Assis - Cicerone Robot With Visual Obstacle Avoidance Using a Stack of Odometric Data

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

Abstract— **Modern research has shown that intelligent behaviour is, to a great extent, strongly supported on the use of a sophisticated memory. Additionally, vision is the main source of information for the average human. Constant increase of sensing and processing power and constant decrease of the cost of memory have made vision-based approaches in robotics more attractive. ASSIS is a prototype of an assistant robot which uses vision and memory-based navigation, including intelligent obstacle avoidance. ASSIS uses a parallel implementation of a Sparse Distributed Memory to store images and use them later for localisation and navigation. Obstacle avoidance is achieved using a stack-based method and odometric data. Users can command the robot through a web-based interface. ASSIS implements behaviours such as courier or cicerone.**

Index Terms—**ASSIS, Autonomous Navigation, Obstacle Avoidance, SDM, Vision-based Localization, Vision-Based Navigation.**

I. Introduction

DEVELOPMENT of intelligent robots is an area of intense and accelerating research. Different models for localization and navigation have been proposed. The present approach uses a parallel implementation of a Sparse Distributed Memory (SDM) as the support for vision and memory-based robot localization and navigation, including obstacle avoidance [1]. The SDM is a type of associative memory suitable to work with high-dimensional Boolean vectors. It was proposed in the 1980s by P. Kanerva [2] and has successfully been used before for vision-based robot navigation [3], [4]. Simple vision-based methods, such as implemented by Matsumoto [5], although sufficient for many environments, in monotonous environments such as corridors may present a large number of errors. Cristóforis *et. al* [6] used a similar approach, of "teach and follow" navigation based on monocular vision. On their method the teaching process is simplified when a detectable path is available. The path is determined from landmark features using the SURF method. During the learning phase the robot is guided to map the environment. The map is described as a set of segments enriched with automatically detected features. To navigate autonomously, the robot must start at the beginning of a known segment. During the autonomous navigation phase, the robot retrieves relevant landmarks from the map and estimates their position in the current view. Based on the estimated displacement, the robot adjusts its heading.

[*]ESTGOH, Polytechnic Institute of Coimbra, Portugal. E-mail: mmendes@estgoh.ipc.pt.

[†]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.

A. Cicerone robots

Rhino was perhaps the first cicerone robot [7]. Rhino was built for the purpose of being a robot tour guide, operating at the Deutsches Museum Bonn, Germany. Based on the Rhino's experience, the same group built Minerva, a second generation museum tour guide robot [8]. Minerva is an autonomous robot that operates in the large area of the Smithsonian's National Museum of American History, United States of America. Minerva is equipped with cameras, laser range finders and ultrasound sensors. It is able to map and navigate through large areas, even if they are crowded with visitors, relying on previously learnt maps of the building views and ceiling. Minerva interacts with people and offers tours through the exhibitions, through a simple interface which mimics some emotional states like happiness or frustration. In Rhino, user interface was less sophisticated. As for mapping and localisation, Rhino relied on a manually derived map, while Minerva has the ability to learn maps from scratch. Both robots provide remote web interfaces, though Rhino offers only a limited set of 13 locations for the remote user, while Minerva's location is arbitrary.

Indigo is another cicerone robot proposal, operating at the Cultural Centre Hellenic Cosmos, in Greece. Indigo intends to mimic human behaviour in a biologically inspired way [9]. Language generation and motion follow human models. The robot tracks human faces and movements using images and laser range finders. Other service robots include REEM [10] and Care-O-bot [11].

B. Vision-based navigation

For most human beings, vision provides 80% of the sensory inputs [12]. The human eyes provide a wealth of information which the human brain processes quickly and effectively, in a *natural* way. Therefore, vision-based methods for robot navigation are appealing for being biologically inspired.

Modern computer vision hardware and software can achieve impressive performances, unthinkable decades ago. Simple vision-based methods for robot navigation, such as implemented by Matsumoto [13], are sufficient for many environments. They consist in simply storing views and retrieve them later for robot localisation. In monotonous environments such as corridors the method may present a large number of errors. When the database of images becomes large it is also difficult to process in real time. This method, while relying on visual information, does not try to mimic the workings of the human brain for image processing.

Robust navigation will also require an algorithm to avoid collisions. Collision avoidance relies on detecting objects and estimating distance between the robot and the object,

planning a collision-safe path or halting the robot in time to prevent the robot from hitting the obstacle. A popular approach is based on potential field models, where the obstacles exert repelling forces on the robot, keeping it away at safe distance [14]. Distance to objects is measured using laser range finders, ultrasounds, stereo or even monocular vision [15]. ASSIS relies on both ultrasound and infrared (IR) sensors.

C. ASSIS' memory-based navigation

ASSIS is a prototype of an assistant robot that uses vision for localisation and navigation. The robot learns new paths during a supervised learning stage. While learning, the robot captures and stores views of the surrounding environment, and stores them in an SDM, with some odometric and additional information.

The SDM is a type of associative memory suitable to work with high-dimensional boolean vectors. It was proposed by Pentti Kanerva in the 1980s [2] and has successfully been used before for vision-based robot navigation [4], [16]. It has also been implemented in parallel using a Graphics Processing Unit [17].

In autonomous navigation, ASSIS captures updated views and uses the memory to search for the closest image, using the image's additional information as basis for localization and navigation. Memory search is performed in parallel in a Graphics Processing Unit (GPU).

Still during the autonomous navigation mode, the environment is scanned using sonar and infra-red sensors (IR). If obstacles are detected in the robot's path, an obstacle-avoidance algorithm takes control of navigation until the obstacle is overcome. In straight paths, the algorithm creates a stack of odometric data that is used afterwards to return to the original heading, when vision-based navigation is resumed.

Section II briefly describes the SDM. Section III presents the key features of the experimental platform used. The principles for vision–based navigation are explained in Section IV. Section V describes two of the navigation algorithms implemented in the robot. It also presents the results of the tests performed with those algorithms. In Section VI, the obstacle avoidance algorithms are described, along with the validation tests. The results are discussed in Section VII. Conclusions and future work are presented in Section VIII.

II. Sparse Distributed Memory

The properties of the SDM are inherited from the properties of high-dimensional binary spaces, as originally described by P. Kanerva [2]. Kanerva proves that high-dimensional binary spaces exhibit properties in many aspects related to those of the human brain, such as naturally learning (one-short learning, reinforcement of an idea), naturally forgetting over time, ability to work with incomplete information and large tolerance to noisy data.

A. Original SDM model

Fig. 1 shows a minimalist example of the original SDM model. The main structures are an array of addresses and an array of data counters. The memory is sparse, in the sense that it contains only a minuscule fraction of the locations of

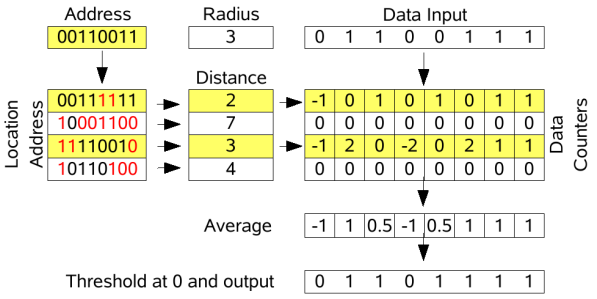

Fig. 1. Diagram of an SDM, according to the original model, showing an array of bit counters to store data and an array of addresses.

the addressable space. The locations which physically exist are called hard locations. Each input address activates all the hard locations which are within a predefined access radius (3 bits in the example). The distance between the input address and each SDM location is computed using the Hamming distance, which is the number of bits in which two binary numbers are different. A quick way to compute the Hamming distance is to count the number of ones resulting from an exclusive OR operation, as represented in Equation 1, where x_i is the i^{th} bit of vector x, and y_i is the i^{th} bit of vector y.

$$hd(x, y) = \sum_{i=0}^{i=n} x_i \oplus y_i \tag{1}$$

Data are stored in the bit counters. Each location contains one bit counter for each bit of the input datum. To write a datum in the memory, the bit counters of the selected locations will be decremented where the input datum is zero and incremented where the input datum is one. Reading is performed by sampling the active locations. The average of the values of the bit counters is computed column-wise for each bit, and if the value is above a given threshold, a one is retrieved. Otherwise, a zero is retrieved. Therefore, the retrieved vector may not be exactly equal to the stored vector, but Kanerva proves that most of the times it is, based on the statistical properties of boolean spaces.

B. Simplified arithmetic SDM model

The original SDM model, using bit counters, has some drawbacks which have been studied and lead to further improvements. One problem is that it has a low storage rate, of about 0.1 data bits per bit of physical memory. Another problem is that the counters slow down the system in real time, specially if they are implemented in conventional serial processors. Yet another problem is that data encoded using the natural binary code are sensitive to the positional value of the bits [18], and that negatively affects the performance of the system. In order to overcome some of the drawbacks described, other SDM models have been proposed [18], [19], [20]. Fig. 2 shows a model based on Ratitch et al.'s approach [19], which groups data bits as integers and uses the sum of absolute differences instead of the Hamming distance to compute the distance between an input address and location addresses in the memory. That has been the model used in the present work. It was named "arithmetic SDM," and its performance has been superior to the original model in vision-based robot navigation [4].

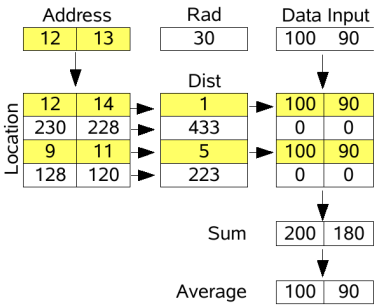

Fig. 2. Arithmetic SDM model, which uses decimal integers instead of bit counters.

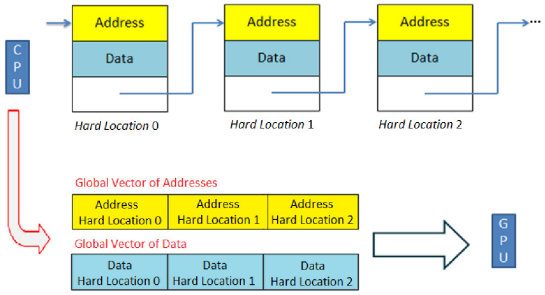

Fig. 3. Data structures of the SDM implemented in the CPU and in the GPU.

C. Parallel SDM implementation

The arithmetic SDM model, as described in II-B, was first implemented in a CPU, using linked lists as depicted in the upper part of Fig. 3. Each element of the linked list contains a memory item, represented as a pair of data and corresponding address, along with the pointer for the next element of the list. The list grows when new items (new paths) are stored into the memory, thus increasing the search time required to find an element in the memory.

As the volume of data required to process in real time increased, it became clear that the system could benefit from a parallel implementation of the search procedure. Part of the SDM was then implemented in parallel, in a GPU, using CUDA architecture, as shown in the lower part of Fig. 3.

During the learning stage of the robot, the linked list is built, using only the CPU and central RAM memory. After the learning step is complete, the list contents are copied to an array of addresses and to an array of data in the GPU's memory. Later, when necessary to retrieve any information from the SDM, multiple GPU kernels are launched in parallel to check all memory locations and get a quick prediction. The parallel implementation is described in more detail in [17], where the results also show significant improvements in the search speed compared to the original serial implementation.

III. ASSIS Experimental Platform

ASSIS is based on an X80Pro robot, controlled by a laptop running all the software and providing a web based user interface.

A. Hardware

The robot used is an X80Pro, as shown in Fig. 4. It is a differential drive vehicle equipped with two driven wheels, each with a DC motor, and a rear swivel caster wheel, for stability. The robot has a built-in digital video camera and an integrated WiFi communications module using 802.11g protocol. It also offers the possibility of communication and control by USB or serial ports. For object detection, it has six ultrasound sensors and seven infra-red sensors with a sensing range of respectively 2.55 m and 0.80 m. In the present implementation only ultrasound sensors were used for collision avoidance, due to the larger area covered.

Fig. 4. ASSIS cicerone robot carrying a small load on its tray.

The robot is controlled in real time from a laptop with a 2.40 GHz Intel Core i7 processor, 6 Gb RAM and a NVIDIA GPU with 2 Gb of memory and 96 CUDA cores.

The robot was enhanced with a light wooden body which supports a tablet for the user interface. This wooden structure also confers on it the possibility of carrying small objects over a wooden tray.

B. Control Software

Fig. 5 shows the interactions between the different software modules developed and the robot. The laptop is carried on board and connected to the robot through a serial port. The serial port was chosen because it provides faster and more reliable control than the integrated WiFi communication, which relies on network connectivity and quality of the wireless signal.

The "SDM" module is where the navigation information is stored and processed, so it is the most important module.

The "path learning" module is used for supervised learning of the paths. It controls the robot during supervised learning and feeds the relevant information to the SDM module.

The "autonomous run" module takes control of the robot during the autonomous run mode. It is in this module that the navigation algorithms described in Section V are deployed. They rely on sensory information provided from the motor and sensor control module, and on memories retrieved from the SDM to make decisions. This autonomous run module requires the obstacle avoidance module, which is activated when an obstacle is sensed in close proximity to the robot.

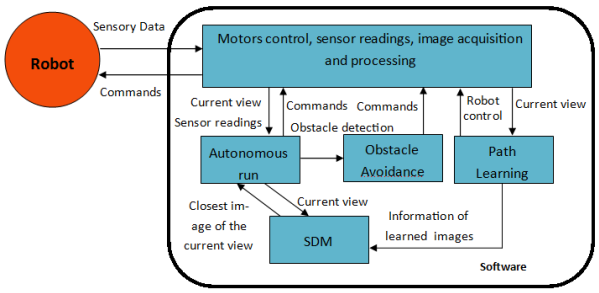

Fig. 5. Interactions between software modules and the robot.

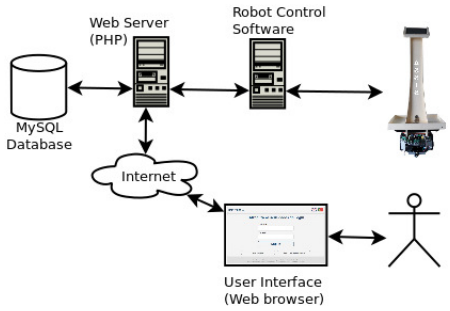

Fig. 6. Interactions between all system components and the user.

C. User interaction

Fig. 6 illustrates the overall system. The control software described in Section III-B runs on a laptop which is carried by the robot itself. That control software provides a socket interface, from where the user interface application can take control. The user interface is web based. It is a PHP application, which stores state, mission and users' information on a MySQL database.

Since the user interface is web based, it can be accessed remotely using any common web browser. Nonetheless, for direct control of the robot, the wooden structure carries a standard tablet running a web browser. The interface application provides different profiles, for the remote and local users, as well as session control, so that one user can only take control of the robot after the previous mission was completed. The only exception is for administrators, who can unlock the robot and take control at any moment.

The interface application allows the administrators to teach and manage different paths and different missions. Standard users can store and manage their missions. Anonymous users will be able to choose destinations and execute missions locally using the tablet carried by the robot.

IV. Vision-based navigation

ASSIS' localization and navigation are based on visual memories, which are stored into the SDM during the supervised learning stage and used later in the autonomous run mode.

A. Supervised learning

In the supervised learning stage, the user drives the robot along a path, issuing steering and moving commands using the remote user interface. During this process, the robot acquires pictures in BMP format with 176×144 resolution, at about 10 frames per second, using its in-built digital video camera. The images are then converted to Portable Grey Map (PGM) format, which is more appropriate to store and manipulate in the SDM. All the captured images are saved in the disk, along with additional tagging data, and later stored into the SDM.

Each path has a unique sequence number and is described by a sequence of views, where each view is also assigned a unique view number. Hence, the images are used as addresses for the SDM and the data vectors stored into the SDM are in the following format, where d_i is data vector i:

$$d_i =< S_j, im_i, vr_i, vl_i > \quad (2)$$

The sequence number is S_j, and the number of the image in S_j is im_i. The angular velocity of the right and left wheels is, respectively, vr_i and vl_i. This information is sufficient for describing the robot's localisation and motion. To localise and navigate the robot during the autonomous run, an image im will retrieve the position of the robot in the sequence S_j, and also the velocities $< vr_i, vl_i >$ of the wheels when the robot was at that point during the learning stage.

B. Autonomous running

When the robot starts the autonomous run mode, it will capture an image and use it for localisation. It will query the SDM to obtain the most similar image in memory. If the distance of the current view to the most similar view stored in the SDM is less than the SDM access radius the place is recognised and the robot assumes it is in a known position of sequence S_j. The robot will then search for all the goal points which can be reached from its current position and show a list of possible goals to the user.

After a destination is chosen, the robot starts the process of navigating towards the goal point. During navigation, the robot consecutively uses its current view as address to the SDM and thus retrieve the associated data vector d_i as described in Equation 2. From the d_i the robot infers its probable location and the most probably correct motion commands $< vr_i, vl_i >$. Thus, in the autonomous run mode the robot continually locates itself in the sequence of images previously learnt and tries to mimic the same motor commands that lead it to the goal during the supervised learning stage.

In order to reduce drifts and errors in the trajectory and orientation of the robot, the images are improved and the navigation process is monitored as explained below.

C. Image filtering and pre-processing

To improve the tolerance of the system to illumination changes, all images are equalized once they are acquired from the camera. That leads to a significant performance improvement under different illumination conditions [4].

To decrease memory requirements, improve speed and minimize chances of confusion between stored images, only images which are considered relevant are selected to be stored into the disk and loaded into the SDM during the learning stage. An image is considered irrelevant if the difference to the previous stored image, computed using the SDM's similarity calculation method, is less than the SDM's access radius. This procedure leads to filtering many images

of corridors and other long monotonous scenarios, without impacting the performance of the system. In practice, during the autonomous run mode the robot will be guided by some images for a longer period of time, thus saving significant amounts of memory. This process automatically adjusts the sampling rate as needed for each scenario.

D. Lateral drift correction

Since the robot is following the paths based essentially on following the same commands executed during the learning stage, small drifts inevitably occur. In order to prevent those drifts from accumulating a large error, a correction algorithm was also implemented, following Matsumoto *et al.*'s approach [8], as described in more detail in [2]. Once an image has been predicted, a block-matching method is used to determine the horizontal displacement between the robot's current view and its view during the learning stage. If a difference is found, the robot's heading is adjusted by proportionally decreasing wheel velocities vr or vl, in order to compensate the lateral drift.

Lateral drift correction is only turned on during straight segments of the paths. It is turned off during curves to prevent long correction loops, as well as during obstacle avoidance deviations.

E. Sequence disambiguation

During the autonomous run mode, the data associated with each image that is retrieved from the memory is checked to determine if the image belongs to the sequence (path) that is being followed. Under normal circumstances, the robot is not expected to skip from one sequence to another. Nonetheless, under exceptional circumstances it may happen that the robot is actually moved from one path to another, for unknown reasons. Possible reasons include slippage, manual displacement, mismatch of the original location, among many others. Such problem is commonly known as the "kidnapped robot," for it is like the robot is kidnapped from one point and abandoned at another point, which can be known or unknown.

To deal with the "kidnapped robot" problem and similar difficulties, ASSIS uses a short term memory of n entries (50 was used in the experiments). This short term memory is used to store up to n of the last sequence number S_j that were retrieved from the memory. If ASSIS is following sequence S_j, then S_j should always be the most popular in the short term memory. If an image from sequence S_k is retrieved, it is ignored, and another image is retrieved from the SDM, narrowing the search to just entries of sequence S_j. Nonetheless, S_k is still pushed onto the short term memory, and if at some point S_k becomes more popular in the short term memory than S_j, the robot's probable location will be updated to S_k. This disambiguation method showed to filter out many spurious predictions while still solving kidnapped robot-like problems.

F. Use of a sliding window

When the robot is following a path, it is also expected to retrieve images only within a limited range. For example, if it is at the middle of a long path, it is not expected to get back to the beginning or right to the end of the path. Therefore, the search space can be truncated to a moving "sliding window," within the sequence that is being followed. For a sequence containing a total of z images, using a sliding window of width w, the search space for the SDM at image im_k is limited to images with image number in the interval $\{max(0, k - \frac{w}{2}), min(k + \frac{w}{2})\}$. The sliding window in the SDM is implemented by truncating the search space, a method similar to Jaeckel's selected coordinate design [21]. In Jaeckel's method, coordinates which are de1emed irrelevant to the final result are disregarded in the process of calculating the distance between the input address and each memory item, so computation can be many times faster. In the present implementation, however, the selected coordinates are used just to select a subset of the whole space. The subset is then used for computing the distance using all the coordinates.

The sliding window may prevent the robot from solving the kidnapped robot problem. To overcome the limitation, an all-memory search is performed first and the short-term memory retains whether the last n images were predicted within the sliding window or not, as described in Section IV-E. This means that the sliding window actually does not decrease the total search time, since an all-memory search is still required in order to solve the kidnapped robot problem. The sliding window, however, greatly reduces the number of momentary localisation errors (MLE). A momentary localisation error is counted when the robot retrieves from the memory a wrong image, such as an image from a wrong sequence or from the wrong place in the same sequence. When a limited number of MLE occur the robot does not get lost, due to use of the sliding window and the sequence disambiguation procedures.

V. Comparison of navigation algorithms

Different navigation algorithms were implemented and tested. The two most relevant of them are described in the following subsections: the simplest and the most robust.

A. Basic algorithm

The first navigation algorithm is called "basic," for it performs just the simplest search and navigation tasks, as well as a very basic filtering technique to filter out possibly wrong predictions.

Image search, for robot localisation, is performed in all the memory. Detection of possibly wrong predictions (MLEs in the same sequence) is based on the number of the image, balanced by the total size of the sequence. If the distance between image im_t, predicted at time t, and image $im_{t\pm1}$, predicted at time $t \pm 1$, is more than $\frac{1}{3}z$, for a path described by z images, $im_{t\pm1}$ is ignored and the robot continues performing the same motion it was doing before the prediction. The fraction $\frac{1}{3}z$ was empirically found for the basic algorithm.

The performance of this basic algorithm was tested indoors in the corridors of the Institute of Systems and Robotics of the University of Coimbra, Portugal. The robot was taught a path about 22 meters long, from a laboratory to an office, and then instructed to follow that path 5 times. Fig. 7 shows the sequence numbers of the images that were taught and

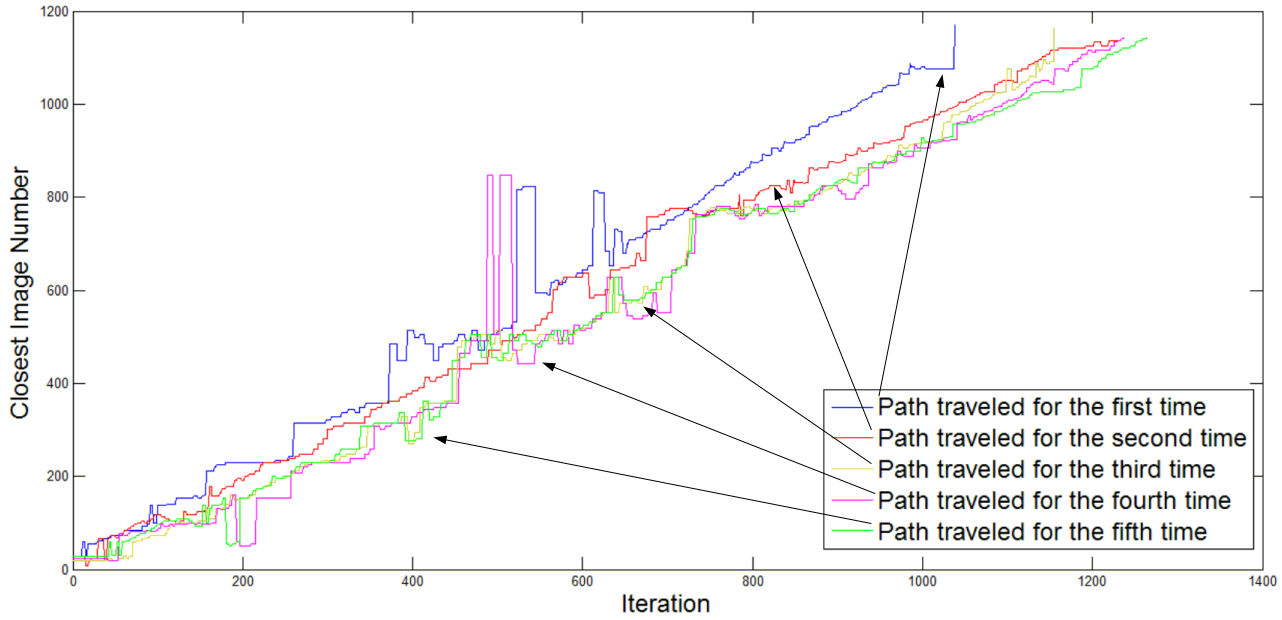

Fig. 7. Image sequence numbers of the images predicted by the SDM following the path from a laboratory to an office (approx. 22 m). The graph shows the number of the images that are retrieved from the memory as the robot progresses towards the goal.

retrieved each time. The robot never got lost and always reached a point very close to the goal point.

In a second test, the robot was taught a path about 47 meters long. The results are shown in Fig. 8. As the figure shows, the robot was not able to reach the goal, it got lost at about the 810^{th} prediction in the first run and at the 520^{th} in the second run.

The results obtained in the second test show that the basic algorithm is not robust enough, at least for navigating in long and monotonous environments such as corridors in the interior of large buildings.

B. Improved autonomous navigation with sliding window

Many of the prediction errors happen where the images are poor in patterns and there are many similar views in the same path or other paths also stored in the memory. Corridors, for example, are very monotonous and thus prone to prediction errors. The use of a sliding window to narrow the acceptable predictions improves the results, but it is not enough. The improved sliding window algorithm with the other principles described in Section IV-F worked in all situations that it was tested.

Fig. 9 shows the result obtained with this navigation algorithm when the robot was made to follow the same path used for Fig. 8 (second test path). A sliding window 40 images wide was used. As the graph shows, the sliding window filters out all the spurious predictions which could otherwise compromise the ability of the robot to reach the goal. Close to iterations number 200 and 1300, some of the most similar images retrieved from the memory are out of the sliding window, but those images were not used for retrieving control information (because of them being out of the sliding window).

C. Patrol mode

To make the robot able to carry out patrol missions, a special navigation mode was developed, which is called "the patrol mode."

The patrol mode uses a closed path in which the end overlaps with the beginning. During the learning stage, the robot recognizes that the path ends at a point where the views coincide with those of the beginning of the same path. If the patrol mode is turned on, then the robot is able to jump directly from the end of a sequence to its beginning, thus following the same path continuously, until stopped by the operator. Other stopping criteria can be applied, such as a determined number of loops or start and stop times. Fig. 10 shows an example of the predictions made while following a path continuously twice. The path is 55 meters long and described by about 480 images in the SDM.

VI. Obstacle Avoidance

In order for the robot to navigate in real environments, it is necessary that the navigation process in autonomous mode is robust enough to detect and avoid possible obstacles in the way. Two algorithms were implemented, one for obstacles which appear in straight line paths and another for obstacles that appear in curves.

A. Obstacles in straight paths

In the autonomous navigation mode, the front sonar sensors are activated. All objects that are detected at less than 1 m from the robot are considered obstacles and trigger the obstacle avoidance algorithm. A median of 3 filter was implemented to filter out possible outliers in the sonar readings. When an obstacle is detected, the robot suspends memory-based navigation and changes direction to the side of the obstacle that seems more free. The robot chooses the side by reading the two lateral front sonar sensors. The side of the sensor that gives the higher distance to an object is considered the best side to go. If both sensors read less than 50 cm the robot stops to guarantee its safety. Providing the robot senses enough space, it starts circumventing the

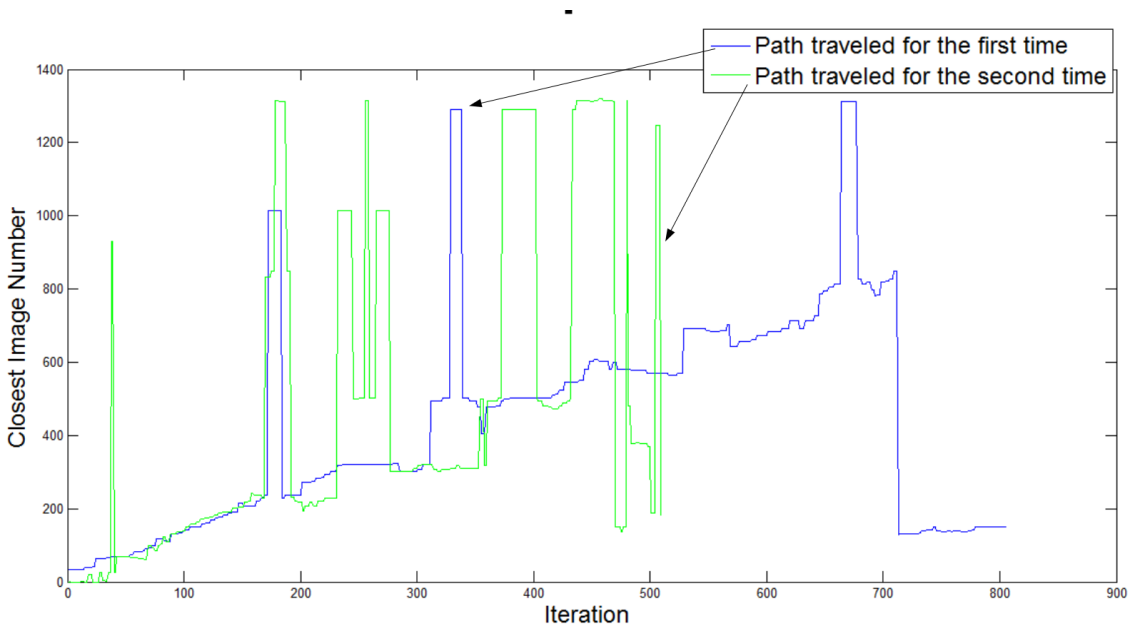

Fig. 8. Images predicted by the SDM following the second test path (approx. 47 m) using the basic algorithm.

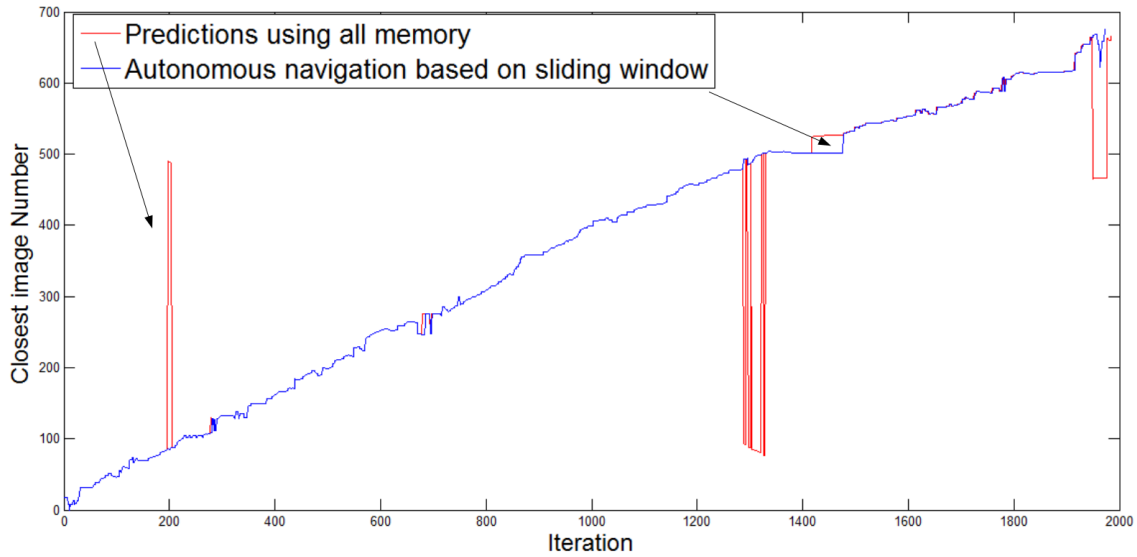

Fig. 9. Images predicted by the SDM following the second test path using the sliding window algorithm.

obstacle by the safest side. While circumventing, it logs the wheel movements in a stack. When the obstacle is no longer detected by the sensors, the wheel movements logged are then performed in reverse order emptying the stack. This process returns the robot to its original heading. When the stack is emptied, the robot tries to localize itself again based on visual memories and resume memory-based navigation. If the robot cannot localise itself after the stack is empty, then it assumes it is lost and navigation is stopped. In the future this behaviour may be improved to an active search method.

Fig. 11 shows an example of a straight path previously taught and later followed with two obstacles placed in that path. After avoiding collision with the first obstacle, the robot resumes memory-based navigation maintaining its original heading, performing lateral drift correction for a while. It then detects and avoids the second obstacle and later resumes memory-based navigation maintaining its original heading. Note that in the figure, because the lines were drawn using a pen attached to the rear of the robot, when the robot turns to the left it draws an arc of a line to the right.

B. Obstacles in curves

The stack method described in the previous subsection works correctly if the obstacle is detected when the robot is navigating in a straight line. If the obstacle is detected while the robot is performing a curve, that method may not work, because the expected robot's heading after circumventing the obstacle cannot be determined in advance with a high degree of certainty. If an obstacle is detected when the robot is changing its heading, then the stack method is not used. The robot still circumvents the obstacle choosing the clearer side of the obstacle. But in that case it only keeps record of which side was chosen to circumvent the obstacle and what was the previous heading. Then, when the obstacle is no longer

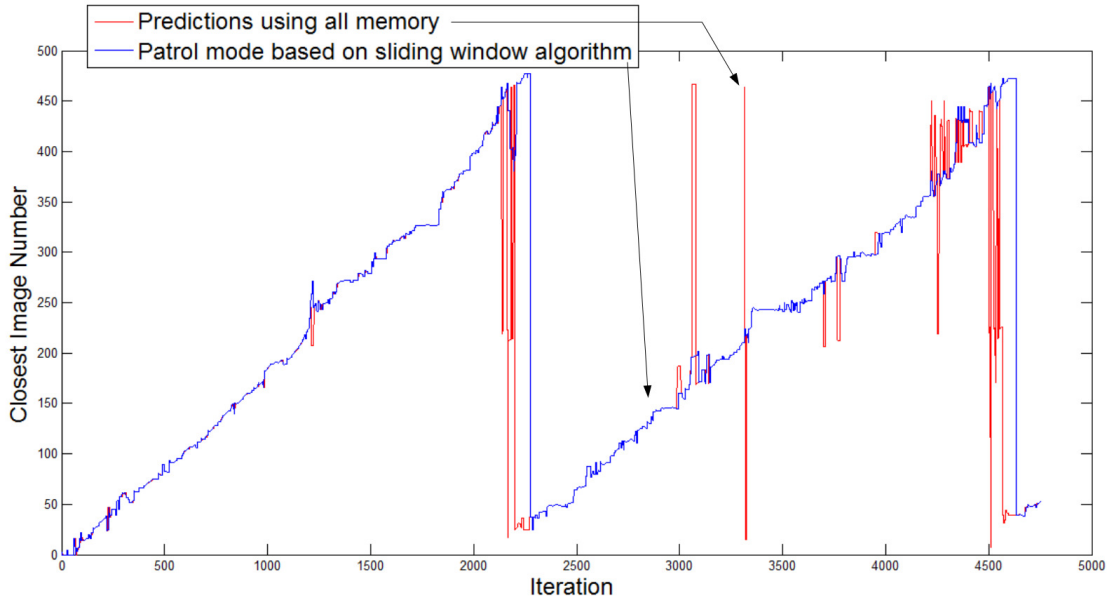

Fig. 10. Images predicted by the SDM following a path, using the sliding window in the patrol mode. Close to image 480 the robot gets back to the beginning of the sequence.

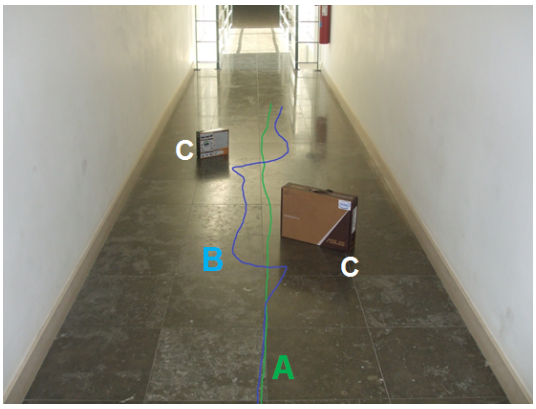

Fig. 11. Examples of obstacle avoidance in a straight path (the lines were drawn using a pen attached to the rear of the robot, so they actually mark the motion of its rear, not its centre of mass). A) Path followed without obstacles. B) Path followed with obstacles. C) Obstacles.

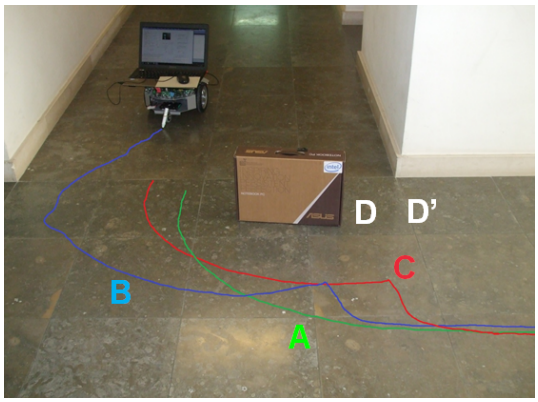

Fig. 12. Examples of obstacle avoidance in a curve. A) Path taught. B) Path followed by the robot avoiding the obstacle D and the left wall corner. C) Path followed by the robot when the obstacle was positioned at D'.

detected, the robot uses vision to localise itself and fine tune the drift using the algorithm described in Section IV-D. In curves, the probability of confusion of images is not very high, even if the images are captured at different distances. The heading of the camera often has a more important impact on the image than the distance to the objects. Therefore, after the robot has circumvented the obstacle it will have a very high probability of being close to the correct path and still at a point where it will be able to localise itself and determine the right direction to follow.

Fig. 12 shows examples where the robot avoided obstacles placed in a curve. In path B (blue) the wall corner at the left is also detected as an obstacle, hence there is actually a second heading adjustment. The image shows the robot was still able to localise itself after the obstacle and proceed in the right direction, effectively getting back to the right path.

VII. Results and Discussion

Fig. 13 illustrates the robot following a mission. The green arrows indicate the path taught during the supervised learning step. The red arrows mark the path chosen by the robot during the autonomous run mode. The arrows were manually placed behind the robot while it was moving along the path and later enhanced in the picture.

The figure shows only a small difference between the paths. In general it is possible to affirm that the robot achieves the goal point with a minimum drift. The drift is larger during curves, where a small change in the heading can cause a difference of a few centimetres in the actual path. In straight paths, the drift is corrected by the drift-correcting algorithm, which is not used during curves.

VIII. Conclusion

A method of navigating a robot, using visual and odometric information stored into an SDM, has been proposed. The SDM is implemented in parallel in a GPU, for better performance. Assis, the robot, uses supervised learning to learn new paths for later autonomous missions. A sliding window is used to segment the search space and improve the performance. A novel view-based obstacle-avoidance

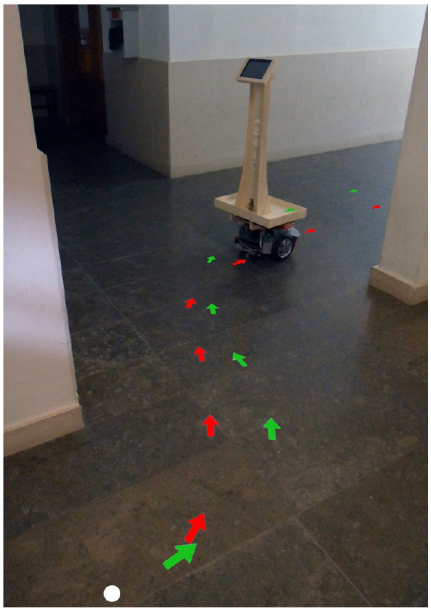

Fig. 13. ASSIS cicerone robot during a mission. Green arrows indicate learned trajectory and red arrows indicate the path followed.

algorithm was also described. The use of a stack to store the robot's motions when circumventing obstacles showed good performance in straight paths. During curves, experimental evidence shows that, after circumventing the obstacle, the robot adjusts its heading faster using memory-based navigation for localization and the drift-correction algorithm for heading adjustment. Experimental results show good performance of the system for indoors navigation. The robot is able to perform autonomously tasks such as surveillance, guiding people inside a building or carrying objects to previously taught places. Future work will include development of higher level modules to implement advanced surveillance and cicerone behaviours. In the patrol mode the robot must detect changes in the environment which should trigger an alert, such as intruder detection. In cicerone mode, the robot must be able to interact with people.

Acknowledgment

The authors acknowledge Fundação para a Ciência e a Tecnologia (FCT) and COMPETE 2020 program for the financial support to the project UID-EEA-00048-2013.

References

[1] M. Mendes, A. P. Coimbra, and M. M. Crisóstomo, "Circumventing obstacles for visual robot navigation using a stack of odometric data," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2017*, London, U.K., 5-7 July 2017, pp. 172–177.

[2] P. Kanerva, *Sparse Distributed Memory*. Cambridge: MIT Press, 1988.

[3] R. P. N. Rao and D. H. Ballard, "Object indexing using an iconic sparse distributed memory," The University of Rochester, Computer Science Department, Rochester, New York, Tech. Rep. 559, July 1995.

[4] M. Mendes, A. P. Coimbra, and M. M. Crisóstomo, "Robot navigation based on view sequences stored in a sparse distributed memory," *Robotica*, July 2011.

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

[6] P. D. Cristóforis, M. Nitsche, T. Krajník, T. Pirea, and M. Mejail, "Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments," *Pattern Recognition Letters*, no. 53, 2015.

[7] W. Burgard, D. Fox, G. Lakemeyer, D. Haehnel, D. Schulz, W. Steiner, S. Thrun, and A. Cremers, "Real robots for the real world - the rhino museum tour-guide project," in *Proceedings of the 1998 AAAI Spring Symposium*, 1998.

[8] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Minerva: A second-generation museum tour-guide robot," in *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'99*, 1999.

[9] S. Konstantopoulos, I. Androutsopoulos, H. Baltzakis, V. Karkaletsis, C. Matheson, A. Tegos, and P. Trahanias, "Indigo: Interaction with personality and dialogue enabled robots," 2008.

[10] R. Tellez, F. Ferro, S. Garcia, E. Gomez, E. Jorge, D. Mora, D. Pinyol, J. Oliver, O. Torres, J. Velazquez *et al.*, "Reem-b: An autonomous lightweight human-size humanoid robot," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE, 2008, pp. 462–468.

[11] R. Kittmann, T. Fröhlich, J. Schäfer, U. Reiser, F. Weißhardt, and A. Haug, "Let me introduce myself: I am care-o-bot 4, a gentleman robot," *Mensch und computer 2015–proceedings*, 2015.

[12] S. Johnson, *Mind wide open*. New York: Scribner, 2004.

[13] Y. Matsumoto, M. Inaba, and H. Inoue, "View-based approach to robot navigation," in *Proc. of IEEE/RSJ IROS 2000*, 2000.

[14] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, March 1986.

[15] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 593–600.

[16] R. Rao and O. Fuentes, "Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory," *Machine Learning*, vol. 31, no. 1-3, pp. 87–113, April 1998.

[17] A. Rodrigues, A. Brandão, M. Mendes, A. P. Coimbra, F. Barros, and M. Crisóstomo, "Parallel implementation of a sdm for vision-based robot navigation," in *13th Spanish-Portuguese Conference on Electrical Engineering (13CHLIE)*, Valência, Spain, 2013.

[18] M. Mendes, M. M. Crisóstomo, and A. P. Coimbra, "Assessing a sparse distributed memory using different encoding methods," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2009*, London, U.K., 1–3 July, 2009, pp. 37–42.

[19] B. Ratitch and D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning." in *ECML*, 2004.

[20] J. Snaider, S. Franklin, S. Strain, and E. O. George, "Integer sparse distributed memory: Analysis and results," *Neural Networks*, no. 46, pp. 144–153, 2013.

[21] L. A. Jaeckel, "An alternative design for a sparse distributed memory," Research Institute for Advanced Computer Science, NASA Ames Research Center, Tech. Rep., July 1989.