Reinforcement Learning based Omnidirectional Vision Agent for Mobile Robot Navigation

E. Menegatti[1*], G. Cicirelli [2†], C. Simionato [1], A. Distante [2], and E. Pagello [1]

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

Abstract. This paper presents an Omnidirectional Vision Agent able to learn to navigate a mobile robot in its working environment. The novelty of the work is the application of Reinforcement Learning paradigm to Vision Agents aiming to develop a totally autonomous system able to learn control policies by on-line learning, to deal with changing environment and to improve its performance during lifetime. $SARSA(\lambda)$ method is used by the Vision Agent to learn the control policy for the robot. The *LEM* strategy is also applied to speed up learning. The knowledge acquired by one Vision Agent is then "copied" to another Vision Agent in a network of cameras implementing a Distributed Vision System (DVS). By copying the knowledge the aim is both reducing learning time and exploiting the knowledge already learned. Since our prime interest was to investigate how the Vision Agent learns the knowledge by using $SARSA(\lambda)$ and to evaluate its performance, we carried out the experimentation in simulation. The good results obtained during the experimental phase are very encouraging to transfer all the experimentation in a real context.

1 Introduction

Nowadays, the robotics community is shifting its attention from a single agent performing its task in a cell environment to a group of agents cooperating together to achieve a common goal in unmodified human environments. Several projects explored the possibility to support the human and robot activity in the environment with a network of smart sensors [3, 11]. In a previous paper 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 [5, 6]. The network of cameras is named Distributed Vision System (DVS)[5, 6] to emphasize that the system of cameras acts as a whole at the aim of reaching the final objective: to 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 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. Each VA will take the control of the robot every time it enters its field of view. The novelty of our approach is that the cameras are uncalibrated and the robot does not have any sensor or processing power on board. The robot is just a “dummy” mobile platform which motors are driven by the Vision Agents in the DVS. Besides, we want to deal with uncalibrated cameras, because the idea is to use the system in environments in which there is already a network of surveillance sensors. 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. For instance, Olsen proposed to use a set of special tiles to calibrate a large set of cameras observing multiple connected rooms and corridors [8].

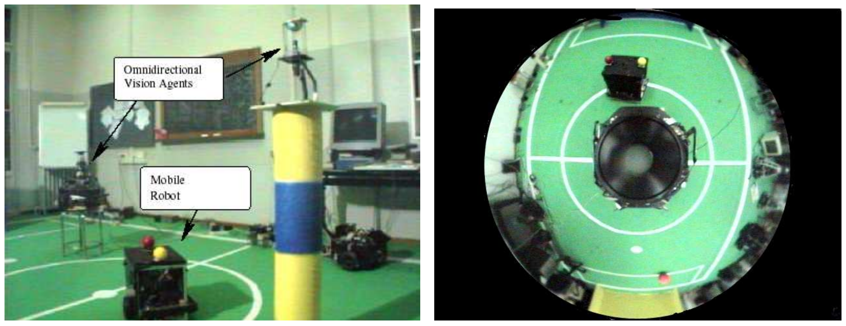

Fig. 1. a) A picture of the whole system showing two Vision Agents and the robot used in the experiments. b) Example of image taken by the omnidirectional camera.

We propose a different approach. We do not want to program a robot by using 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 [7] the same DVS was introduced, but what differs that system from the one proposed in this work is the learning procedure. In [7] neural networks were used to learn 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, instead, a different learning methodology is used. 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) [10] seems a very appropriate paradigm to face this learning problem since its main attractive is 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 in last years [1, **?**,**?**]. By using RL, the learning agent acquires the control policy by directly interacting with initially unknown environments and then by discovering which actions are the best ones, in each encountered situation, simply by trying them. The only feedback to the agent, coming from the environment, is the reward (credit or blame) received for each situation/action pair. The final aim of the RL agent is to maximize a value function that estimates the expected cumulative reward in the long term.

In this paper we apply RL paradigm to our DVS. We developed 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 which can improve it still experiencing the world and adapting the initial knowledge to the eventual differences with respect to the first VA. In this way the learning time is reduced since the new VA exploits the initial knowledge, instead of starting from a zero one.

Aim of this work is to prove that the VA is able to learn to control the robot and to improve and to adapt its knowledge as new situations in the environment happen. This is achieved by using a RL method, and carrying out hard problems such as obstacle avoidance and path optimization. In particular $SARSA(\lambda)$ with *Replacing Eligibility Traces* has been used as RL method. Besides, the *LEM* [2] technique has been applied at the aim of speeding up learning. In this first stage of the work, the experimentation has been carried out in simulation in order to investigate how $SARSA(\lambda)$ method applies to the VA, considering also an optimization of the involved parameters. Moreover, experimentation on the adaptability of the learned knowledge to a change in the environment has been carried out. The experimental results proved the efficiency and the robustness of the learning system.

The rest of the paper is organized as follows. Section 2 details the learning system. Section 3 describes the simulator used for the experimentation. Section 4 reports the experimental results. Finally some conclusions end the paper.

2 LEARNING SYSTEM

The task of each VA is to navigate the mobile robot in its own field of view, sending it motion commands to move from a starting position to a target one. The choice among such 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. 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. But our aim is to have a system of completely uncalibrated cameras able to learn by itself to guide the robot by using the adequate velocities and to adapt the acquired policy to changes in the environment. In the experimentation we will consider a free-obstacle environment. This is not a limitation since the cameras, being fixed on tripods, represent obstacles in the environment. The VA has also to guide the robot avoiding these obstacles.

Concluding 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 collides with the tripods (*obstacle avoidance problem*)
– to adapt the learned knowledge in case of new situations in the environment

Next subsections describe the principal components for a RL agent. In particular 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.

2.1 $SARSA(\lambda)$

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 control method [9]. The main peculiarity of such a method is that it evaluates the policy that is used to make decisions. In other words on-policy methods update gradually the policy based on the approximate values for 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.

At each time step, $SARSA(\lambda)$ updates all 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* which are updated for all s, a as follows:

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

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.

2.2 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 possible 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 [10] that chooses most of the time the actions with the maximal estimated action value. The actions with a lower action value are chosen with probability ϵ. The probability is higher at the beginning, whereas it is decreased incrementally with time. This enables a major exploration at the beginning and more exploitation of the acquired knowledge during advanced learning phase.

2.3 State Space

The 640×480 image captured from the VA is the only source of information it has about the environment. 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 $480 pixel$ and internal diameter of $20 pixel$. 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 is represented by a point 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 analyzed its particular task. The information needed by the VA to carry out its task are the following:

- 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 has been classified into 8 different classes considering a partition of the range interval of d as shown in table 1.

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

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}	$240 < d \leq 480$

Considering the distance between the robot and the camera, the circular ring of the image, described before, has been divided into five concentric regions as shown in fig. 2. The black circle (with 20 $pixel$ radius) in the center of the image represents the reflexed image of the camera. The black area around the ring is out of the field of view of the camera. Z_1 region has a distance of 30 $pixel$ 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

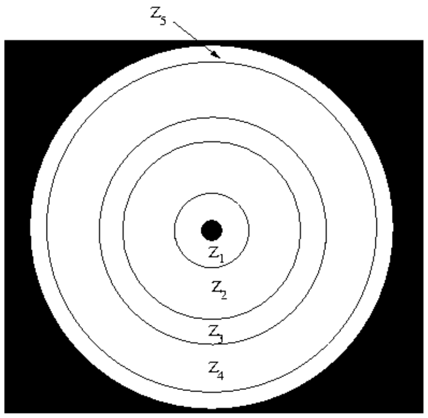

Fig. 2. The five regions of the image space.

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) have been 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 has been divided only into three regions: $ZT_1 = Z_1 + Z_2$, $ZT_2 = Z_3$ and $ZT_3 = Z_4 + Z_5$ 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 to reach the target position with a straight trajectory. On the contrary it is $notfree$ if the 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 have been considered. Fig. 3a) 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: $Left$, $Front\text{-}Left$, $Front$, $Front\text{-}Right$, $Right$, $Right\text{-}Rear$, $Rear$ and $Rear\text{-}Left$ as shown in fig. 3b).

Concluding, the state of the VA has been defined considering all the elements described above. In particular the state is represented by the 5-tuple (d, Z_i, ZT_j, S_C, S_T) where $i = 1, ..., 5$, $j = 1, 2, 3$. The total number of states is 7680, some of them are terminal states, but a lot of them are impossible states. At the end the number of effective states does not exceed 2500.

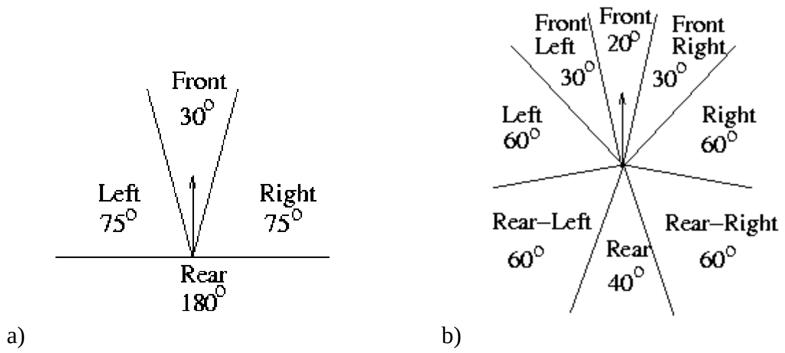

Fig. 3. The angular sectors which define the relative orientation between a) the robot and the camera and between b) the robot and the target position.

2.4 Action Space

The robot can receive commands in terms of linear and angular velocity $(linear, jog)$, then we have defined an action as a pair of these two velocity values. Each velocity has been divided into sub-actions: 3 for the $linear$ velocity ($stop$, $slow$, $fast$) and 5 for the jog one ($stop$, $slow$-$left$, $slow$-$right$, $fast$-$left$, $fast$-$right$). The ($stop, stop$) action has not been considered, 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 (*State-Action Deviation Problem)*. To deal with this problem we have adopted 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 it correctly updates the action value function by using eq. 1.

2.5 Reward Function

The VA 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.

3 THE SIMULATOR

In order to study the feasibility and the quality of the proposed learning system, we have 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 has been considered. Knowing the starting position of the robot the simulator receives as input the pair of velocities $(linear, jog)$ and gives as output the new position of the robot on the image. A graphical interface has not been implemented because our aim was to build a system fast and with a very low computational cost. The graphical package of Scilab [4] has been applied afterwards to visualize the behavior learned by the VA.

The simulated system has been very useful not only to learn and to test the policy of the VA in a virtual environment, but also to optimize the parameters (α, γ, λ) involved in the RL algorithm used. There's not a standard way to define those parameters to guarantee the convergence of learning. Usually they are heuristically defined and are strictly connected with the particular application. In our work we have deeply studied that problem, but due to the limited space we don't deal with it here. The optimal values obtained for those parameters and used in the implementation of $SARSA(\lambda)$ are the following: $\alpha = 0.5$, $\gamma = 0.95$ and $\lambda = 0.2$.

4 EXPERIMENTAL RESULTS

The experimental phase have been conducted in order to analyze both the performance of the VA and the adaptability of the learned policy to changes in the environment. To this purpose a number of simulations have been carried out. Each simulation consists of two stages: a learning phase, starting from a zero knowledge and by using the LEM strategy and a testing phase to examine the performance of the VA. By using the LEM strategy easy situations (or missions) are presented to the VA first and more difficult ones as learning goes on. This technique has been proved to speed up learning [2]. It is based on partitioning the state space in a number of missions. In our work 1120 missions have been defined combining all the elements involved in the state definition: distance from the camera, distance from the target and so on. The learning phase took 16744 episodes ($123sec$ running on a 2GHz Pentium 4) to complete all those missions. In each episode the VA starts from a state, randomly chosen among the possible ones in a mission, and at each state transition it updates the action value function so increasing its knowledge. If that update does not exceed a fixed threshold than the VA passes to the next mission. Each mission is explored performing Δt episodes. The Δt parameter is fixed considering that it must exceed the maximum cardinality of the missions (notice that a mission is a subset of the state space)[2]. Besides we have chosen it as a trade-off between learning time and percentage of success. During the execution of the learning missions the success rate of the VA has been also evaluated (i.e. the percentage of successful episodes). The VA guides the robot to the target with a rate of 96%. This is not enough to evaluate the performance of the VA, therefore at the end of the learning phase the learned knowledge has been tested running 200 additional trial episodes. In each episode the VA has to guide the robot from an arbitrary initial position toward a fixed target position. During testing phase the VA chooses the actions by using a greedy policy and learning is turned off. The success rate obtained after the test is 99%. In the following, some representative examples of the obtained paths will be detailed showing the capability learned by the VA agent to guide the robot paying attention to avoid obstacles, to maintain the robot in the field of view and to choose the proper velocities.

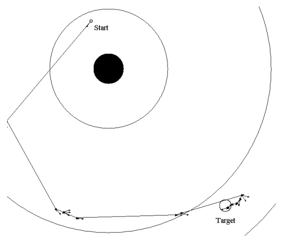

Fig. 4. Sample path 1: the VA moves the robot avoiding successfully the obstacle (black circle).

Even though an empty environment has been considered, the camera itself represents an obstacle. Fig. 4 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, one can see that the VA chooses high velocities at the beginning of the path and lower ones as it approaches the target position. Fig. 5 shows another sample path which underlines this ability. The VA chooses low velocities when the robot moves in proximity of both the obstacle and the target. Fig. 6 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.

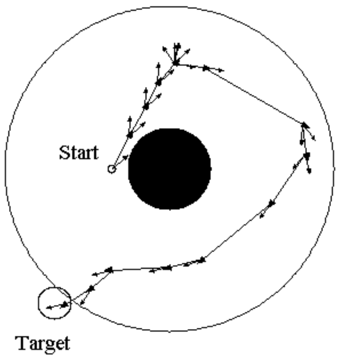

Fig. 5. Sample path 2: the VA moves the robot avoiding successfully the obstacle and choosing different velocities.

All the presented examples prove 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.

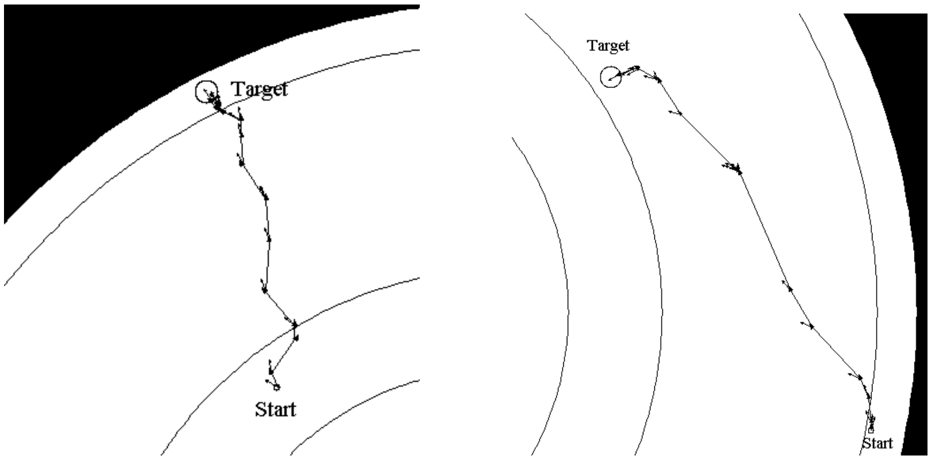

Fig. 6. Sample paths 3/4: the VA keeps the robot inside its field of view.

In the second part of the experimentation the aim was to prove the adaptability of the VA if something changes in the environment. In particular, if the camera setup changes the VA should learn again the whole policy from a zero knowledge, but this would involve again a learning time of 16744 episodes. Our aim is to demonstrate that it's possible to exploit the acquired knowledge in order to reduce learning time. Moreover in such a way the same knowledge can be spread over other VAs so that they will need only a little amount of re-learning to be operational.

The camera setup has been changed modifying its height from the floor, then the VA has to adapt its knowledge to the new environment. Exploiting the fact that the VA has a success rate of 96% just after the previous learning phase, we decided to use the previous knowledge as a starting policy and to define new learning missions 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. The re-learning phase took 14000 episodes by using $\Delta t = 2000$. The learning time has been considerably reduced and could be further minimized if a lower ΔT was used. After learning additional 200 test episodes were executed. They have already shown that the VA is still able to guide the robot toward the target. In addition, an improvement of the policy has emerged after the re-learning phase. Fig. 7 shows two paths starting from the same initial position and reaching the same target position inside the Z_2 region: the first one has been performed by the robot before the re-learning phase (fig.7 a)); the second one after the re-learning phase (7 b)). Comparing both paths we can notice that in the last one the VA chooses high velocities at the beginning and low velocities only when the robot is very close to the target. The VA has improved its policy.

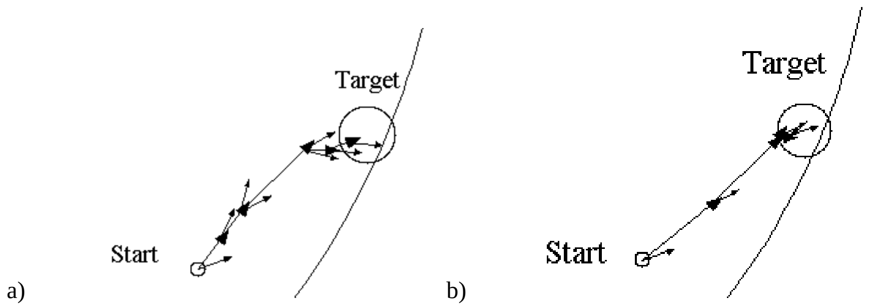

Fig. 7. Path executed by the robot inside the Z_2 region a) before and b) after the re-learning phase.

5 CONCLUSIONS

This work presents the application of $SARSA(\lambda)$ to a complex robotic problem, that is the possibility for a Vision Agent (VA) to guide a service robot from a starting position to a target one. The complexity of such a problem is also connected to the use of omnidirectional vision sensors. However the experiments have demonstrated 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. Next step of this work will be to continue the experimentation in a real environment exploiting the policy learned in simulation and allowing the real agent to continue its learning in the real situations.

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. R. Collins, A. Lipton, and T. Kanade. A system for video surveillance and monitoring. Technical report, Robotics Institute at Carnagie Mellon, 2000.
4. C. Gomez, editor. *Engineering and scientific Computing with Scilab*. 1999.
5. 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.
6. 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.

7. 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.
8. B. D. Olsen. Calibrating a camera network using a domino grid. *Pattern Recognition*, 34(5), 2001.
9. R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
10. R. S. Sutton and A. G. Barto. *Reinforcement Learning: an introduction*. A Bradford Book, 1998.
11. 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.