activate all the hard locations within 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, be used as the measure of distance between the addresses. All the locations that differ less than a predefined number of bits from the input address are selected for the read or write operation.

Data are stored in arrays of counters, one counter for every bit of every location. Writing is done by incrementing or decrementing the bit counters at the selected addresses. To store 0 at a given position, the corresponding counter is decremented. To store 1, it is incremented. Reading is done by averaging the values of all the counters columnwise and thresholding at a predefined value. If the value of the sum is below the threshold, the bit is zero, otherwise it is one. Initially, all the bit counters must be set to zero, for the memory stores no data. The bits of the address locations should be set randomly, so that the addresses would be uniformely distributed in the addressing space.

Such a memory exhibits the properties of a large boolean space. It is proven mathematically that those properties include, among others, high tolerance to noisy data, ability to deal with incomplete data, natural forgetting over time and ability to process sequential information. Hence, a robot equipped with such a memory at its helm must be able to succeed in a wide range of difficult situations which are typical in robot navigation, such as partial occlusion, illumination changes and memory overflow. A number of experiments have been carried out to prove that assumption, and the results are described in [8].

Another interesting characteristic of the SDM model is that the same set of vectors can be used simultaneously to store the addresses and the data, as long as any given datum ζ is always stored at address ζ. A practical consequence of this is that one of the arrays can be discarded, cutting the memory size down to about one half its original size. Such a memory, comprising only one array, where datum ζ is stored at address ζ, is called auto-associative.

3.3 The models used

The original SDM model has been subject to various improvements and alternative implementations. In the present work, four variations have been studied: the arithmetic mode, the bitwise mode using the natural binary code, the bitwise mode using an optimised code, and the bitwise mode using a sum-code. Those models are described in [5].

All the models used are auto-associative and use the Randomised Reallocation (RR) algorithm [16]. Using the RR, the system starts with an empty memory and allocates new hard locations when there is a new datum which cannot be stored into enough existing locations. The new locations are placed *randomly* in the neighbourhood of the new datum address.

3.3.1 Bitwise SDM

The bitwise implementation is very similar to the original model. The difference is that it stores only one bit per input vector bit, thus dropping the bit counters, as shown in Figure 2. Writting in such a model consists in just replacing the old datum. The advantages are that the capacity of storing data is improved, and reading and writing is much faster. The model was inspired by Furber et al.'s approach [17].

3.3.2 Use of an optimised code

As described in [5], the Hamming distance between two binary numbers is not proportional to the arithmetic distance. For example, the Hamming distances $h_1(0111_2, 1111_2) = h_2(1110_2, 1111_2) = 1$. That happens because the Hamming distance does not take into account the positional values of the bits. However, the sensorial data is encoded using the natural binary code, which takes into account the positional values of the bits. Using arithmetic distances, $d_1(0111_2, 1111_2) = 8$ and $d_2(1110_2, 1111_2) =$

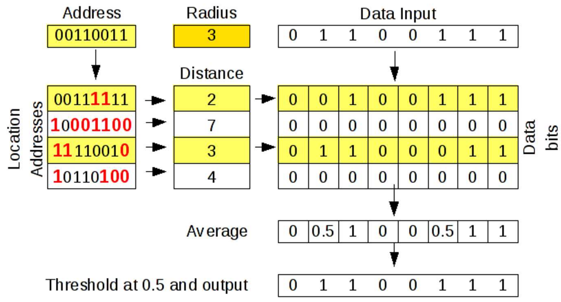

Figure 2: Bitwise SDM model, which contains single bits instead of bit counters.

1. Hence, different criteria are used to encode the input information and to process it inside the SDM according to Kanerva's original model. That difference causes a loss of performance of the system, and to overcome the problem other memory models were implemented. The first alternative encodes the data using an optimised code. In that optimised code some bytes are sorted, in order to minimise the effect of using different criteria to encode the input data and to process it inside the SDM.

3.3.3 Use of a sum-code

In another model, the data is encoded using a sum-code of 9 graylevels. In that code, each binary number is mapped into the range {00000000, 00000001, 00000011,..., 11111111}. That way the Hamming distance between any two binary numbers is proportional to the arithmetic distance.

3.3.4 Arithmetic SDM

In the arithmetic implementation, the bits are grouped as byte integers, as shown in Figure 3. Addressing is done using an arithmetic distance, instead of the Hamming distance. Learning is achieved updating each byte value using the equation:

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

In the equation, h_t^k is the k^{th} number of the hard location, at time t, x^k is the corresponding number in the input vector x and α is the learning rate. In the present implementation α was set to 1, enforcing one shot learning.

4 Experimental platform

The robot used was a Surveyor[1] SRV-1, a small robot with tank-style treads and differential drive via two precision DC gearmotors (Figure 4). Among other features,

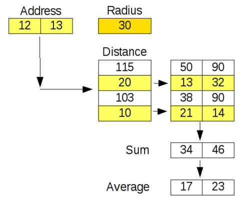

Figure 3: Architecture of the auto-associative arithmetic SDM, using integers instead of bit counters.

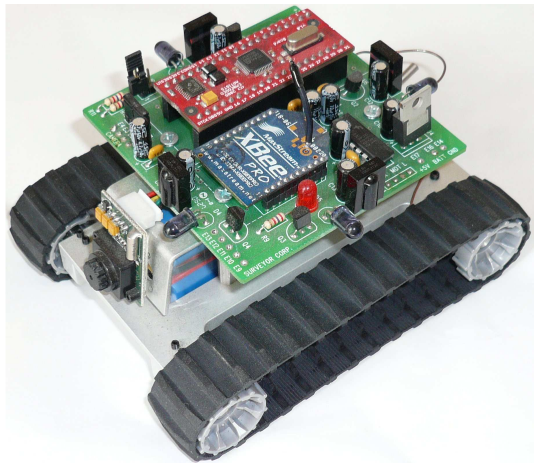

Figure 4: Robot used.

it has a built in digital video camera and a 802.15.4 radio communication module. The robot was controlled in real time from a laptop with a 1.8 GHz processor and 1 Gb RAM. The overall software architecture is as shown in Figure 5. 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. An operational 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: supervised learning, in which the robot is manually guided and captures images to store for future reference; and autonomous running, in which it uses previous knowledge to navigate autonomously, following any sequence previously learnt. The vectors stored in the SDM consist of arrays of bytes, as summarised in Equation 2:

$$x_i =< im_i, seq\_id, i, timestamp, motion > \quad (2)$$

In the vector x_i, im_i is the image i, in PGM (Portable Gray Map) format and 80×64 resolution. In PGM images, every pixel is represented by an 8-bit integer. 0 is black, 255 is white. seq_id is an auto-incremented,

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

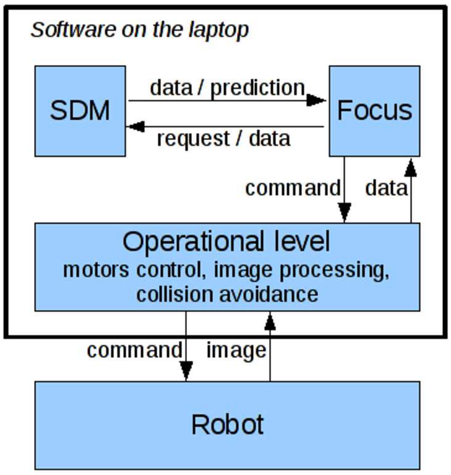

Figure 5: Architecture of the implemented software.

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, used to quickly identify every image in the sequence. *timestamp* is a 4-byte integer, storing Unix timestamp. It is not being used so far for navigation purposes. *motion* is a single character, identifying the type of movement the robot performed after the image was grabbed. The image alone uses 5120 bytes. The overhead information comprises 13 additional bytes. Hence, the input vector contains 5133 bytes.

5 Use of a search window

The use of a search window, which truncates the search space, greatly improves the speed and performance of the method.

5.1 The problems

There are two weaknesses of the view-based navigation approach described: i) processing time required to store and retrieve one image and ii) prediction errors, when the memory outputs a wrong image and motion command.

As for the processing time, it is proportional to the number of images stored in the memory. Each new image has to be compared to all the hard locations that exist in the memory. That may be a problem for real time operation, specially if a single processor is used.

As for the second problem, it is due primarily to the existence of noise in the images, which is impossible to avoid. When following a path, it is normal that the robot makes some wrong predictions. It is difficult to count the exact number of errors, but in this case we define the concept of “Momentary Localisation Error” (MLE). A MLE occurs when the system retrieves image im_{i-j} after having retrieved im_i, for $i, j > 0$. That is a reasonable assumption, since, under normal circumstances, the robot is not expected to get back in the sequence. If at some point of a path the prediction is im_i, and after that it is im_{i-j}, then it means that at least one of the predictions was wrong. Those MLEs are not to worry when the robot is performing the same movement in both the correct and the wrongly retrieved image. That is often the case, since there are only 4 possible motions (forward, backward, turn left and turn right). But a prediction error could compromise the robot’s ability to complete a path if the correct motion and the motion associated with the retrieved image are different.

5.2 Distribution of the Momentary Localisation Errors

Table 1 shows the number of MLEs measured while following a typical path, described by a sequence of 130 images. The first row of the table indicates the distance of the image predicted by the memory to the last predicted image. The first column is the operation mode.

As the table shows, most of the MLEs occur with adjacent images: the distance between the expected image and the retrieved image is 1. More than 60% of the MLEs are between adjacent images, regardless of the memory operation mode. In the bitwise mode the MLEs are more distributed in the range of distances [1–5] than in the other modes. That makes sense, considering that the bitwise mode is, in general, the weakest of all [5]. In the example path no MLEs were detected at distances greater than 5 images, and that is also a normal behaviour of the system. Figure 6 shows an histogram of the distribution of MLEs.

5.3 The use of a sliding window

The use of a sliding window helps improving both the processing time and the number of MLEs. It works like the use of a kind of *context*, in which the topic is narrowed to a given subject. In the case of the SDM, that is equivalent to segmenting the search space.

L. Jaeckel proposed a method of segmenting the space by way of using only a limited set of coordinates, instead of all the binary vector, to determine the set of active locations [18]. The method implemented in the present work has some similarities to Jaeckel’s approach. The idea is narrowing the search field to a number of images before and after the last predicted image, as illustrated in Figure 7. For example, if the robot is following path A and the last image retrieved is image i, in the next prediction it is expected to be still following path A and retrieve either image i or image *i+1*. Since the length of the step used in the autonomous run is $1/16^{th}$ of that used during the learning stage, it will see image i for some time and that is no prediction error. The use of a sliding window of width $2 \times j + 1$ consists in narrowing

Table 1: Distribution of the MLEs according to the operation mode, without search window, in a typical path described by 130 images.

	-5	-4	-3	-2	-1
Arithmetic	0	0	4 (36.4%)	0	7 (63.6%)
Bitwise	2 (3.8%)	3 (5.7%)	2 (3.8%)	9 (17.0%)	37 (69.8%)
Optimised code	0	4 (7.3%)	3 (5.5%)	6 (10.9%)	42 (76.4%)
Sum-code	0	0	1 (7.1%)	3 (21.4%)	10 (71.4%)

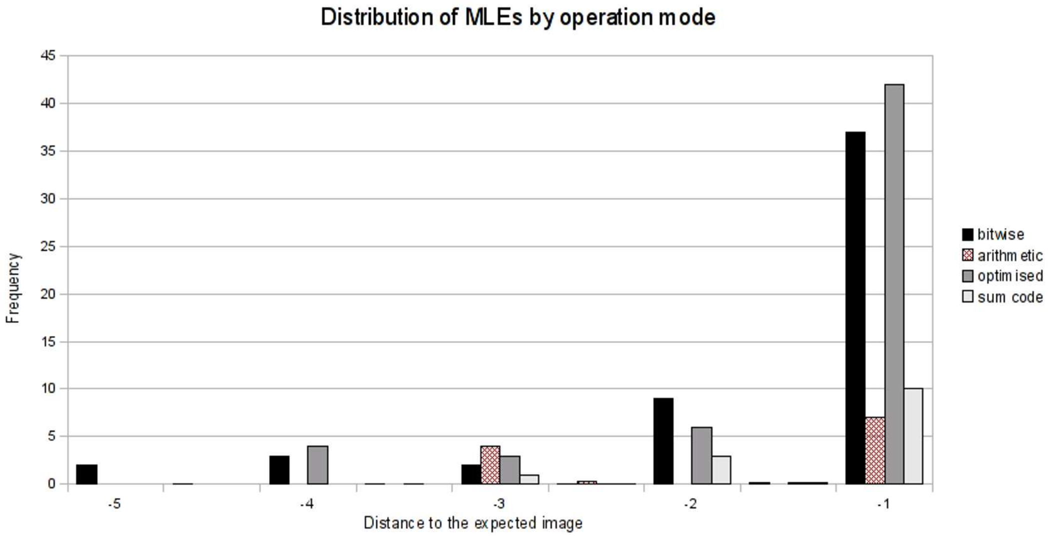

Figure 6: Distribution of the MLEs, in the four operation modes, without search window, in a typical path described by 130 images.

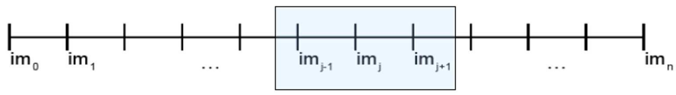

Figure 7: Example of a sliding window of width 3. For image j, search is first performed in the interval $[j-1, j+1]$.

the search field to sequence A and images in the interval $[im_{i-j},\ im_{i+j}]$, for $i, j > 0$. The search algorithm of the SDM was updated, so that it skips images that: i) do not belong to sequence A, and ii) belong to sequence A but are not in the range $[im_{i-j},\ im_{i+j}]$. The images that are within the sequence and the window are processed normally.

6 Experiments and results

As Table 1 shows, more than 60% of the MLEs occur between adjacent images (distance -1). The other MLEs appear at absolute distances of 2, 3, 4 or 5 images. Although those errors account for less than 40% of the total, they are still undesirable.

In order to assess the performance of the system using a sliding window, the navigation algorithm was updated to narrow the search to the same sequence and a window of three images, in the interval $[im_{i-1},\ im_{i+1}]$—*i.e.*, for each image, perform the search by comparing just with the last seen image, the image prior to that one in the sequence and the next expected image in the sequence.

Table 2 shows the results obtained when following the already presented example path, using a search window of width 3 . One interesting conclusion is that the search window cut more MLEs than those counted out of its range, except for the arithmetic mode. That is explained by the fact that some MLEs may actually be the reason of other MLEs. For example, a MLE that causes a wrong motion of the robot may cause drifts and additional MLEs in the future. The improvements are of 50% or more, except for the arithmetic mode.

Figure 8 illustrates the data shown in Table 2, related to the number of MLEs counted with and without using the search window. The histogram clearly shows the impact of the method, specially in the bitwise modes.

Figure 9 illustrates the differences in processing time, as shown in Table 2. It is clear that there is an improvement of about 93% in the processing time. That makes sense, considering that the memory is loaded with a sequence of 130 images. The use of the search window makes the

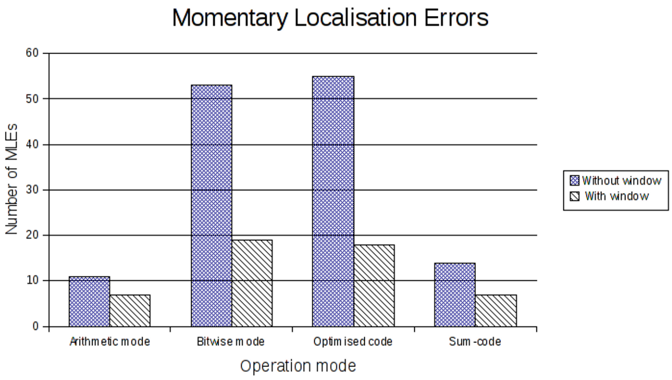

Figure 8: Comparison of the number of MLEs with and without search window, in a typical path described by 130 images.

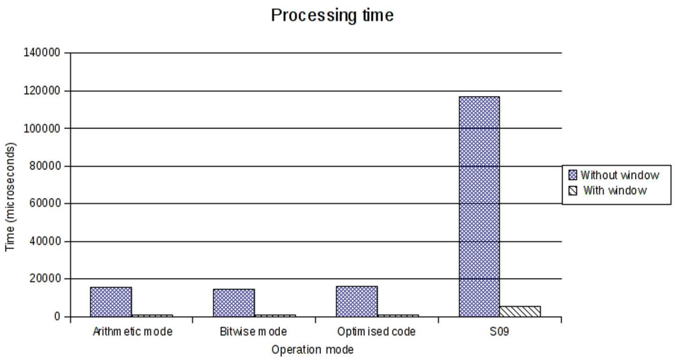

Figure 9: Comparison of the processing time with and without search window.

algorithm skip all but three images, and those three images represent only 2.31% of the whole sequence. Since most of the time necessary to make a prediction is actually spent comparing images, an improvement of 93% is coherent with the theory.

7 Discussion

As shown in Section 6, the use of a search window greatly improves the performance of the system. In the example path it reduced the number of momentary localisation errors up to 67%, and the processing time up to 95%. That improvements are possible at the cost of truncating the search space. Under normal circumstances, truncating the search space should pose no problem to the robot. However, the solution looses generality, because it is strongly based on the robot's short memory: the algorithm is based on the assumption that the robot is always *close* to its last position. Nonetheless, it may happen that the robot slips while moving, is manually moved by a human to another location, etc. That is commonly known as the as the "kidnapped robot" problem.

To achieve robust navigation, a robot must not rely strictly on a search window, otherwise it will not solve the kidnapped robot problem. Using a SDM that problem may be easily overcome. A general solution to the problem is to use an algorithm that, for each new image:

1. Search within the sliding window. If the search retrieves one or more images within the SDM access radius (as explained in Section 3.2), then assume that the prediction is correct.
2. If the search within the sliding window does not retrieve at least one image within the SDM access radius, then perform a global search in the SDM and use the best prediction.

The algorithm as described still takes advantage of the sliding window under normal circumstances, and is able to solve the kidnapped robot problem.

8 Conclusions

Robot navigation based on visual memories is a long sought goal. However, it requires heavy processing due to the amount of information that has to be processed in real time. The approach followed in the present work is vision-based robot navigation using images stored into a Sparse Distributed Memory. The speed of the process can be largely improved with the use of a search window. The search window truncates the search space, reducing significantly the processing time as well as the number of prediction errors, thus improving the real time performance operation of the robot.

Acknowledgements

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

References

[1] Steven Johnson. *Mind wide open.* Scribner, New York, 2004.

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

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

[4] Mateus Mendes, Manuel M. Crisóstomo, and A. Paulo Coimbra. Robot navigation using a sparse distributed memory. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, May 2008.

[5] Mateus Mendes, Manuel M. Crisóstomo, and A. Paulo Coimbra. Assessing a sparse distributed memory using different encoding methods. In *Proceedings of the World Congress on Engineering 2009, WCE 2009*, London, UK, July 2009.

Table 2: MLEs and processing time without using search window and with search window of size 3, in a typical path described by 130 images.

		Arithmetic mode	Bitwise mode	Optimised code	Sum-code
MLE	Without search window	11	53	55	14
	With search window	7	19	18	7
	Improvement	36%	64%	67%	50%
Time (μs)	Without search window	15 511.38	14 567.16	16 269.21	116 846.54
	With search window	1 011.14	988.44	1 000.95	5 539.48
	Improvement	93%	93%	94%	95%

[6] Mateus Mendes, A. Paulo Coimbra, and Manuel M. Crisóstomo. Robot navigation using view sequences and sliding window search. In *Lecture Notes in Engineering and Computer Science: Proceedings of the World Congress on Engineering 2011, WCE 2011*, London, UK, July 2011.

[7] Christopher Rasmussen and Gregory D. Hager. Robot navigation using image sequences. In *In Proc. AAAI*, pages 938–943, 1996.

[8] Mateus Mendes, A. Paulo Coimbra, and Manuel M. Crisóstomo. Intelligent robot navigation using view sequences and a sparse distributed memory. *Paladyn Journal of Behavioural Robotics*, 1(4), 2011.

[9] Yoshio Matsumoto, Kazunori Ikeda, Masayuki Inaba, and Hirochika Inoue. Exploration and map acquisition for view-based navigation in corridor environment. In *Proc. of the Int. Conference on Field and Service Robotics*, pages 341–346, 1999.

[10] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. View-based navigation using an omniview sequence in a corridor environment. In *Machine Vision and Applications*, 2003.

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

[12] Hiroshi Ishiguro and Saburo Tsuji. Image-based memory of environment. In *in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1996.

[13] Niall Winters and José Santos-Victor. Mobile robot navigation using omni-directional vision. In *In Proc. 3rd Irish Machine Vision and Image Processing Conference (IMVIP'99)*, pages 151–166, 1999.

[14] Rajesh P.N. Rao and Olac Fuentes. Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory. *Machine Learning*, 31(1-3):87–113, April 1998.

[15] Michiko Watanabe, Masashi Furukawa, and Yukinori Kakazu. Intelligent agv driving toward an autonomous decentralized manufacturing system. *Robotics and computer-integrated manufacturing*, 17(1-2):57–64, February-April 2001.

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

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

[18] Louis A. Jaeckel. An alternative design for a sparse distributed memory. Technical report, Research Institute for Advanced Computer Science, NASA Ames Research Center, July 1989.