Hierarchical Reinforcement Learning for Robot Navigation using the Intelligent Space Concept

L. A. Jeni, Z. Istenes
Eötvös Loránd University
Faculty of Informatics
Pázmány Péter st., 1/c – 1117 Budapest, Hungary
{jedi,istenes}@inf.elte.hu

P. Korondi
Budapest University of Technology and Economics
Department of Automation and Applied Informatics
Goldmann György tér, 3 – 1111 Budapest, Hungary
korondi@elektro.get.bme.hu

H. Hashimoto
University of Tokyo
Institute of Industrial Science
4-6-1 Komaba, Meguro-ku, Tokyo, Japan, 153-8505
hashimoto@iis.u-tokyo.ac.jp

***Abstract* – Navigation in an unknown environment is a difficult task, because mobile robots need topological maps in order to operate in the environment. Another fundamental problem is that robot programming is a time-consuming process, so it is better to use a learning method with reinforcement. In previous work we proposed a learning framework, which used the capability of the Intelligent Space in order to build a topological map of the environment. In this paper we present an extension of this framework to decompose the learning problem into sub-problems, which can be learned faster.**

I INTRODUCTION

Navigation in a completely unknown environment is still a hard problem, because mobile robots need topological maps in order to operate in the environment. Building a map of the environment while also using it for learning is of prime importance for mobile robots but until recently, it has only been confined to small-scale environments.

Reinforcement learning is a machine learning paradigm that is well-suited use on mobile robots. The main problem, which arises when reinforcement learning is applied to larger scale environment, is referred to as the state space explosion. The learning time grows at least as fast as the size of the state space, so the learning time will grow exponentially in a larger domain.

One way to overcome this problem is to adopt a divide and conquer strategy. Rather than attempting to solve the whole problem at once, a decomposition is performed to create a hierarchical structure of sub-problems. These algorithms are known as Hierarchical Reinforcement Learning methods.

It is worth mentioning here the hybrid learning architectures, for example the Layered learning [1], which is successfully applied to learn low-level behaviors for a Robocup team [2]. Temporal abstraction, like H-DYNA [3] and HAMs [4], means the decomposition of a task into a set of sequential subtasks, which can be used to complete the task. The MAXQ decomposition [5] and ALisp [6] methods combines Temporal Abstraction with State Abstraction to improve the learning process. There has been some research into determining useful task decompositions without prior knowledge [7,8]. A recently proposed approach, called HEXQ [9], exploits a factored state representation and builds a task hierarchy using sorted state variables.

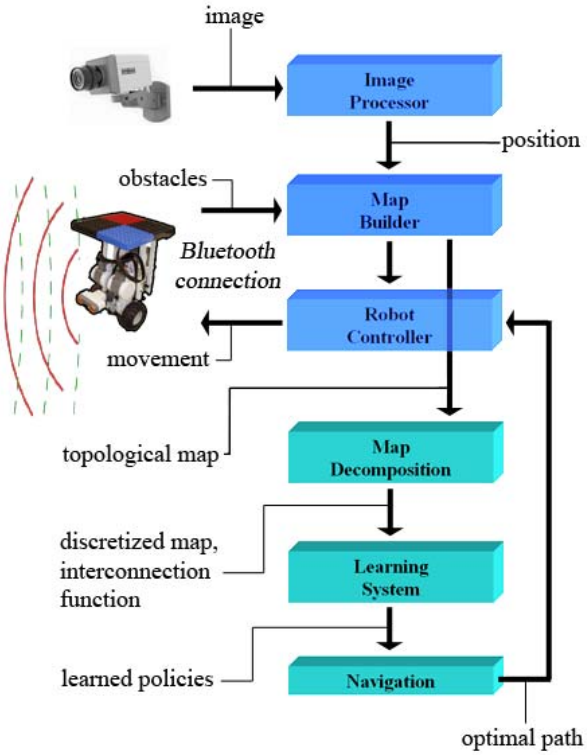

Fig. 1. Overview of the system.

In previous work we proposed a learning framework, which used the capability of the Intelligent Space in order to build a topological map of the environment [10]. In this paper we present an extension of this framework, which can create a hierarchy in the learning process. This approach is a generalization of the HEXQ method in some ways. It uses an interconnection function to build a task hierarchy.

An overview of the system can be seen in Fig. 1. The Intelligent Space can recognize and track the path of mobile robots and we can use this capability to create a topological map of the environment. We can use this feature to make a hierarchical generalization in the learning process by abstraction. The first three components of the system are described in detail in [10].

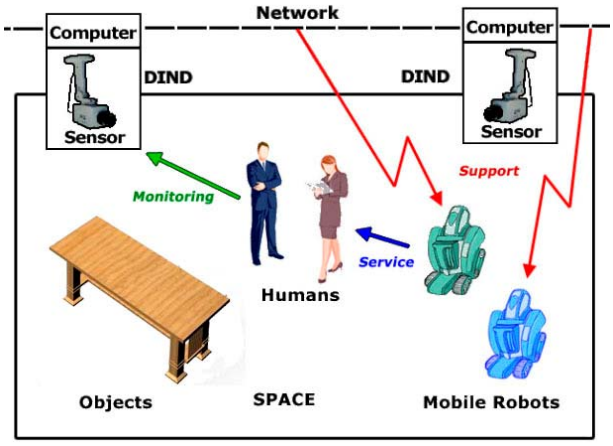

Fig. 2. Intelligent Space Concept.

This paper is organized as follows. The next section introduces the Intelligent Space concept. Section 3 briefly describes reinforcement learning and the TD-learning method, which is used in the framework. Section 4 describes the hierarchical decomposition method. In Section 5 we give an overview of the test environment and Section 6 shows experiment results.

II THE INTELLIGENT SPACE

Conventionally, there is a trend to increase the intelligence of a robot operating in a limited area. The Intelligent Space concept (see Fig. 2) is the opposite of this trend [11]. The Intelligent Space is a space (room or corridor), which has distributed sensory intelligence (various sensors, such as cameras and microphones with intelligence) actuators (projectors, speakers, and mobile robots) to manipulate the space. A robot without any sensor or own intelligence can operate in an Intelligent Space. In the conventional solution the robot measures, calculates and decides. The heart of the iSpace concept is that the robots must not calculate or make decision. They just carry out, execute commands getting information from the distributed devices called Ubiquitous Sensory Intelligence which is realized by Distributed Intelligent Networked Devices (DIND).

A DIND is consisting of three basic elements. These are the sensor, the computer and the communication device. In the Intelligent Space DINDs monitor the space, acquire data and share them trough the network. Since robots in the Intelligent Space are equipped with wireless network devices, DINDs and robots together organize a network.

The basic concept of Intelligent Space has extended with its development. The iSpace is a system for supporting people in it. Events, which happen in it, are understood. However, to support people physically, the intelligent space needs robots to handle real objects. Mobile robots become physical agents of the Intelligent Space and they execute tasks in the physical domain to support people in the space. Task includes movement of objects, providing help to aged or disabled persons etc. Thus, the Intelligent Space is an environmental system, which supports people in it electrically and physically. Another interesting application here is that the room can serve as a high level, context sensitive interface to robots. The Intelligent Space is a platform to which desultory technologies are installed.

The ongoing research activities about Intelligent Space achieved several results and solutions in the field of motion control [12], feature extraction [13] and recognition and tracking the path of moving objects [14].

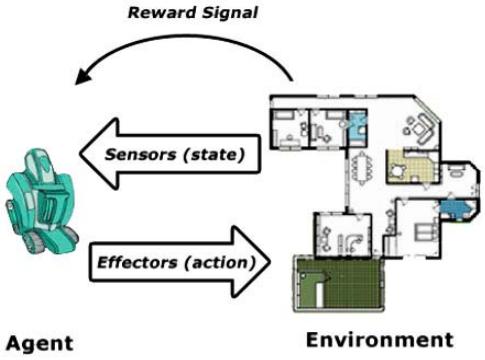

Fig. 3. Basic components of a reinforcement learning problem.

Recent research focuses on image recognition and on solutions that are developed on the analogy of the human vision processing [15].

III REINFORCEMENT LEARNING

A *Markov Decision Processes*

The easiest to describe the concept of a reinforcement learning problem is by considering an agent situated in some environment as shown in Fig. 3. The agent can detect information about the state of the environment. The agent can also affect the environment by taking one of a set of actions available to it. After each action is taken, the agent receives a feedback signal from the environment called the reward, which determines how well the agent is performing the target task in the environment. The goal in a reinforcement learning problem is to learn which action to take in each state to maximize some optimality criterion based on the rewards received over time. Some examples of optimality criteria are average reward per time step, total return over a finite horizon and total discounted return.

A reinforcement learning problem can be formalized as a Markov Decision Process [16] or MDP. An MDP is described by a quadruple <*S, A, T, R*> where:

- *S* is the set of possible states.
- *A* is the set of available actions.
- $T(s,a,s') \rightarrow [0,1]$ is the transition function defining the probability that taking action *a* in state *s* will result in a transition to state *s'*.
- $R(s,a,s') \rightarrow \boldsymbol{R}$ is the reward function defining the reward received when a transition is made.

A particular strategy for choosing actions in an MDP is known as a policy, and is specified formally as a function $\pi(s,a) \rightarrow [0,1]$, which defines the probability of selecting each action in a given state.

For some policy π and a discount factor $\gamma \in [0,1)$, the value function $V^{\pi}(s)$ can be defined as the expected total discounted return when starting in state *s* and using policy π to choose actions:

$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right] \quad (1)$$

Intuitively, the value function $V^{\pi}(s)$ represents how good it is for an agent to be in a particular state of the MDP, given that subsequent actions are to be chosen using policy π. The discount factor is used to determine the relative

worth of future rewards in comparison to rewards available immediately in the current state. The value of γ is chosen to be less than 1 to give V^{π} a finite value for each state. The optimal policy π^* is the policy which, according to the optimality criterion, performs better in the environment than any other policy π. The formal definition of π^* is:

$$V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in S. \tag{2}$$

While our goal is to find π^*, MDP solution methods are often based on a calculation of the value function for the optimal policy V^{π^*}, also denoted by V^*. Once V^* has been calculated, the parameters of the MDP can be used to calculate π^* as well:

$$\pi^*(s) = \arg\max_{a} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \tag{3}$$

B *Temporal Difference Learning*

The proposed learning framework uses Temporal Difference (TD) learning [18] to learn the value function V^{π} for the policy π being followed by the learning agent. The Temporal Difference method is a learning-method driven by the difference between two successive state values to adjust former state values, which decrease the difference between all two successive state values:

$$V_k(s_t) = V_{k-1}(s_t) + \alpha_k\left((r_t + \gamma V_{k-1}(s_{t+1})) - V_{k-1}(s_t)\right) \tag{4}$$

In (4) r_t is the immediate reward and α_k is a learning rate series that sums up to infinity, but whose squares sum up to a finite value. This method guaranteed converges to the optimal policy within a finite amount of evaluation.

In the framework TD is implemented using an eligibility trace. The eligibility trace for a state s is a value e_s which determines the extent to which s should be updated using the value of the current state s_t. At every time step each of the e_s values is updated as follows:

$$e_s \leftarrow \begin{cases} \gamma\lambda e_s & \text{if } s \neq s_t \\ \gamma\lambda e_s + 1 & \text{if } s = s_t \end{cases} \tag{5}$$

Once the eligibility trace values have been updated, the current estimate of each state value can also be updated:

$$\begin{aligned} \delta_t &\leftarrow r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \\ V(s) &\leftarrow V(s) + \alpha e_s \delta_t \end{aligned} \tag{6}$$

Furthermore we used ε-greedy [17] strategy (with ε = 0.1) to balancing exploration and exploitation in the learning process. This is a simple but effective mechanism for trading off the exploration of the random policy against the exploitation of the greedy policy. There is a small probability ε at each time step of picking an action at random, otherwise the greedy policy is followed. With a good choice of the value for ε, the policy will quickly converge to one which selects the optimal action with probability (1-ε).

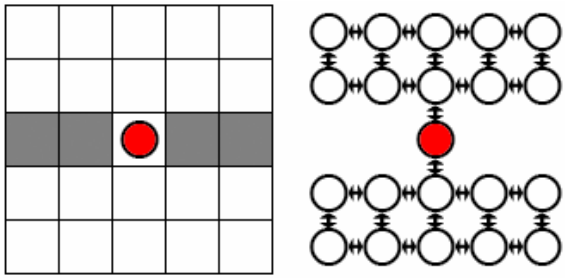

Fig. 4. An interconnecting state in a gridworld. This state connects two rooms.

IV HIERARCHICAL DECOMPOSITION

The main problem, which arises when reinforcement learning is applied to a larger scale problem, is referred to as state space explosion. Without any prior knowledge, reinforcement learning is almost certainly infeasible for state spaces above a certain size.

We can modify the original reinforcement learning problem to use an interconnection function to decompose the large state space. This function represents our prior knowledge about the problem in a formal way. We can describe this modified problem by a triple <*S, A, i*>, where *i* is the interconnection function.

At first, we describe the interconnection function and show how it can be used to decompose the problem, and then provide a method for learning on the decomposed problem.

A *The interconnection function*

The interconnection function defines the probability that a given state is a connecting state, which connects two partitions of problem. It maps the state of the environment to a single number, a probability value. This represents our prior knowledge about the problem.

$$i(s) \rightarrow [0,1],\ s \in S \tag{7}$$

We can say "bottleneck" states in the state space are interconnecting states, because they separate larger partitions. For example, in a gridworld problem where the world consist rooms and passages, the passage states are interconnecting states, because they connect two rooms (see Fig. 4).

If we have some exact knowledge about the problem (for example if we can recognize passages from the current state), we can explicitly define this function. Otherwise we need to approximate it by exploring the state space.

In the framework we used a topological map to define this function. The next section describes the details.

B *Learning Process using the Interconnection Function*

In the first part of the learning process the algorithm explores the state space by starting several trajectories. If we reach an interconnecting state (a state where $i(s) > \delta$, where δ is a constant) we create a new abstract state, which contains the states of the trajectory. If we found a path between two abstract states, which not contains a connecting state, then we merge the two states. At the end of this stage, we have a set of abstract states and each state represents a partition of the original state space.

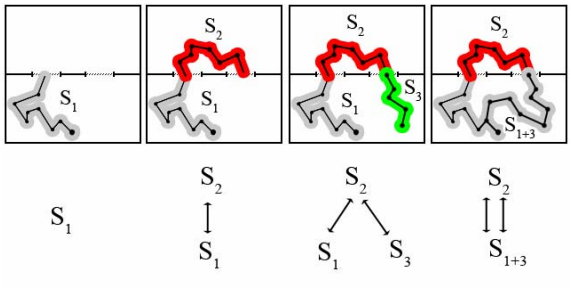

Fig. 5. Four steps from the abstraction process.

Furthermore each abstract state contains one or more connecting states. In the second part of the learning process the algorithm learns partial policies on the partitions represented by the abstract states (see Fig. 5).

Let M denote the number of abstract states. If an abstract state has N_i interconnecting states, then the algorithm learns N_i optimal policies on that partition (one for each connecting state or "passage"):

$$\pi_k^i, i \in [1..M], k \in [1..N_i] \tag{8}$$

Let A_k^i be a macro action, which represents the shortest path on the ith partition to the kth passage of this partition based on the optimal π_k^i policy. Let S_i denote the ith abstract state. With this notation we can define a Semi-MDP over the abstract states:

$$\left\langle \{..S^i...\}, \{..A_k^i...\}, T', R' \right\rangle \tag{9}$$

We can define the T' transition function using the connectivity information of the partitions.

In the final part of the learning process the system learns the optimal π' policy for this SMDP. This problem has a much smaller state space than the original.

V Overview of the Test Environment

For testing the learning algorithm, we used the image processing module of our robot soccer test framework developed at the Eötvös Loránd University. This provided us a good background for implementing the algorithm.

A *Building topological map using the iSpace concept*

To build the topological map of the environment the system propagates mobile robots in the space and tracks their movement.

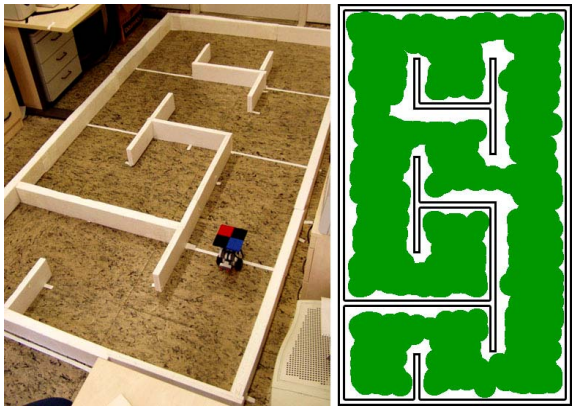

Fig. 6. The test scene in our lab (left) and the walkable area map (right).

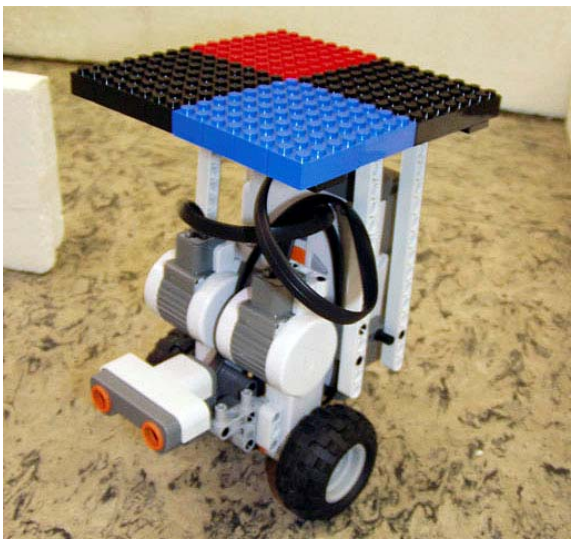

Fig. 7. The mobile robot. The ultrasonic sensor is located on the front side.

The robot recognition is done in two steps. First, the area of the moving objects is separated from the static background. Second, colored, square-shaped labels on the top of the robot are located. This colored label is proposed to recognize the orientation of the robot. Taking the images of several cameras we can then calculate the spatial position of the robot. The positions are dilated with a morphological operator to obtain a connected area map.

Our experiments were performed using a small test scene built from polystyrene sheets. The picture of the scene and the walkable area map can be seen in Fig. 6.

B *The mobile robot*

We used a unicycle-type wheeled LEGO robot for the simulation (see Fig. 7). This robot is based on a 32-bit ARM7 microcontroller controlled mobile platform (the NXT brick), which can communicate with the framework via Bluetooth.

It has two motor-powered wheels and it is equipped with an ultrasonic sensor, located on the front side, to measure the distance of obstacles. This ultrasonic sensor is quite accurate within small distances (below 20 cm), and the minimum range is approximately 3 cm, so it is suitable for our application.

The system recognizes the position and orientation of the robot, and queries the ultrasonic sensor for the distance of obstacles. Then using these data it creates the walkable map of the room.

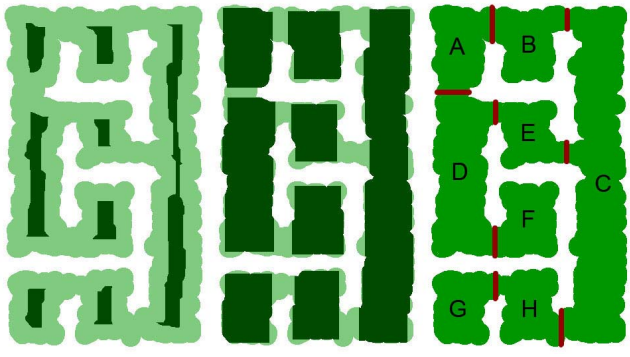

Fig. 8. The eroded (left) and dilatated (center) maps. The final decomposed map can be seen on the right.

C *Decomposition of the map*

In the next step the system decomposes the whole map into connected sub-maps. This decomposition will be the base of the temporal abstraction in the learning system.

For the decomposition we used two morphological operators. First, we eroded the walkable area map to break it into partitions, and then a dilatation operator was used to connect these parts again. At the end of this process we can allocate the passages between the partitions. The decomposition steps can be seen in Fig. 8.

VI EXPERIMENTAL RESULTS

In this section we present some experimental results comparing the flat TD-learning and our hierarchical learning algorithm.

Before the walkable area map is given to the learning system, the system discretizes the map into squares, to reduce computational complexity of the learning. Each square represents a state in the learning problem.

For the hierarchical algorithm we used the passages on the decomposed map as the interconnection function in the following way:

$$i(s) = \begin{cases} 1, & s \text{ lies on a passage} \\ 0, & \text{otherwise} \end{cases} \qquad (10)$$

In the flat problem the agent starts from the middle of area A, and gets a reward of +10000 for reaching the middle of area G (goal position). The agent can move in the four directions on the discretized map, with a reward of -1 on every step that does not end at the goal state. Furthermore the agent gets a reward of -5 for each bounce.

In the hierarchical problem the agent starts from a random position on each sub-map, and gets a reward of +100 for reaching a passage on that partition (or the goal position of area G). On the second level of the hierarchy, the agent gets a reward of +100 for reaching the goal state and a reward of -1 on every step that does not end at the goal state.

Fig. 9 displays the learning curves of the flat and hierarchical algorithms obtained by averaging over 100 runs. With the hierarchical decomposition it is possible to learn the task in very few learning episodes. The curve of the flat learner stabilizes around a performance of 70.9 after 41 episodes. The curve of the hierarchical method stabilizes around a performance of 70.3 after 13 episodes.

The results presented in this paper depend on the parameters of the used algorithm. In this example we set the trace-decay parameter λ to 0.95 and the discount factor γ to 0.9. Furthermore we used $\varepsilon = 0.1$ for the ε-greedy strategy.

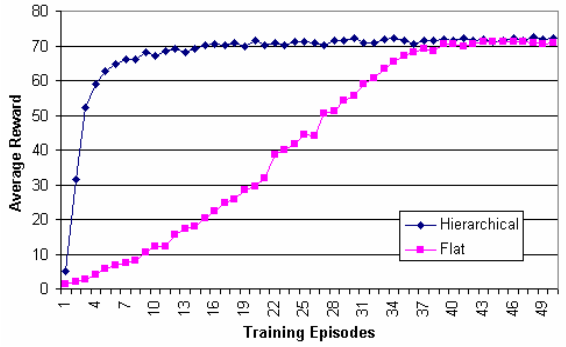

Fig. 9. Learning curves of the flat and hierarchical algorithms.

VII CONCLUSION

In this paper we have described a hierarchical reinforcement learning method, which can reduce the computational complexity of the learning problem. The results of the simulations illustrate that this algorithm performs much better than the flat learner algorithm, but it requires some prior knowledge about the problem.

The final goal is to create the learning hierarchy without any prior knowledge about the problem. In the future we need to investigate the possibility of approximating the interconnection function by exploring the state space.

Acknowledgments

The authors wish to thank the National Science Research Fund (OTKA K62836), Control Research Group and János Bolyai Research Scholarship of Hungarian Academy of Science for their financial support and the support stemming from the Intergovernmental S & T Cooperation Program.

REFERENCES

[1] P. Stone, "Layered Learning in Multi-Agent Systems." PhD thesis, Carnegie Mellon University, December 1998.

[2] H. Kitano, editor. RoboCup-97: Robot Soccer World Cup I. Springer Verlag, Berlin, 1998.

[3] S. P. Singh, "Reinforcement learning with a hierarchy of abstract models." In Proceedings of the AAAI, 1992, pp 202–207.

[4] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines." In Advances in Neural Information Processing Systems, volume 10. The MIT Press, 1997.

[5] T. G. Dietterich,. "Hierarchical reinforcement learning with the MAXQ value function decomposition." Journal of Artificial Intelligence Research, 13, 2000, pp227–303.

[6] D. Andre, "Programmable Reinforcement Learning Agents." PhD thesis, U.C. Berkeley, 2003.

[7] S. B. Thrun and A. Schwartz, „Finding structure in reinforcement learning." In Advances in Neural Information Processing Systems, volume 7, The MIT Press, 1995, pp 385–392.

[8] A. K. McCallum, "Reinforcement Learning with Selective Perception and Hidden State." PhD thesis, University of Rochester, Rochester, NY, 1996.

[9] B. Hengst, "Discovering Hierarchy in Reinforcement Learning", PhD thesis, Computer Science and Engineering, University of New South Wales, Sydney Australia, 2003.

[10] L. A. Jeni, Z. Istenes, P. Korondi, H. Hashimoto, "Mobile Agent Control in Intelligent Space using Reinforcement Learning," in Proc. 7th International Symposium of Hungarian Researchers on Computational Intelligence (HUCI'06), 2006, pp. 201–210.

[11] P. Korondi, H. Hashimoto, "INTELLIGENT SPACE, AS AN INTEGRATED INTELLIGENT SYSTEM," Keynote paper of International Conference on Electrical Drives and Power Electronics, Proceedings, 2003, pp. 24-31.

[12] P. T. Szemes, "Human Observation–based Motion Control Strategies in Intelligent Space," PhD Thesis, Tokyo, 2005.

[13] K. Morioka, H. Hashimoto, "Color Appearance Based Object Identification in Intelligent Space," The 8th IEEE International Workshop on Advanced Motion Control, 2004, pp. 505-510.

[14] B. Reskó; P. Szemes; P. Korondi; Péter Zoltán Baranyi; H. Hashimoto, "Artificial Neural Network based Object Tracking," SICE Conference, 2004, pp 1398-1403.

[15] Z. Petres, B. Reskó, P. Baranyi, H. Hashimoto, "Biology inspired intelligent contouring vision device in intelligent space," in Proc. of the 6th international symposium on advanced intelligent systems. Yeosu, 2005. pp 865-870.

[16] R. E. Bellman, "Dynamic Programming.," Princeton University Press, Princeton, NJ, 1957.

[17] C. J. C. H. Watkins, "Learning from Delayed Rewards," PhD Thesis, King's College, Cambridge, 1989.

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