Knowledge propagation in a Distributed Omnidirectional Vision System

E. Menegatti * C. Simionato * S. Tonello * G. Cicirelli†
A. Distante† Hiroshi Ishiguro‡ E. Pagello *§

April 26, 2006

Abstract

In this paper an omnidirectional Distributed Vision System (DVS) is presented. The presented DVS is able to learn to navigate a mobile robot in its working environment without any prior knowledge about calibration parameters of the cameras or the control law of the robot (this is an important feature if we want to apply this system to existing camera networks). The DVS consists of different Vision Agents (VAs) implemented by omnidirectional cameras. The main contribution of the work is the explicit distribution of the acquired knowledge in the DVS. The aim is to develop a totally autonomous system able not only to learn control policies by on-line learning, but also to deal with a changing environment and to improve its performance during lifetime. Once an initial knowledge is acquired by one Vision Agent, this knowledge can be transferred to other Vision Agents in order to exploit what was already learned. In this paper, first we investigate how the Vision Agent learns the knowledge, then we evaluate its performance and test the knowledge propagation on three different VAs. Experiments are reported both using a system simulator and using a prototype of the Distributed Vision System in a real environment demonstrating the feasibility of the approach.

1 Introduction

In recent years, the robotics community is shifting its attention from a single agent performing its task in a structured environment to *distributed systems* where a number of agents cooperate together to achieve a common goal in unmodified human-made environments. Several projects explored the possibility to support the human and robot activity in the environment with a network of smart sensors [8, 31, 24].

*Intelligent Autonomous Systems Laboratory, Department of Information Engineering (DEI), Faculty of Engineering, University of Padua, Padova, Italy. email:emg@dei.unipd.it

†Institute of Intelligent Systems for Automation National Research Council, Bari, Italy, email:grace@ba.issia.cnr.it

‡Department of Adaptive Machine Systems, Osaka University, Suita, Osaka, 565-0871 Japan

§Institute of Biomedical Engineering of the National Research Council (ISIB-CNR), Padova , Italy

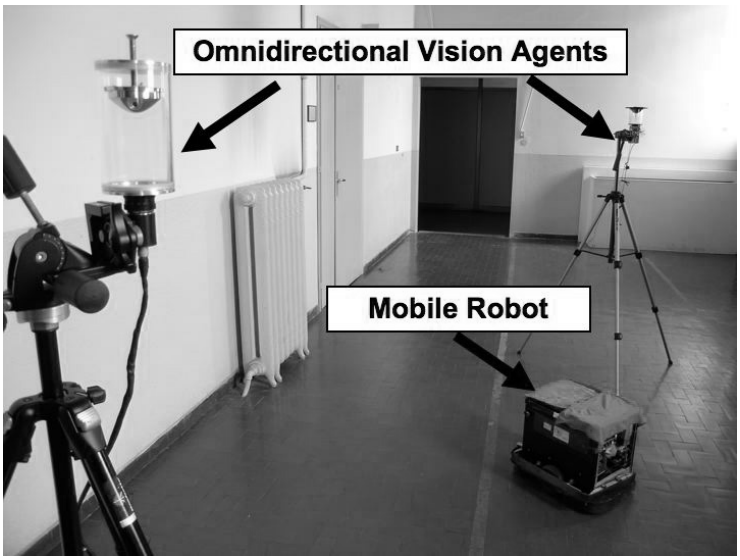


Figure 1: A picture of the whole system showing two Vision Agents and the robot used in the experiments.

In [17] we proposed to use a pre-existing network of surveillance cameras to navigate a mobile robot, i.e. to implement an intelligent infrastructure able to support robots' activities in a way similar to the ones proposed in [11, 12]. In these works, the network of cameras is named Distributed Vision System (DVS) to emphasize that the system of cameras acts as a whole at the aim of reaching the final objective: navigate the robot in the environment. In our implementation the DVS, as shown in Fig. 1, is composed of several Vision Agents (VA). Each VA is an omnidirectional vision sensor, Fig. 2a), and is able to acquire and process images, to communicate with other VAs over the network and to send the movement commands to the robot, Fig. 4.

In our approach the cameras are not calibrated and the robot does not have any sensor or processing power on board. Each VA will take the control of the robot every time it enters its field of view. The robot is just a "dummy" mobile platform whose motors are driven by the Vision Agents in the DVS. On the contrary, the classical approach to this kind of problems is to calibrate the camera in order to find the mapping between the image space and the world space and to use the control law of the robot to drive it to desired points in the 3D-world. We want to bypass this procedure, as sketched in Fig. 3, and be able to relate the image space directly to the motor space of the robot. An additional reason to deal with uncalibrated cameras, is the aim to use the system in environments in which there is a pre-installed network of surveillance cameras. These cameras usually are not calibrated. To calibrate by hand all these cameras can be tedious or even unfeasible if the number of cameras is large or if the cameras are distributed over a large space. Several researchers are exploring the issue of multiple-camera network calibration (e.g., [30] [21] [9]).

We propose a different approach. We do not want to program a robot by using

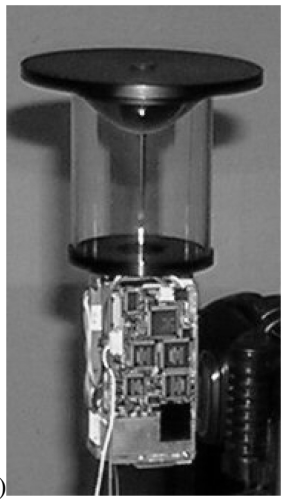
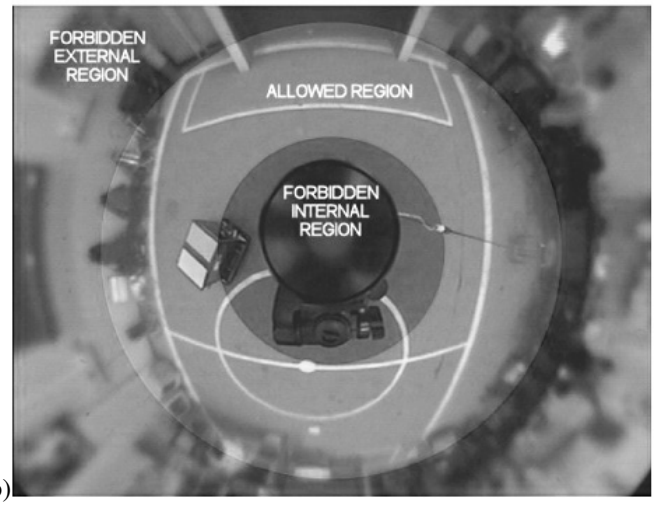
a)  b) 

Figure 2: a) One of the omnidirectional cameras used in the experiments. b) Example of an image taken by the omnidirectional camera.

the information provided by a network of cameras calibrated by hand. We propose to build a network of uncalibrated vision sensors with overlapping fields of view, able to autonomously learn to navigate a service robot in a large indoor environment composed of several connected rooms and corridors.

In [17] the same DVS is introduced, but what makes that system different from the one proposed in this work is the learning procedure. In [17] neural networks were used to learn how to control the robot. However the experimental results showed a poor generalization ability of the neural system due to an overfitting of the training data. In the current work a different learning methodology is used instead of a neural network. A totally autonomous system should be able to learn and to adapt itself to the eventual variations of the environment in order to improve its performance during its lifetime. Reinforcement Learning (RL) [29] seems a very appropriate paradigm to face this learning problem because of the possibility of a continuous and on-line learning without the need of a teacher which indicates the best association between situations and actions (*policy*).

RL has been receiving great attention by robotics researchers to solve complex real-world problems. Initially individual autonomous agents were used in many domains [1, 2, 19, 7]. In last years, instead, there has been increased interest to apply RL and, in a more general view, *Machine Learning* to problems in the area of distributed systems or multi-agent systems. Learning based approaches provide a new way of designing agents in which agents interactions do not need a priori specific definition [3]. Learning methods may provide effective, near-optimal solutions to complex planning and decision-making problems under uncertain environments. A good and exhaustive survey of cooperative multi-agent learning in a spectrum of areas, including not only RL, but a wide range of *Machine Learning* techniques, is presented in [22]. RL based ap-

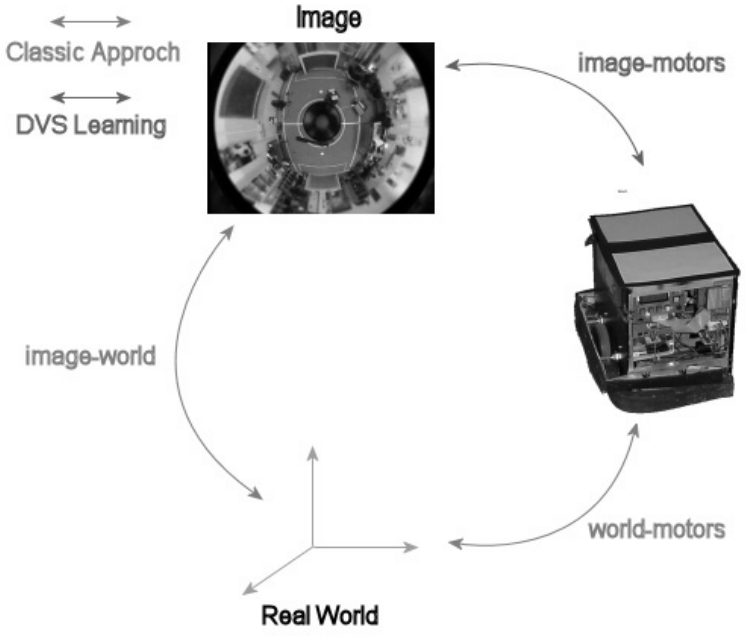


Figure 3: A sketch representing a comparison of our approach with respect to the classical approach for robot navigation by an external camera.

proaches are reported in [5, 25, 14, 16, 4, 13], where different applied problem domains and different interaction mechanisms are explored.

In this work we apply the RL paradigm to our DVS. We develop a totally autonomous system which learns to guide a robot from an initial position to a desired one (*target*). In particular one VA learns to control the robot, then the same knowledge is passed to other VAs. Those can improve the initial knowledge whilst still experiencing the world and adapting it to the eventual differences with respect to the first VA. In this way the learning time is reduced since the new VAs exploit the initial knowledge, instead of starting from scratch.

The aim of this paper is twofold: first we prove that a VA is able to learn autonomously to control the robot, second we prove it is possible to distribute the knowledge acquired by one VA to a different VA located at a different position in the environment. The knowledge is learnt by the VA by using a RL method. In particular $SARSA(\lambda)$ with *Replacing Eligibility Traces* has been used as RL method. In addition the *LEM* (*Learning by Easy Missions*) [2] technique has been applied with the aim of speeding up learning. During learning each VA has to cope with hard problems such as obstacle avoidance and path optimization. In the first stage of the work, the experimentation is performed in simulation in order to investigate how $SARSA(\lambda)$ method applies to the VA. Then the adaptability of the learned knowledge to different VAs in the environment is tested. An optimization analysis of the parameters involved in the

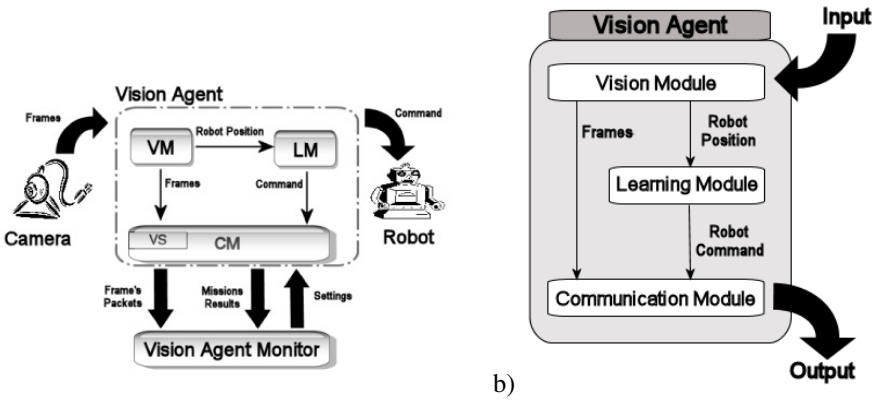
$SARSA(\lambda)$ and LEM method is also carried out. The choice of the optimal parameters is essential in the application of learning algorithms due to their relevant influence on both the learning time and the success rate. The experimental results obtained in the simulation phase prove the efficiency and the robustness of the learning system and encourage the application of the system in the real context. In fact additional experiments are conducted in the real environment by using a network composed of two omnidirectional VA and one mobile robot. The experiments showed the system is able to use the knowledge acquired in the simulation and to improve it to learn how to control the real robot in the real environment and to navigate it from a starting point to a goal point, both of which are indicated by the user by clicking on the omnidirectional image.

The rest of the paper is organized as follows. Section 2 gives an overview of the learning system and its task. Section 3 describes the simulator used for the experimentation. Section 4 gives an overview of $SARSA(\lambda)$ algorithm and a description of all its elements: state and action spaces, action selection mechanism and reward function. The LEM strategy is briefly introduced in section 5. Section 6 describes the optimization of the RL parameters. Sections 7 and 8 report the experimental results both in simulation and in the real environment respectively. Finally some conclusions and future work end the paper.

2 System Overview

Each VA in the DVS has the task of navigating the mobile robot in its own field of view, sending to the robot motion commands to move from a starting position to a target one. Fig. 4a) represents the information flow in the system. Once the image is grabbed by the camera of one of the VAs (on the left) it is processed by the VA and using the knowledge acquired by the VA motor commands for the robot are sent via the wirelass LAN. Fig. 4b) represents the internal structure of the VA module. The Vision Module processes the image seeking the robot. The robot's position is transferred to the Learning Module that outputs the motor command to be sent to the robot by the Communication Module. The Communication Module can also send the frames grabbed by the VA to the VA Monitor for debugging purposes.

The choice among the motor commands must be done considering that the robot has to reach the target position as fast as possible. This means that the VA should choose the shortest path and the proper velocities for the robot. That choice depends on the visual sensor used. Considering that the VA is an omnidirectional camera, the distances between points of the environment estimated by processing the images are different from the real ones. An accurate calibration of the system could allow a more correct estimation of distances and then a good definition for the velocities to be sent to the robot. However, we want the VA to perform its task without any calibration of the omnidirectional camera and without any knowledge about the robot control law, but learning by itself the best policy to guide safely the robot (i.e. the best commands to give to the robot for each encountered situation). In addition the VA must be able to adapt the acquired policy to changes in the environment such as variations of the camera parameters (e.g. different height between the camera and the floor). In the experimentation we test the DVS in an environment where only the cameras, which



a) b)

Figure 4: a) The conceptual representation of the software. b) The Vision Agent software module.

are fixed on tripods, represent obstacles for the robot. Then the VA has also to learn to guide the robot avoiding these obstacles.

Summarizing the VA's task is to learn:

- to guide the robot
- to choose the optimal path from the starting position to the target one
- to choose the proper velocities
- to avoid the robot exits from the field of view of the VA
- to avoid the robot colliding with the tripods (*obstacle avoidance problem*)
- to adapt the learned knowledge in case of new situations in the environment

3 The Simulator

In order to study the feasibility and the quality of the proposed learning system, we built a simulator able to reproduce the peculiarities of the VA. In particular, the relation between the pixel radial distance of a point from the center of the camera and the corresponding radial distance from the camera in the real world is evaluated. The simulated system identifies the robot by a point on the virtual image and three values (x, y, β): x, y being the pixel coordinates of that point and β its orientation. Knowing the starting position of the robot the simulator receives as input a pair of velocities $(linear, jog)$, which refer to an action to be taken by the robot, and gives as output the new position of the robot on the image. A graphical interface has not been implemented because our aim is to build a fast system and with a very low computational cost.

4 Learning the task

As previously introduced, the VA learns to guide the robot by using the RL paradigm. Then, by the direct interaction with the unknown environment, it discovers the best commands to send to the robot. In order to understand how the VA learns the correct policy, a description of the principal components for a RL agent is needed. In this section first a brief overview of $SARSA(\lambda)$, the RL algorithm implemented to learn the policy, is given. Then we show how the actions are selected during learning, how we construct the state and action spaces and finally how the reward function is defined. Finally a description of the LEM mechanism, used to reduce the learning time, is presented.

4.1 $SARSA(\lambda)$

In RL problems the learning agent attempts to acquire on-line a policy which maximizes the expected cumulative reward in the long term. A policy π is a mapping from each state s and action a to the probability of taking action a when the agent is in state s ($\pi : S \times A \rightarrow [0, 1]$, where S is the state space and A is the action space). $SARSA(\lambda)$ is an on-policy Temporal Difference (TD) control method [28]. The main peculiarity of on-policy TD methods is that, during learning, they incrementally evaluate or improve the policy that, at the same time, is used to make decisions. In other words on-policy methods update gradually the policy by using the learned estimates of the current policy. These methods differs from off-policy methods which, instead, estimate a policy whereas they use another policy for the control. A policy π is evaluated estimating an action value function $Q : S \times A \rightarrow \mathbb{R}$ which represents the expected return when the agent performs a given action in a given state. The off-policy methods directly approximate the optimal action value function, independent of the policy being followed. The on-policy methods, instead, update the action value function considering the current approximate policy.

The value function is the key element of RL methods and estimates the long-term reward. A value function can be a function of states only ($V : S \rightarrow \mathbb{R}$), but since we use $SARSA(\lambda)$ method, we consider the value function as a function of state-action pairs. These values are used to select actions so as to maximize the discounted cumulative reward $\sum_{k=0}^{\infty} \gamma^k r$ (where γ is the discount factor and r the reward value).

$SARSA(\lambda)$ updates all state-action values $Q(s, a)$ according to the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))e(s, a) \tag{1}$$

where γ is the discount factor, α is the learning rate parameter, a' is the action the robot takes in the next state s' and $e(s, a)$ is the eligibility trace of action a in state s. In this work we use *replacing eligibility traces* [26] which are updated for all s, a as follows:

$$e_t(s, a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise} \end{cases}$$

where λ is the decay parameter $0 \leq \lambda \geq 1$. An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of state or the taking of an action.

By using eligibility traces the learning system is able to give out credit or blame to the explored state-action pairs in a more efficient way.

4.2 Action Selection

During learning the choice among the possible actions, in each state, must be carried out taking into account the exploration/exploitation dilemma. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore new actions in order to learn possibly better solutions. A way to achieve that is to select actions stochastically according to their action values. In our work, actions are selected according to the ϵ-$greedy$ policy [29] that chooses most of the time the action with the maximal estimated action value. Actions with a lower action value are chosen with probability ϵ. The probability is higher at the beginning, whereas it decreases incrementally with time. This enables more exploration at the beginning and more exploitation of the acquired knowledge during advanced learning phase.

4.3 State Space Definition

The 640×480 image captured from the VA is the only source of information it has about the environment. The VA has to estimate the position and the heading of the robot in the image. An important thing to consider when an omnidirectional system is used, is that the robot can appear only on a subset of all possible pixels on the image. This subset is a circular ring with external diameter of $480pixel$ and internal diameter of $20pixel$. The internal circle contains the self-reflection of the camera. The region external to the ring is not imaging the omnidirectional mirror. Therefore the robot can be detected only inside the ring. Since the robot, once detected, is represented by a point (one pixel position) in the image, the number of the different positions it can occupy inside the ring is considerable (about 180600). Besides, for each position we must consider the different orientations of the robot. As a consequence if we think of the state of the robot as a position-orientation pair, the state space will be really large and then difficult to manage by using a RL algorithm in its discrete form.

In order to construct a reasonable state space for the VA, we have to analyze its particular task. To perform successfully its task, the VA needs to know:

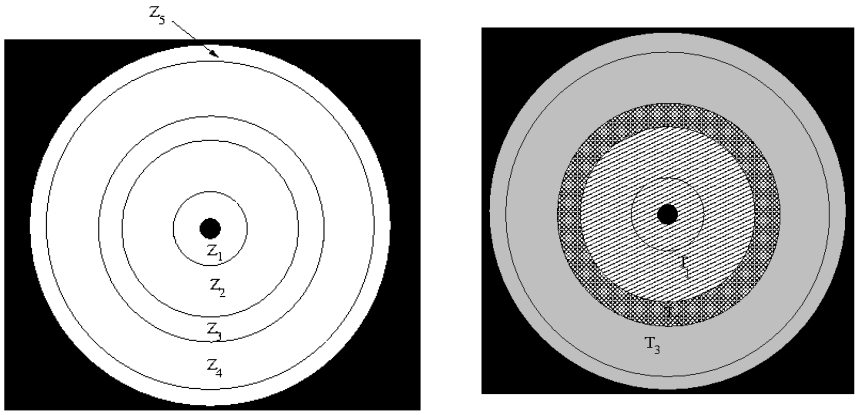
- the distance between the robot and the target position
- the distance between the robot and the camera
- the distance between the target position and the camera
- the orientation of the robot with respect to the target position
- the orientation of the robot with respect to the camera

The distance d between the robot and the target position is classified into 8 different classes considering a partition of the range interval of d as shown in table 1.

Considering the distance between the robot and the camera, the circular ring of the image, described before, can be divided into five concentric regions as shown in

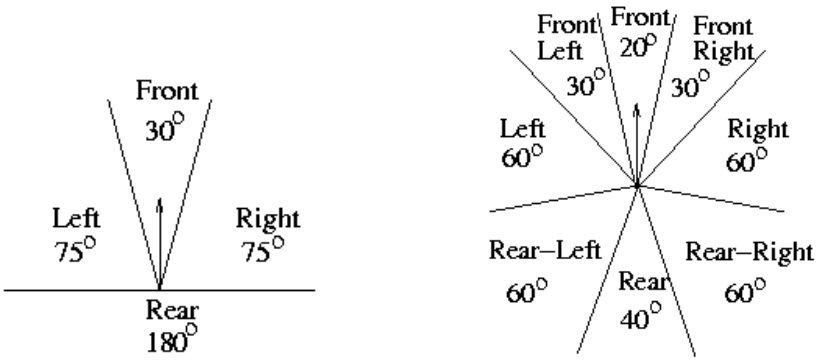
Classes	Distance $d(pixel)$
D_0	$d \leq 4$
D_4	$4 < d \leq 8$
D_8	$8 < d \leq 12$
D_{12}	$12 < d \leq 30$
D_{30}	$30 < d \leq 60$
D_{60}	$60 < d \leq 120$
D_{120}	$120 < d \leq 240$
D_{240}	$240 < d \leq 480$

Table 1: Definition of the classes for the distance d between the robot and the target position



a) b)

Figure 5: Regions of the image space: a) regions defined by the distance between the robot and the camera; b) regions defined by the distance between the target and the camera.



a) b)

Figure 6: The angular sectors which define the relative orientation between a) the robot and the camera and between b) the robot and the target position.

fig. 5. The black circle (with 20 $pixel$ radius) in the center of the image represents the reflected image of the camera. The black area around the ring is out of the field of view of the camera. The Z_1 region has a distance of 30 $pixels$ from the black circle. Z_1 and Z_5 are alarm regions because of their closeness to the black areas. If the robot is detected in those regions, the VA has to pay more attention to the movement of the robot. Z_5 is 15 $pixel$ thick. The remaining regions Z_2, Z_3 and Z_4 (with 110 $pixel$, 145 $pixel$ and 320 $pixel$ radius respectively) are defined considering that a camera with a hyperbolic mirror is used and then the resolution of a pixel changes depending on the distance from the center of the image.

Considering the distance between the target position and the camera, the ring is divided only into three regions: $T_1 = Z_1 + Z_2$, $T_2 = Z_3$ and $T_3 = Z_4 + Z_5$ and each of them is classified $free$ or $notfree$ depending on the relative position of the robot with respect to the target position. In particular each region is considered $free$ if the camera does not represent an obstacle for the robot which can reach the target position by a straight trajectory. On the contrary it is $notfree$ if the camera is between the robot and the target position.

Finally, the relative orientation between the robot and the camera and the one between the robot and the target position are determined. Figure 6a) shows the angular sectors (S_C) defined for the camera: $Left$, $Right$, $Front$ and $Rear$. They indicate if the camera is on the left side of the robot, on its right side, in front of it or behind it. For the target position the angular sectors have been augmented in order to optimize the robot behavior. These sectors (S_T) are: $Left$, $Front$-$Left$, $Front$, $Front$-$Right$, $Right$, $Right$-$Rear$, $Rear$ and $Rear$-$Left$ as shown in fig. 6b).

Concluding, the state of the VA is defined considering all the elements described above and is represented by the 5-tuple (d, Z_i, T_j, S_C, S_T) where $i = 1, ..., 5$, $j = 1, 2, 3$. The total number of states is 7680 ($= 8 \times 5 \times 3 \times 4 \times 8 \times 2$), some of them are terminal states, but a lot of them are impossible states. With impossible states we mean states which cannot happen such as for example $(D4, Z_1, T_3, ., .)$. At the end the

number of effective states does not exceed 2500.

4.4 Action Space

The robot can receive commands in terms of linear and angular velocities $(linear, jog)$, then we define an action as a pair of these two velocity values. Each velocity is additionally discretized into sub-actions: 3 for the $linear$ velocity ($stop$, $slow$, $fast$) and 5 for the jog one ($stop$, $slow\text{-}left$, $slow\text{-}right$, $fast\text{-}left$, $fast\text{-}right$). The $(stop, stop)$ action is not included, as the prime task of the VA is to move the robot in its environment. During the learning phase it may happen that performing one action does not correspond to a state transition, i.e. the passage from one state to another different state, (*State-Action Deviation Problem)*. To deal with this problem we adopt the solution proposed in [2]. Each aforesaid action is considered as a *micro-action*. The robot continues to perform one micro-action until a state transition happens. A sequence of micro-actions is defined as a *macro-action*. The last one is considered as the effective action by the learning system and only when the state changes the value function is correctly updated by using eq. 1.

4.5 Reward Function

During learning the VA receives reinforcements as it performs actions. In particular it is penalized when it guides the robot outside the field of view of the camera ($r = -60$). On the contrary it receives a positive reward if the robot reaches successfully the target position ($r = 10$). During the other state transitions the VA receives penalties or rewards proportionally both to the selected linear velocity and to the angle between the robot heading and the target position. Higher linear velocities are preferred in order to reach the target faster. Similarly the reward is high if the angle between the robot and the target is low, but decreases as this angle increases.

5 *LEM* Strategy

A drawback with RL methods is the long learning time. Several strategies have been proposed in literature to accelerate learning. In [15, 6] a splitting of the whole task into different parts or behaviors is used. In [19] the learning agent receives advice for figuring out what part of the action space deserves attention for each situation. Asada [2] introduces a new technique known as *LEM* strategy. The learning schedule is constructed such that the agent can learn in easy situations (or missions) at the early stages and in more difficult ones as learning goes on. The difficulty related to the application of *LEM* is the assumption about the knowledge of the ordered sequence of state transitions toward the target one, but the whole knowledge is not needed. In our problem due to the definition of states the agent roughly knows the ordering of state transitions then a partition of the state space can be applied. We categorize the state space S into sub-sets S_k (missions) by considering first the distance between the robot and the target, then the regions where the robot can be, then the orientation of the robot with respect to the target and finally the orientation of the robot with respect

to the camera. So the first easy mission S_1 contains those states having D_4 as distance between robot and target, Z_1 and T_1 as regions where the robot is, frontal orientation between robot and target and different orientation between robot and camera. Once the robot learns to reach the target from the states of S_1, it learns the second closest easy mission S_2, where different orientations between robot and target are considered. After that the robot learns the third closest mission S_3 and so on. The total number of missions obtained is 1120 with $|S_k| \leq 6$, considering that there are a lot of impossible states.

Asada proved that in order to shift initial situations into more difficult ones, which means shifting from sub-set S_{k-1} to S_k, the following relation should be satisfied for each $k = 1, 2, 3, \ldots$:

$$\Delta Q_t(S_k, a) = \sum_{s \in S_k} |\max_{a \in A} Q_t(s, a) - \max_{a \in A} Q_{t-\Delta t}(s, a)| < \epsilon \tag{2}$$

where Δt is a time interval which indicates the number of episodes attempted for the S_k mission and should be greater than $|S_k|$, whereas $\epsilon > 0$ is equal to:

$$\epsilon = \alpha \frac{(1-\gamma)}{\gamma} \sum_{s \in S_k} \max_{a \in A} Q(s, a) \tag{3}$$

Asada proved these relations in the case of Q-learning algorithm [32], a commonly used RL algorithm. It is straightforward to prove that the same relations hold also in our case where a different RL algorithm and a different reward function are used. By our knowledge no additional information, instead, can be found in literature about the Δt parameter.

5.1 The Δt parameter

It is considered that the Δt parameter needs particular attention since it is related to the learning time. We analyze in greater detail how that parameter affects both the learning time and the agent performance. The system simulator is used for this aim.

Figures 7 and 8 show the results obtained after running a number of simulations for different values of Δt, averaging the result every 25 simulations and with $\alpha = 0.5$, $\gamma = 0.95$, $\lambda = 0.02$ (see next section). Each simulation consists of the completion of a learning phase which starts from a zero knowledge and executes all the 1120 missions defined above. Figure 7 plots the number of episodes versus Δt. As expected, time grows as Δt increases, but observing the plot we can infer that the relation is practically linear. Figure 8 shows a plot of the success rate (i.e. the percentage of successful episodes) versus Δt. Such a percentage is evaluated performing 200 test trials after each simulation. Notice that the success rate initially grows but it stabilizes for Δt values higher than 12. This values satisfy the relation $\Delta t > |S_k|$ since in the partition that we apply to the state space $\max_k |S_k| \simeq 6$, disregarding the unreachable states. In the experimentation we choose the value $\Delta t = 14$, because it produces a more stable success rate with respect to the value 12.

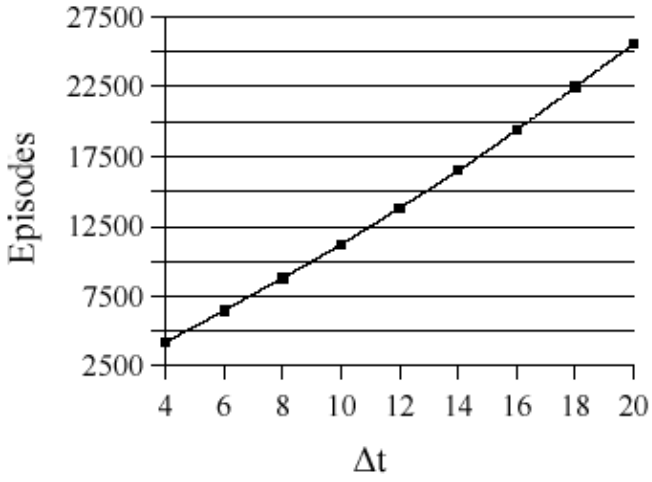


Figure 7: Learning time versus Δt parameter. Each plotted point refers to the average number of episodes evaluated on 25 simulations.

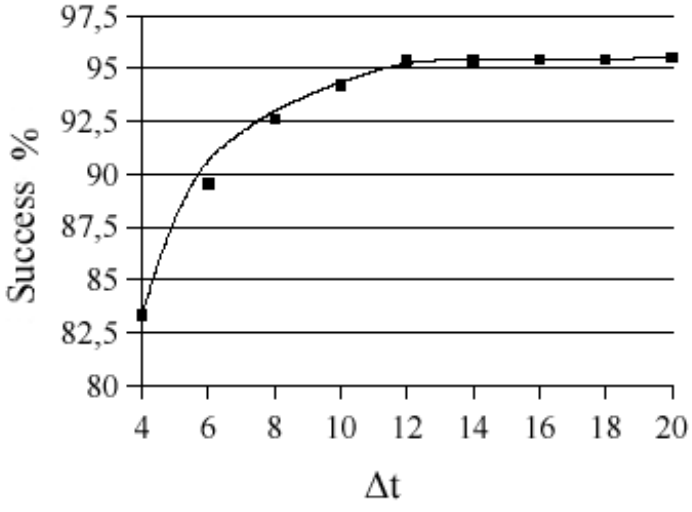


Figure 8: Percentage of success versus Δt parameter. Each plotted point refers to the average success rate evaluated on 25 simulations.

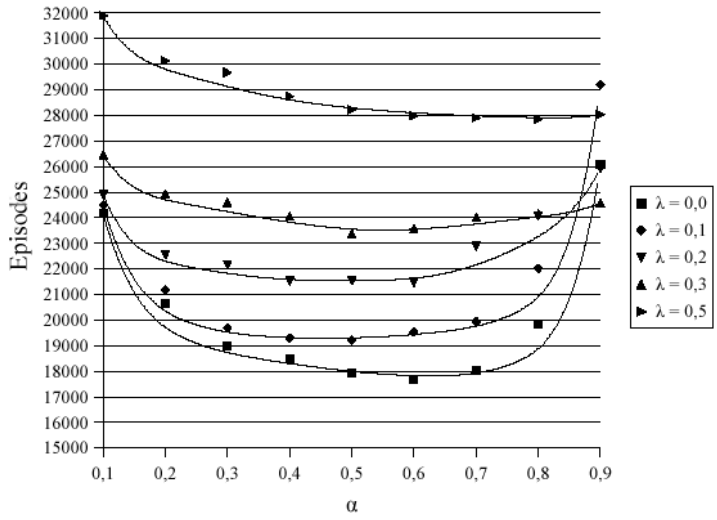


Figure 9: Learning episodes versus the α parameter for $\gamma = 0.95$ and different values of λ.

6 RL Parameter Optimization

The system simulator is very useful not only to learn and to test the policy of the VA in a virtual environment, but also to study how to choose the best values for the parameters $(\alpha, \gamma, \lambda)$ involved in $SARSA(\lambda)$ method. There is not a standard way to define those parameters to guarantee the convergence of learning. Usually they are heuristically defined and are closely connected with the particular application. In our work we, in depth, study those parameters since they influence the learning time, the success rate and the shifting parameter in the LEM strategy (see previous section). The study is carried out considering triplets of different values for α, γ and λ, performing 20 simulations for each combination of values and averaging the results. For the evaluation of the success rate 200 test trials are executed after each simulation.

Figures 9 and 10 show the results obtained varying α and λ and with a fixed value for γ (=0.95). As can be seen the value for α is close to 0.5 to have both the minimum number of learning episodes and the high percentage of success. As expected by applying $SARSA(\lambda)$ the percentage of success grows as λ increases, but also the learning time does the same.

Finally figures 11 and 12 show the results obtained varying α and γ, whereas λ has been fixed to 0.1. As the discount rate γ decreases the number of episodes and also the success rate decrease. This is closely connected to the shifting parameter (see relation (3)) of LEM which is influenced by the discount rate in a deeper way than the learning rate parameter α (see fig. 11).

Concluding, after an analysis of the results, we choose the optimal values for α, γ and λ that gave the best tradeoff between learning time and success rate. The values

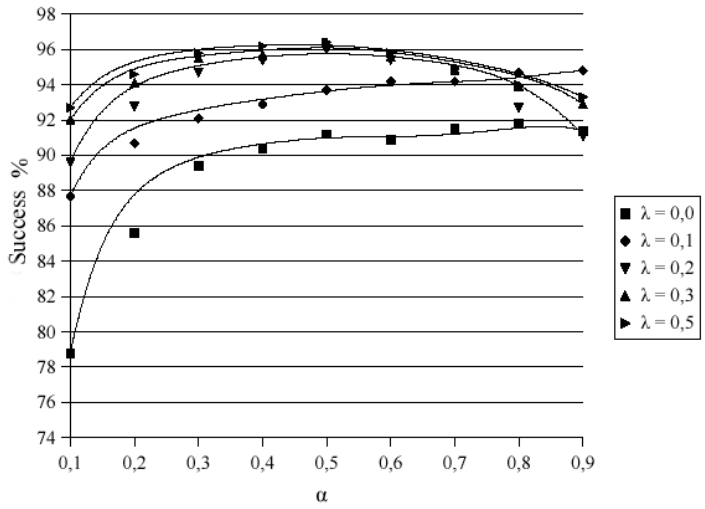


Figure 10: Percentage of success versus the α parameter for $\gamma = 0.95$ and different values of λ.

used in the experimentations are the following: $\alpha = 0.5$, $\gamma = 0.95$, $\lambda = 0.02$.

7 Experiments in simulation

The experiments in simulation consist of two stages: first the VA learns to guide the robot toward the target, then the learned policy is applied to different VAs in order to analyze its adaptability to changes of the environment. The first VA simulates an omnidirectional camera that is $1.8m$ high above the floor. A number of simulations are carried out to learn the optimal policy. Each simulation consists of two phases: a learning phase, starting from a zero knowledge and using the LEM strategy and a testing phase to examine the performance of the VA. The average number of learning episodes is 16400 ($123sec$ running on a 2GHz Pentium 4) to complete all the LEM missions. In each episode the VA starts from a state, randomly chosen among the possible ones in a mission. At each state transition it updates the action value function increasing its knowledge. If that update does not exceed a fixed threshold, the VA passes to the next mission. Each mission is explored performing at the most Δt episodes. At the end of each simulation the learned knowledge is tested running 200 additional trial episodes to examine the performance of the VA. During this testing phase the VA chooses the actions by using a greedy policy and learning is turned off. The average success rate obtained after the testing phase is 95%. Among the different policies obtained after the learning phase we choose the best one, i.e. the one with the higher success rate (98.5% in 16100 learning episodes). In the following, some representative examples of the obtained paths are detailed. They show the capability learned by the VA agent to guide the robot paying attention to avoid obstacles, to maintain the robot in the field of

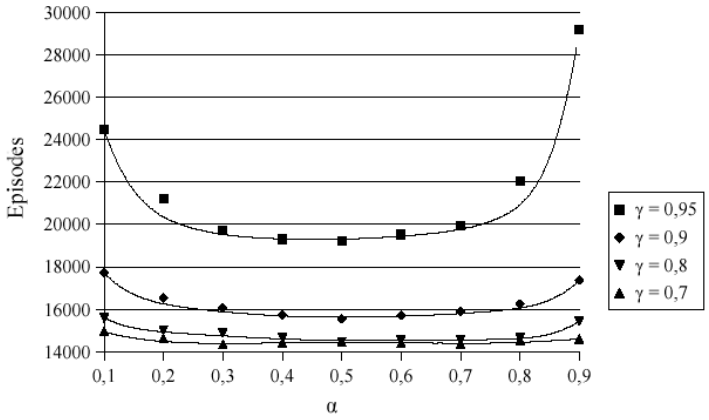


Figure 11: Learning episodes versus the α parameter for $\lambda = 0.1$ and different values of γ.

view and to choose the proper velocities. To visualize the behavior learned by the VA, the graphical package of Scilab [10] has been applied.

Figures 13, 14 and 15 show some representative sample paths obtained by using that policy.

Even though an empty environment has been considered, the camera itself represents an obstacle. Fig. 13 shows that the agent has acquired the capability to avoid the obstacle. The path is drawn with the long vectors which represent the direction of movement, the short vectors, instead, indicate the robot orientation. Observing the figure, it is evident that the VA chooses high velocities at the beginning of the path and lower ones as it approaches the target position. Fig. 14 shows a sample path which underlines this ability. The VA chooses low velocities when the robot moves in the proximity of both the obstacle and the target.

Fig. 15 displays the ability of the VA, to keep the robot inside its field of view. In particular the VA moves the robot carefully in the most external region (Z_5) choosing low velocities, then it increases the velocities as the robot moves away from Z_5 and finally it becomes careful again close to the target.

All the presented examples demonstrate that the VA has learned to move the robot by using the proper velocities not only as function of the distance from the target, but also as function of the distance from the camera (calibration ability). In the inner regions the VA has learned to use low velocities even if the target is far from the robot. Notice that little image distances near the camera correspond to little distances in the real environment, but little image distances far from the camera correspond to large distances in the real world.

The second part of the experimentation is conducted in order to prove the distribution of the learnt knowledge to new VAs. In particular two new VAs are considered in the Distributed Vision System. These VAs have a different camera setup: one has the camera at the height of $2.0m$ above the floor and the second one at $2.4m$. Our aim is to

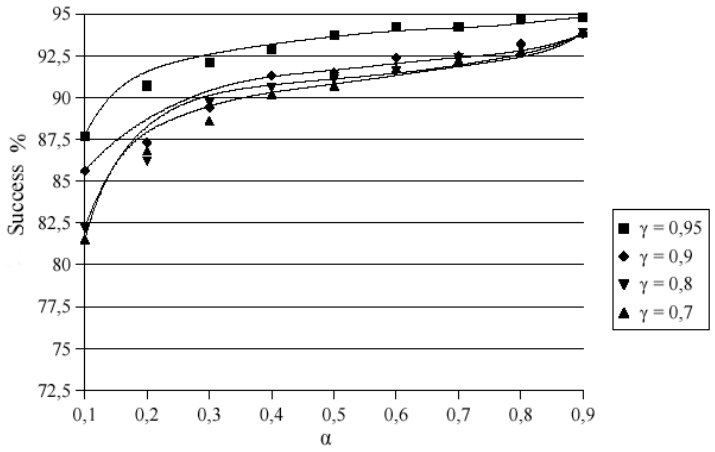


Figure 12: Percentage of success versus the α parameter for $\lambda = 0.1$ and different values of γ.

demonstrate that the new VAs can exploit the policy acquired by the initial VA in order to reduce learning time and, more important, to improve their performance. In such a way the same knowledge can be spread over different VAs so that they will be operational from the very beginning and they will need only a little amount of re-learning to perform optimally.

The knowledge learnt by the first VA is used as a starting policy for the new VAs, and new learning missions are defined based on the LEM technique. Partitioning the state space only considering the distance between the robot and the target, we obtain seven missions each one with a cardinality not over 960 states. Several re-learning simulations are carried out considering different values for the Δt parameter. Recalling that $\Delta t > |S_k|$, Δt is varied in the range $[1800, 2200]$, obtaining a range of [14000,15400] for the number of learning episodes. Testing the policies on additional 200 test episodes, after each simulation, both VAs reveal high success rates ($> 91\%$). As expected the higher is Δt, the higher is the success rate. In the next referenced figures the paths executed by using the policies with the highest success rates (97.5% for the first VA and 97% for the second one) will be displayed. Both new VAs are still able to guide the robot toward the target, but a clear improvement of the policy emerges after the re-learning phase.

Figure 16 shows two paths, that start from the same initial position and reach the same target position, before the re-learning phase (fig.16a)) (camera height = $1.8m$) and after the re-learning phase (fig.16b)) (camera height = $2.0m$) respectively. Improved paths are also been obtained by considering the other VA with the camera but at the height of $2.4m$ above the floor. Figure 17 shows a sample path for this case. The presented results reveal that the knowledge distribution allows the new VAs to acquire improved control policies. In fact, for the sake of completeness, we carry out another experiment where each new VA learns its own policy starting from a zero knowledge.

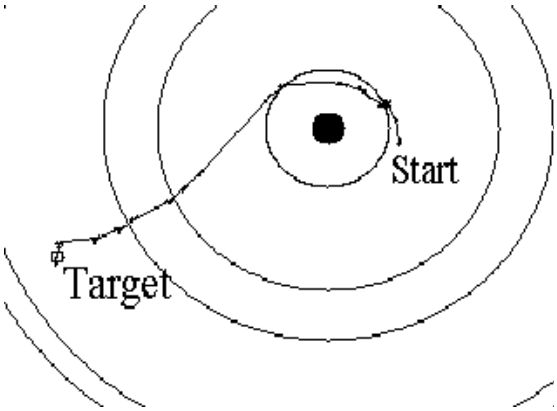


Figure 13: Sample paths obtained after the learning phase: the robot successfully avoids the obstacle (black circle).

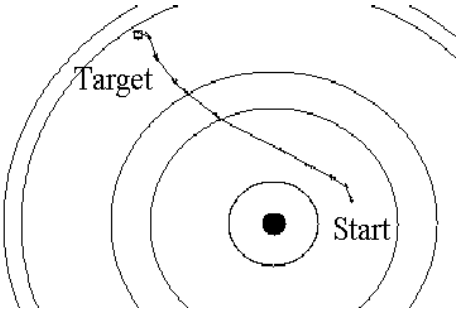


Figure 14: Sample paths obtained after the learning phase: the robot uses different velocities.

Apart from the learning time and the success rate, that are comparable with the ones of the first experiment, what emerges is that with knowledge distribution each VA exhibits a more optimal behavior.

8 Experiments in the Real Environment

The good results obtained in simulation, as shown in the previous section, encouraged the experimentation of the whole system in a real environment.

The real system is composed of two omnidirectional cameras with hyperbolic mirrors with a maximum resolution of 640×480 pixels. The robot is based on a Pioneer 2 platform but was modified to play in RoboCup as a goalkeeper. In this work the omnidirectional camera on board of the robot is removed and the robot is controlled via IEEE802.11 wireless LAN by the Omnidirectional Distributed Vision System. A screenshot of the the Vision Agent Monitor showing the image captured by one VA is

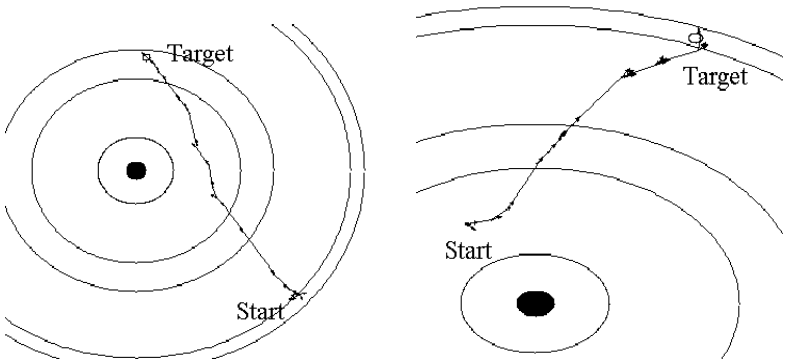


Figure 15: Sample paths: the VA keeps the robot inside its field of view.

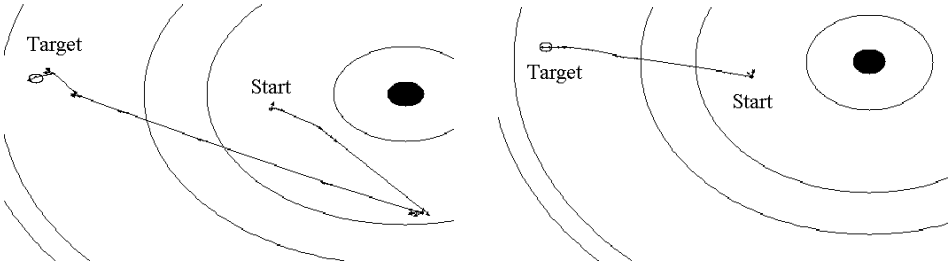


Figure 16: Sample paths: before the re-learning phase (left) and after the re-learning phase for the VA with $2.0m$ high camera (right).

shown in Fig. 18b).

To easily detect the robot in the image, two coloured markers are fixed on the top of the robot (see Figure 3). By colour thresholding, the VA is able to detect the coloured blobs on the image. The midpoint between the centers of gravity of the two blobs gives the position of the robot, whereas the line passing between these centres gives the heading of the robot. When the robot is far away from the VA, due to the low resolution achievable with the omnidirectional camera the estimation of the markers is noisy. To have a more reliable measurement of the robot's position, this is calculated as the average of the position estimated in five consecutive frames. In order to avoid to look for the coloured blobs on the whole image, a background subtraction algorithm is applied to detect the region where the robot is located.

Fig. 18a) shows the timing of the information flow introduced in Fig. 4. This is a cyclic process starting with the acquisition of a frame from the camera. Then, the image is processed by the VM (Vision Module) and the information about the robot position is passed to the LM (Learning Module). The LM sends the motor commands to the robot. The robot execute the motor command moving from one position to the new one. Once the robot stops a new frame is grabbed by the camera and the cycle is repeated.

The stop and go procedure adopted for the robot is good for the learning method

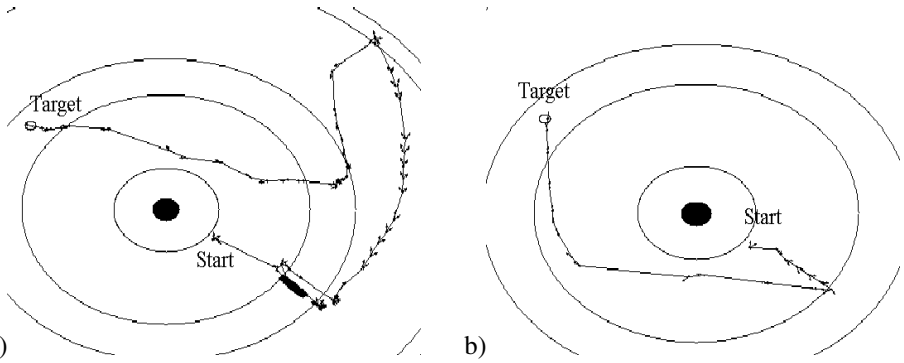


Figure 17: Sample path a) before the re-learning phase and b) for the VA with $2.4m$ high camera after the re-learning phase.

Start (pxl)	Stop (pxl)	Episodes	Successes	% of Success
4	8	14	10	71%
8	12	14	9	64%
12	30	14	11	79%
30	60	14	11	79%
60	120	14	8	57%
120	240	14	6	42%
240	480	14	5	36%
TOT		96	60	62%

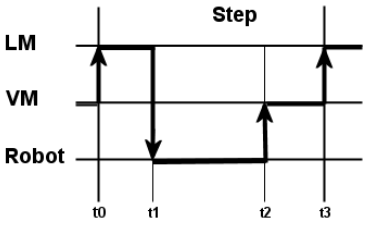
Table 2: Experimental results of missions in the real world using the knowledge acquired in simulation.

adopted, but it is time consuming. Considering that to learn a very effective policy requires the robot to accomplish 15'000 missions, to start from no knowledge and learn everything from scratch with the real robot would take too long. So, we use as initial knowledge the knowledge acquired in the simulations. Exploiting the ability of life-long learning of RL, the system can adapt the knowledge acquired in the simulation with the specific knowledge required to drive the robot with that specific camera.

Experiments showed that using the simulation knowledge, even if the simulated camera and robot greatly differ from the real ones, the real system is able to accomplish simple missions with a high percentage of success, see Table 2. More complex missions (i.e. longer missions, in term of pixels) requires additional learning to be executed reliably, see Table 2. Nevertheless, this shows the selected method is capable of achieving our goal.

The experimentation in the real world revealed some problems when the knowledge acquired in the simulation is used in the real world. The first problem is the time constrains. In simulation we can execute a huge number of simulation steps in a relatively short time, on the contrary in the real world every mission is time consuming,

a)



b)

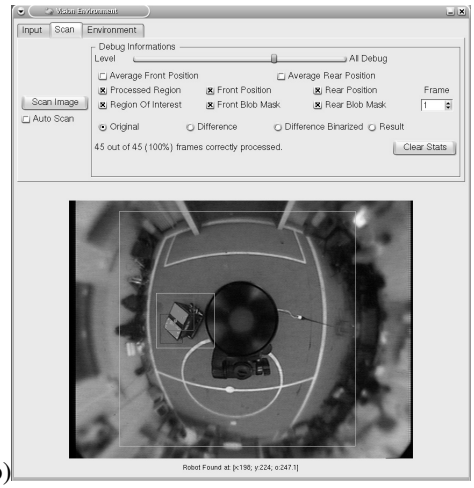


Figure 18: a) The timing of a mission execution. b) A screenshot of the VA monitor showing the GUI and the image grabbed by one VA.

so we must extract the most from every mission. One problem is the missions of the real robot must have a maximum number of step allowed before the mission is declared failed. This is to avoid the system will spend too much time in a endless mission. The problem arises when the system starts the execution of the mission in a wrong way, but then recovers and performs according to the policy to be learnt. If in the first "wrong" stage the system spent a lot of steps, it will not reach the goal in the maximum allowed step. This will result in a negative reward and, because of the eligibility trace technique, the last steps would be penalized more than the first even if they were about to bring the robot to the correct destination. To overcome this, one possibility can be to have a different reward procedure for a failed mission and one mission that reached the maximum number of allowed steps.

Another choice that appeared appropriate for the simulation, but should be further inspected in the real world application is the ε-greedy policy adopted and the choice of the ΔT in the LEM. A better choice of these parameters will enable to speed-up the learning also in the real world.

9 Conclusions and Discussion

This work presents an Omnidirectional Distributed Vision System having the task of navigating a mobile robot in its environment. It consists of a number of VAs that autonomously learn to guide the robot from a starting position to a target one. We apply the RL method of $SARSA(\lambda)$ to learn the control policy. In particular one VA learns an initial policy to control the robot, then the same policy is transferred to other different VAs with the aim of testing the ability to share knowledge. The complexity of such a system is additionally connected to the use of omnidirectional vision sensors.

The main contribution of this work is about the use of uncalibrated vision systems which learn autonomously the proper commands to send to the robot for the navigation.

The experiments in simulation demonstrate that the VA learns successfully its task solving all the problems it involves: obstacle avoidance, choice of the right velocities, path optimization. Also the adaptability of the learned policy to environment changes and then the propagation of the acquired knowledge to different VAs has been proved. Experiments showed the policy learned in simulation can be used as initial knowledge to continue the learning in the real world. At the moment of writing we are evaluating the amount of re-learning necessary in different situations and the capability of distributing the knowledge in the real environment.

The experimentation demonstrate that the distribution of knowledge from one VA to other different VAs is possible and that it is advantageous not only for learning time saving, but also for the improvements that the VAs exhibit after the re-learning phase. In fact the VA is immediately operational, because of the no-zero starting knowledge, and it learns more and improves with experience since it is allowed to explore again the environment. In effect, the two separate steps of initial learning and re-learning constitute two constraints for the whole learning process. As the experiments demonstrate, this effectively improves the decision quality and performance of agents.

A final consideration can be done about the application of $SARSA(\lambda)$ to that complex robotic problem. In this work the value function is approximated by using a standard tabular formulation which assumes discrete sets of states and actions. This choice is not restrictive since our aim is first to investigate if the VA is able to achieve a good policy to guide the robot. A possible optimization of the approach, considering continuous variables, could be a consequence of this first study. To estimate the value function, in fact, another possible solution could be to use function approximators in order to deal with continuous state and action spaces. This problem is receiving a great deal of interest by the scientific community [27, 23, 20, 18] since in real robot applications sensor and action spaces are continuous.

References

[1] M. Asada. A case study on behavior learning for vision-based mobile robot. In *IROS Workshop on Towards Real Autonomy*, pages 3–16, 1996.

[2] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[4] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 709–716, Melbourne, Australia, 2003.

[5] Gang Chen and Zhonghua Yang. Coordinating multiple agents via reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 10:273–328, 2005.

[6] G. Cicirelli, T. D'Orazio, L. Capozzo, and A. Distante. Learning elementary behaviors with khepera robot. In *Proc. of first International Khepera Workshop*, pages 109–118, Paderborn, Germany, 1999.

[7] G. Cicirelli, T. D'Orazio, and A. Distante. Learning a door-reaching behavior using visual information. In *IASTED-Int. Conference on Control and Applications*, Cancun, Mexico, May 2000.

[8] R. Collins, A. Lipton, and T. Kanade. A system for video surveillance and monitoring. Technical report, Robotics Institute at Carnagie Mellon, 2000.

[9] James Davis and Xing Chen. Calibrating pan-tilt cameras in wide-area surveillance networks. In *International Conf. on Computer Vision (ICCV)*, 2003.

[10] Claude Gomez, editor. *Engineering and scientific Computing with Scilab*. 1999.

[11] H. Ishiguro. Distributed vision system: a perceptual information infrastructure for robot navigation. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 36–43, 1997.

[12] H. Ishiguro and M. Trivedi. Integrating a perceptual information infrastructure with robotic avatars: a framework for tele-existence. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'99)*, October 1999.

[13] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Eighteenth National Conference on Artificial Intelligence*, pages 326–331, Edmonton, Alberta,Canada, 2002.

[14] Ling Li, A. Martinoli, and Y. S. Abu-Mostafa. Emergent specialization in swarm systems. *Lecture Notes in Computer Science*, 2412:261, 2002.

[15] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, (2-3):311–365, 1992.

[16] Carlos Mariano and Eduardo Morales. A new distributed reinforcement learning algorithm for multiple objective optimization problems. *Lecture Notes in Computer Science*, 1952:290, 2000.

[17] E. Menegatti, E. Pagello, T. Minato, T. Nakamura, and H. Ishiguro. Toward knowledge propagation in an omnidirectional distributed vision system. In *Proc. of the 1st Int. Workshop on Advances in Service Robotics (ASER2003)*, March 2003.

[18] J. Del R. Millan, D. Posenato, and E. Dedieu. Continuous-action q-learning. *Machine Learning*, 49:247–265, 2002.

[19] J. del Rio Millan. Rapid, safe, and incremental learning of navigation strategies. *Sys. Man and Cybernetics*, 26(3), 1996.

[20] G. Mohammad and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *Eighteenth International Conference on Machine Learning (ICML)*, Williams College, Massachusetts, July, 2001.

[21] B. D. Olsen. Calibrating a camera network using a domino grid. *Pattern Recognition*, 34(5), 2001.

[22] Liviu Panait and Sean Luke. Cooperative multi-agent learning:the state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[23] M. Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Journal of Neural Computing and Application*, 8:323–338, 2000.

[24] P. E. Rybski, M. Gini S. A. Stoeter, D. F. Hougen, and N. P. Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE Trans. on Robotics and Automation*, 18(5), October 2002.

[25] E. Shakshuki and K. Rahim. Multiple reinforcement learning agents in a static environment. *Lecture Notes in Computer Science*, 3029:997–1006, 2004.

[26] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[27] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proc. of the Seventeenth International Conference on Machine Learning*, pages 903–910, San Francisco, USA, 2000.

[28] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[29] R. S. Sutton and A. G. Barto. *Reinforcement Learning: an introduction*. A Bradford Book, 1998.

[30] Tomas Svoboda, Daniel Martinec, and Tomas Pajdla. A convenient multi-camera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments*, 14, 2005.

[31] H. Takeda, N. Kobayashi, Y. Matsubara, and T. Nishida. A knowledge-level approach for building human-machine cooperative environment. *Collective Robotics*, 1456:147–161, 1998.

[32] C. J. Watkins and P. Dayan. Technical note - q-learning. *Machine Learning*, 8(3/4):323–339, 1992.