Compact Q-Learning Optimized for Micro-robots with Processing and Memory Constraints

Originally published in:
Robotics and Autonomous Systems 48(1), https://doi.org/10.1016/j.robot.2004.05.006

Title of paper:

Compact Q-Learning Optimized for Micro-robots with Processing and Memory Constraints

Authors:

Masoud Asadpour (masoud.asadpour@epfl.ch)

Roland Siegwart (roland.siegwart@epfl.ch)

Address:

Autonomous Systems Laboratory (http://asl.epfl.ch)
Swiss Federal Institute of Technology (EPFL)
CH-1015, Lausanne
Switzerland

Fax number: (+41) 21 693 7807

Corresponding Author:

Masoud Asadpour

Compact Q-Learning Optimized for Micro-robots with Processing and Memory Constraints

Masoud Asadpour and Roland Siegwart

Autonomous Systems Lab (http://asl.epfl.ch)
Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland

Email address: masoud.asadpour@epfl.ch , roland.siegwart@epfl.ch

Abstract. Scaling down robots to miniature size introduces many new challenges including memory and program size limitations, low processor performance and low power autonomy. In this paper we describe the concept and implementation of learning of a safe-wandering task with the autonomous micro-robots, Alice. We propose a simplified reinforcement learning algorithm based on one-step Q-learning that is optimized in speed and memory consumption. This algorithm uses only integer-based sum operators and avoids floating-point and multiplication operators. Finally, quality of learning is compared to a floating-point based algorithm.

Keywords: *Reinforcement Learning; Q-Learning; micro-robots.*

1. Introduction

Swarm Intelligence metaphor [1] has become a hot topic in recent years. The number of its successful applications is exponentially growing in combinatorial optimization [8], communication networks [17] and robotics [14]. This approach emphasizes collective intelligence of groups of simple and small agents like ants, bees, and cockroaches. Small robots are also good frameworks to study biology [12]. Small mobile machines could one day perform noninvasive microsurgery, miniaturized rovers could greatly reduce the cost of planetary missions, and tiny surveillance vehicles could carry equipment undetected.

Miniaturizing robots introduces many problems in physical parts and behavior implementations [6]. The robot parts must have low power consumption. This forces the designer to add parts such as sensors conservatively. Due to simplicity of hardware parts, the control program must handle all additional processing such as noise filtering. The instruction set of processors is reduced. The robot behavior must be coded compactly and efficiently while having limitation on program size, memory and processing speed. Additionally, due to limited power autonomy, there is a serious limitation in long tasks such as learning.

In this paper we describe how to practically tackle on-line learning problem on micro-robots with processing constraints and optimize it in program and memory consumption. The proposed algorithm is then verified on learning of a safe-wandering task using the Alice micro-robots [3].

The next section deals with previous works in micro-robots, reinforcement learning and one-step Q-learning algorithm. The third section discusses the problems happening when applying simplifications to Q-learning and introduces an optimized algorithm in size, processing time, and memory consumption based on integer-calculation and low-level instructions. Section 4 presents the micro-robot Alice and its hardware and software features. In the following section the learning of safe-wandering is described and the experimental results are discussed. The 6th section compares the results to floating-point based algorithm and the last section contains conclusion and future works.

2. State of the Art

In his PhD thesis, Gilles Caprari [6] showed processing power of a micro-robot, which is related to available energy, scales down by L^2 factor (L is length). This drastically limits control algorithm capacity. It therefore forces robot designers to further reduce the calculation power by using 8-bit instead of 16 or 32-bit microcontrollers. As a consequence, we have to accept that the intelligence of micro-robots will be limited. Nevertheless, in connection with an external supervisor (computer, human), an adequate collective approach or enough simplifications, small robots might still be able to fulfill complex learning tasks.

Different learning algorithms have been implemented by researchers on micro-robots. Floreano et al. [9] used evolutionary algorithms in combination with spike neurons to train the old version of the Alice micro-robots for obstacle-avoidance. The spiking neural networks are encoded into genetic strings, and the population is evolved. Crossover, mutation and fitness evaluation tasks are optimized and use bitwise operators. But, since no learning is done during fitness evaluation of a newly generated individual, the training takes too much time (3 hours) even for such a simple task.

Dean et al. [7] applied ROLNNET (Rapid Output Learning Neural Network with Eligibility Traces) neural networks to mini-robots for backing a car with trailers. ROLNNET [11] is a mixture of Neural Networks and Reinforcement Learning. It has been designed for real robots with very limited computing power and memory. Input and output spaces are divided to discrete regions and a single neuron is assigned to each region. Neurons are provided with regional sensitivity through the use of eligibility traces. Response learning takes place rapidly using cooperation among neighbor neurons. Even if the mathematical formulation is simple, consists of only summation, multiplication and division, the required floting point operations might still cause a processing problem on autonomous micro-robots with limited processing capacity.

Various implementations of micro-robots exist, such as Sandia MARV [2], MIT Ants [15], Nagoya MARS [10], KAIST Kity [13] and ULB Meloe [16]. However, to our knowledge, no learning task has been implemented on them. In this paper we study the potential for optimization and implementation of reinforcement learning on our micro-robots.

2.1. Reinforcement Learning

Reinforcement learning [19] is one of the widely used online learning methods in robotics. With an online approach, the robot learns during action and acts during learning. Supervised learning methods neglect this feature. With reinforcement

learning the learner perceives the state of its environment (or conditions at higher levels), and based on a predefined criterion chooses an action (or behavior). This action changes the world's state and as a result the agent gets a feedback signal from the environment, called "reinforcement signal", indicating the quality of the new state. After receiving the reinforcement signal, it updates the learned policy based on the type of signal, which can be positive (reward) or negative (punishment).

The reinforcement learning method that we use in this work is the *one-step Q-learning* method [20][21]. However, we have to adapt the algorithms in accordance with the robot's limitations. In the *one-step Q-learning* algorithm the external world is modeled as a *Markov Decision Process* with *discrete finite-time* states. After each action, the agent *immediately* receives a scalar "reward" or "punishment".

An action-value table, called Q-table, determines the learned policy of the agent. It estimates the long-term discounted reward for each state-action pair. Given the current state x and the available actions a_i, a Q-learning agent selects action *"a"* with the probability *"P"* given by the Boltzmann probability distribution:

$$P(a_i|x) = \frac{e^{Q(x,a_i)/\tau}}{\sum_{k \in actions} e^{Q(x,a_k)/\tau}} \qquad (1)$$

Where τ is the temperature parameter which adjusts exploration rate of action selection. High τ values give high randomness to selection at the beginning. The exploration rate will be decreased when Q-values increase gradually, and make exploitation more favorable at the end.

After selecting the favorite action based on the probability distribution, the agent executes the action, receives an immediate reward r, moves to the next state y, and updates $Q(x,a)$ as follows:

$$Q(x,a) \leftarrow (1-\beta)Q(x,a) + \beta\,(r + \gamma\, V(y)) \qquad (2)$$

Where β is the learning rate, $\gamma (0 \le \gamma \le 1)$ is a discount parameter and $V(x)$ is given by:

$$V(y) = \max_{b \in actions} Q(y,b) \qquad (3)$$

Q is improved gradually and the agent learns to maximize the future rewards.

Studies by Sutton [19] showed that convergence of reinforcement learning can be guaranteed. In all of the above and other used formula, numbers are floating-point numbers or at least need floating-point operations, so no discretization or interpolation is applied. In the next section we show what happens to the convergence when the numbers are limited to integers.

3. The Compact Q-Learning Algorithm

In order to implement Q-learning on the micro-robot Alice, we need a simplified Q-Learning algorithm that is able to cope with the limited memory and processing resources and by the restricted power autonomy. Thus we propose a new algorithm based only on integer operations.

3.1. Integer vs. floating point operators

Floating-point operations take too much processing time and program memory. For the sake of comparison, in Table 1, we have listed the number of instructions generated by our C compiler (PCW Compiler, from Custom Computer Services Inc.) and the average execution time for four floating-point operations: *a=b+c*, *a=b-c*, *a=b*c*, and *a=b/c*, and we compared them to integer-based operations. For each operator instance (except for integer sum) there is a call overhead for preparing the registers and copying the results back to memory, and a function execution cost.

Call overhead takes both processing time and program memory for every instance of operator. Function executions need program memory one time but takes processing time for every operator occurrence. Therefore, we prefer to use only integer sum operators since they have no call overhead, require just a few instructions and run very fast. Moreover, we prefer to use unsigned operations to save memory bits, ease computations and reduce overflow-checking.

3.2. Q-Learning problems with Integer operators

To our knowledge, all reinforcement learning algorithms deal with real numbers at least in the action selection mechanisms. The proofs for convergence are valid when numbers are real. In this section we discuss some problems that happen when trying to switch to integer numbers.

The first problem rises in the Boltzmann probability distribution (1). This formula is time-intensive because of computation of powers of e. It also needs a float-type memory cell (generally 4 bytes) to be assigned to the probability of each action, since they will be used then to select an action accordingly. So, the action selection mechanism needs revision.

From now on, let us assume a typical configuration and describe problems with integer operations. Assume at the beginning, Q cells are initialized to c. Since the Q-values must be of type integer, they must be incremented or decremented by one (not a fraction). Applying these conditions to (2) and assuming $\beta=m/n$, $\gamma=p/q$, where m, n, p, and q are integer numbers (to ease integer operations), it is straightforward to show that the reward value at the beginning must be at least:

$$\lceil n/m + c(q-p)/q \rceil = \lceil 1/\beta + c(1-\gamma) \rceil \qquad (4)$$

to effect the new Q-value; otherwise the table remains unchanged and learning task will not converge. Since a typical learning rate is a number around 0.1, the reward value must be at least 10. Also the reward value should be increased according to Q-value increase (see the *c* factor in (4)). This implies that the reward might need too many bits.

Furthermore, in some learning tasks there are many intermediate state-actions that do not come with any reward but lead the learner to near-goal or goal states. In (2) the responsibility of the $\gamma\, v(y)$ term is to increase the value of such state-actions. It can be shown that the value of *v(y)* must be at least:

$$\lceil nq/mp + cq/p \rceil = \lceil 1/\beta\gamma + c/\gamma \rceil \qquad (5)$$

For a typical $\beta=0.1$ and $\gamma=0.9$, *v(y)* must be at least 12 to increment the Q-value by one i.e. it takes a long time to build action chains from initial states to goal states.

Also, let's take *v(y)* out and consider only rewards around zero, or punishments less than or equal to zero. There will be a decrement in the Q-value because of the integer division operator(Remember that e.g. 9/5 =1 in integer division) i.e.:

$$Q_{new}(x,a) = \lfloor ((n-m)Q(x,a) + mr)/n \rfloor \approx \lfloor (n-m)Q(x,a)/n \rfloor \quad (6)$$

Which is less than or equal to Q-1. While, it is better in this case to leave the Q-value unchanged.

Also assume, after some learning episodes, the value of a cell in the Q-table increases to a high integer value; since Q-values are incremented by one this case easily happens. If so, the Q-values in the previous state-chain will be increased by a big integer number, which results in overflow in all states rapidly. Then even a big punishment cannot decrease the Q-values and the learning process gets caught in a local maximum.

3.3. The proposed algorithm

Based on the problems described in the previous section we propose a very simple algorithm dealing with only unsigned integer summation. We assume that Q-values are unsigned integers and have a minimum value of zero.

The probability assignment formula is changed to *Roulette Selection* as following:

$$P(a_i \mid x) = \frac{Q(x,a_i)+1}{|Actions| + \sum_{k \in Actions} Q(x,a_k)} \quad (7)$$

Where $|Actions|$ is the size of action set. Q-values are summed by one so that zero-valued actions have a small positive probability. Roulette Selection method has been widely used in Reinforcement Learning and Genetic Algorithm.

In order to get and idea about the simplification, assume that there are 10 available actions in a state, 9 of them have zero values and the value of the 10th action changes. The assigned probability by the two selection methods to the 10th action is shown in Fig.1. Note that for non-positive values, the Boltzmann function assigns small and very close probabilities, around 0.1. The Roulette function, while dealing only with non-negative numbers (and therefore saving one bit for sign), assigns the same 0.1 probability to non-positive actions. For the positive part, comparing the shape of the curves shows they are similar enough for our purpose: incremental and logarithmic.

The Roulette Selection is capable of adjusting the exploration and exploitation rate during the learning. At the beginning randomness is high since all Q-values are nearly equal and the probabilities are very close. But at the end, some Q-values raised and then big value of the sigma term makes probability of non-efficient actions close to zero, making more exploitation possible.

For implementation we can simplify the action selection mechanism even more. First, a uniform random number between 0 and $|Actions| + \sum_{k \in Actions} Q(x,a_k)$ is generated. The random number then is compared to the partial sum $\sum_{k \in \{1..i\}} (Q(x,a_k) + 1)$ in a loop starting from the first action i.e. i=1. The first action, for which the random number is less than or equal to the partial sum is selected. So there is no need to divide numbers to form probabilities chain in (7).

The policy update formula is changed to:

$$Q(x,a) \leftarrow Q(x,a) + r + f(\mathrm{x},a,y) \tag{8}$$

Where r is the positive or negative reinforcement signal and $f(x,a,y)$ is defined as following:

$$f(x,a,y) = \begin{cases} \gamma(v(y)) & if\ Q(x,a) < \theta_Q \\ 0 & otherwise \end{cases} \tag{9}$$

Where θ is a threshold, $v(y)$ is the same as (3), and γ is the discount function that depends on $v(y)$ and could be implemented via conditional operators without use of multiplication (e.g. if $v(y)>32$ then $\gamma(v(y))=2$). Using the function f limits the unsatisfactory effect of high-value Q-cells discussed earlier.

We propose to use the reinforcement signal r as followed:

$$r = \begin{cases} 1 & reward \\ 0 & don't\ care \\ -1 & punishment \end{cases} \tag{10}$$

because it can be handled by increment and decrement operators, adapted to low-level programming. To work with unsigned numbers, reinforcement signals could be shifted up to 2, 1, 0 and then decremented one unit when adding to Q-value.

A drawback of the proposed algorithm is the lack of the learning rate, but we do not have any better choices since each reward must result in a Q-value increment as described previously. In order to decrease negative effects of the missing learning rate we have to scale down r and γ as much as possible. Otherwise we have to add multiplication and division operators (or in the simplest case, shift operators) to compute $\beta(r+f(x,a,y))$.

4. The Alice Micro-Robot

The Compact Q-Learning Algorithm has been tested for a safe-wandering task with the micro-robot Alice, equipped with a PIC micro-controller. Alice (Fig.2) is one of the smallest (22 x 21 x 20 mm) and lightest (5 gr) autonomous mobile robots in the world [3]. In its basic configuration it has two bi-directional watch motors for locomotion (up to 40 mm/s), four active infrared proximity sensors, a micro-controller, a NiMH rechargeable battery, and an IR TV remote receiver for communication, all mounted on PCB and installed in a plastic frame. . The power consumption has been highly minimized to 12-17 mW, providing high energetic autonomy of up to 10 hours. This makes the robot suitable for collective robotics and learning experiments. However, the autonomy can be increased by adding an extra battery, if required.

Alice is a programmable and modular robot and a number of modules can be added to it, such as a linear camera, wireless radio or IR communication, touch sensors, interface to personal computers, and recharging pack. Thanks to its programmability and interface with personal computers, Alice has been used in various research [5] [3] and educational projects [4].

A PIC16F877 micro-controller from Microchip(R) is the central for controlling of the robot's behavior. It is an 8 bit RISC (Reduced Instruction Set Computer) micro-controller and has only a set of 35 basic instructions, including bit-wise sum and conditional and unconditional branch operations. It has 8K x 14-bit words flash program memory and 368 x 8-bit RAM data memory. It directly drives the two low-power watch motors through 6 pins and reads the values of the proximity sensors through analog-digital converters. To save energy, the clock speed is set to 4 MHz. Each instruction takes four clock-cycles, so each instruction takes one micro-second (except for jumps which require two instructions).

An important percentage of the memory resources is assigned to manage the sensors and motors. The software core is composed of a simple real-time operating system, which handles different time-critical tasks:

1- Communication with a TV remote controller to receive remote commands from the operator or an external computer. Currently there are four commands for the supervision of the learning process:

- Start learning
- Stop learning and behave based on the learned policy (for evaluation),
- Save the learned policy to EEPROM so that it can be downloaded after learning, and
- Load the saved policy from EEPROM and resume the learning task whenever the task is too long to be completed in one battery life cycle

2- Proximity sensor reading and state detection

3- Action selection

4- Control of right and left motors to run the selected action correctly

5- Policy update while learning, and

6- Computing statistics and quality measures and writing them to EEPROM for later evaluation purposes.

5. Learning Safe-Wandering Behavior

The learning task is safe-wandering in two X- and H-shape mazes, shown in Figs. 3 and 4. The H-maze is composed of very narrow parts and has a complex shape. The cross-maze is simpler and walls are at larger distance to each other. The task of the robot is to wander in the maze while having a preference to move forward, without hitting the walls.

The robot states are defined based on the four proximity sensor values at the front, front-left, front-right and rear side of the robot. Each sensor value corresponds to one bit in the state bits, and a threshold is defined for value of the sensors to show presence or absence of obstacles. The number of states is 16, but it can be reduced to 15 since no state exists where obstacles surround the robot in all directions.

In order to reduce the size of the Q-table and save memory, we chose only three actions for the robot: move forward, turn right, and turn left with maximum possible speed. Each cell of the Q-table has one-byte length (15x3=45 bytes totally) and

the values range from 0 to 240. The purpose of setting the maximum value to 240 is to avoid overflows in add or subtract operations. The reward function is defined as followed:

$$r = \begin{cases} 1 & straight\,moves \\ -1 & hitting\,the\,wall \\ 0 & otherwise \end{cases} \qquad (11)$$

We defined another threshold to detect hitting of the walls, so actually the robot may receive punishment without hitting any wall, only because it is very close to a wall. The threshold for the discount function (θ_Q) is 100 and the discount function is defined as follows:

$$\gamma(v(y)) = \begin{cases} 2 & v(y) \geq 64 \\ 1 & 64 > v(y) \geq 32 \\ 0 & otherwise \end{cases} \qquad (12)$$

The learning cycle is 200 ms (5 Hz). The program reads the sensor values, and then, after state detection and action selection, sets “pause” of the wheels (we do not change motor velocity directly in order to simplifies the motor control)). During each learning cycle, the motors are controlled in one of their six operation phases. At the end the program detects the next state, computes the reward and updates the policy.

Every minute, the sum of the received rewards during this period is written into the EEPROM so that it can be downloaded after learning for drawing learning curves. Saving the learned policy on EEPROM is on-demand based on the observer decision.

Also, it is possible to stop the learning and command the robot to behave according to the learned policy. If so, the robot selects actions with the maximum value in each state. If the robot behavior is acceptable, the policy can be saved and if learning is incomplete it can be resumed from the last stopping point.

A sample of the learned safe-wandering behavior is shown in fig.3. The figure shows that the Alice robot has learned to avoid obstacles and move straight efficiently. The robot moves forward and turns when sees a wall.

The graph in fig.5 shows the changes of the received rewards during a 20 minutes learning experiment in the X-maze. The experiment could be stopped at 7 minutes but we intend to compare them with a floating-point based algorithm. The maximum received reward during this time could be 300 i.e. if the robot moves forward during all 300 actions. But some times the robot is forced to turn due to the specific configuration of environment and so the maximum could not be attained. However, you can see that after 7 minutes the learning curve converges and oscillates around 190. Oscillations in the received rewards at some steps are because the robot is in different locations of the maze.

A sample of the learned path and learning curve for H-maze is drawn in Figs.4 and 6 respectively. The experiment takes 30 minutes to complete. The sum changes from –213 at initial steps to 33 at the final steps. The minimum peak at the 5^{th} point is because of a new unforeseen situation in the maze where the robot does not know how to deal with.

The resulted behavior is visually pretty good; however, due to special configuration of walls, the robot has to change the direction many times and cannot reach the maximum. The narrow space between the walls is a very important limit for movements in H-maze. The robot states show only presence or absence of wall (0 or 1) and do not deal with more details. As a result, the robot goes beyond the threshold distance several times (especially at sidewalls) and receives punishment; but actually it does not hit the wall and corrects the path by turning to the opposite side.

These two learning tasks converge in 7 and 30 minutes respectively. Comparing to 10-hour autonomy of the Alice robots, it seems that implementation of more complex and time-consuming learning tasks is feasible.

6. Integer vs. Floating-point

To measure the introduced error due to the simplification of the algorithm, we compared the proposed algorithm with the floating-point based algorithm (regular Q-learning algorithm). We tried combination of different learning rates from 0.01 to 0.1 (0.01 steps) and from 0.1 to 0.5 (0.1 steps), with different temperature parameters from 0.1 to 1 (0.1 steps) on safe-wandering task in the X-maze. Finally we found that the three best results are obtained by setting the learning rates to 0.1, 0.2, 0.3 and temperature parameter to 0.5. The discount parameter is fixed to 0.9. Each learning experiment lasted 20 minutes. Then learning is stopped and the robot behavior is tested for 10-minute. To reduce the heterogeneity of the learning episodes, the robots are placed in the same initial position at each learning or testing phase.

We compare the methods in terms of four measures: *Convergence rate*, *optimality*, *memory,* and *program size*. The average received rewards during learning gives an indirect indication of convergence rate. If the learning procedure converges early then this average should be higher. Also, the average received rewards during test phase gives a measure to compare optimality of results. If any learning algorithm finds an optimal policy then its average should be higher in this phase.

Learning measures

The graphs in Figs. 7 and 8 show the average received rewards during each minute in learning and test phases, respectively. Among the three float-based experiments, β=0.1 is the most conservative in terms of learning rate, therefore its average during learning is less than the others (figure 7). However, the test results show that it was able to find the best policy (figure 8). On the other hand, while β=0.3 has the best results during learning phase (163.5), it remains almost the same (163.8) in test phase and is the worst among all. In this case convergence happens too early in a local maximum and the policy remains there for a while.

Comparing these results to the integer-based algorithm, one can see that the time for convergence is between the β=0.3 and β=0.1 case of the float-based experiment. Both, the convergence rate and learning quality is near the second best results of float-based case. Therefore it represents a good trade-off between convergence rate and finding the global optimum.

Implementation measures

Moreover, regarding the implemented program, the whole learning program for the integer-based case takes 42 % of data and 44% of program memory; therefore a large volume of them remains empty. It is important to know that the operating system took already about 29% of data and 19% of program memory. So, the learning program plus the save, load, and evaluation procedures occupy only **13%** of data and **25%** of program memory. Recall that only 3 floating-point operations (i.e. +, *, and /) fill 8.8% of program memory (see Table 1). On the other hand, the floating-point based program plus operating system takes 89% of data and 83% of program memory (**60%** and **64%** without OS). Roughly speaking, since data memory size depends mainly on Q-table representation, using one-byte integers for Q-values reduces memory consumption by a factor of three in the comparison with the 4-byte floating-point representation.

7. Conclusion and future works

In this paper we described the problems faced in programming micro-robots for learning tasks. The main challenges with micro-robots are limitations in power autonomy, processing power and memory (both in program and in data memory). These limitations obliged us to avoid floating-point operations and thus to develop a simplified learning algorithm, rely mainly on integer-based additions and subtractions.

We proposed a simple and fast reinforcement learning algorithm optimized for data and program memory consumption. It was implemented and verified on a safe-wandering task with two test environments. The fast convergence of algorithm made it possible to save at least 95% of power autonomy, while its optimality remains competitive to Q-learning results. The small number of required instructions leaves around 60% of both, program and data memory unused. Therefore, it offers potential for more time-critic and complex learning behaviors.

Convergence of the proposed algorithm has been demonstrated by experiments. However mathematical analysis and prove has still to be made. For future works we plan to test the algorithm on other, more complex tasks, especially collective behaviors and cooperative learning among groups of robots.

Acknowledgment

We would like to thank the two unknown reviewers for their remarkable comments and our colleague Gilles Caprari for his valuable works on the Alice robot and its operating system. This work was supported by the EU project IST-2001-35506, *Artificial Life Control in Mixed-Societies (LEURRE)*.

8. References

[1] E. Bonabeu, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, 1999.

[2] R.H. Byrne, D.R. Adkins, S.E. Eskridge, J.J. Harrington, E.J. Heller, J.E. Hurtado, Miniature mobile robots for plume tracking and source localization research, Journal of Micromechatronics, VSP, vol.1, No. 3, pp. 253-262,2002.

[3] G. Caprari, P. Balmer, R. Piguet, and R. Siegwart, The autonomous micro robot Alice: A platform for scientific and commercial applications, In Proceedings of the 9th International Symposium on Micro-mechatronics and Human Science, 231–235, 1998.

[4] G. Caprari, K.O. Arras, and R. Siegwart, The autonomous miniature robot Alice: From prototypes to applications, In: Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2000), IEEE Press, 793–798, 2000.

[5] G. Caprari, K.O. Arras, and R. Siegwart, Robot navigation in centimeter range labyrinths. In U. Rückert, J. Sitte, and U. Witkowski (eds.), Proceedings of the 5th International Heinz Nixdorf Symposium: Autonomous Mini-robots for Research and Edutainment, AMiRE 2001.

[6] G. Caprari, Autonomous Micro-robots: Applications and Limitations, Ph.D. Thesis, EPFL, Switzerland, 2003.

[7] F.H. Dean, M. Gini, and J. Slagle, Rapid Unsupervised Connectionist Learning for Backing a Robot with Two Trailers, IEEE International Conference on Robotics and Automation, 1997.

[8] R. Durbin, and D. Willshaw. An Analogue Approach to the Traveling Salesman Problem Using an Elastic Net Method, Nature 326 (1987) 689-691.

[9] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel, Evolutionary Bits'n'Spikes, In: R.K. Standish, M.A. Beadau and H.A. Abbass(eds.), Artificial Life VIII: Proceedings of the 8th International Conference on Artificial Life, MIT Press, 2002.

[10] T. Fukuda, H. Mizoguchi, K. Sekiyama, F. Arai, Group Behavior Control for MARS (Micro Autonomous Robotic System), Proceedings Of the 1999 IEEE International Conference on Robotics and Automation, ICRA99, pp.1550-1555,1999.

[11] D.F. Hougen, Use of an Eligibility Trace to self-organize output, In Science of Artificial Neural Networks II, Proc. SPIE 1966, pp.436-447, 1993.

[12] C. Jost, S. Garnier, R. Jeanson, M. Asadpour, J. Gautrais, and G. Theraulaz, The embodiment of cockroach behaviour in a micro-robot, ISR 2004: 35th International Symposium on Robotics, Paris, 23-26 march 2004.

[13] J.H. Kim, M.J. Jung, H.S. Shim, S.W. Lee, Autonomous Micro-Robot 'Kity' for Maze Contest, Proceedings of International Syposium on Artificial Life and Robotics, pp.261-264, 1996.

[14] C.R. Kube, Collective Robotics: From Local Perception to Global Action, Ph.D. Thesis, University of Alberta, 1997.

[15] J. McLurkin, Using Cooperative Robots for Explosive Ordnance Disposal, MIT AI Lab, 1996. http://www.ai.mit.edu/projects/eod-robots/eod-paper.pdf

[16] Meloe Micro Robots, Active Structure Laboratory, Université Libre de Bruxelles, http://www.ulb.ac.be/scmero/robotics.html#micro

[17] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz., Ant-Based Load Balancing in Telecommunication Networks, Adaptive Behavior 5 (1996) 169-207.

[18] R. Siegwart, C. Wannaz, P. Garcia, and R. Blank, Guiding mobile-robots through the web, In: Workshop Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, Victoria, Canada, 1998.

[19] R.S. Sutton, and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[20] C.J.C.H. Watkins, Learning from Delayed Rewards, Ph.D. Thesis, King's College, 1989.

[21] C.J.C.H. Watkins, and P. Dayan, Q-Learning (technical note), In: Sutton R.S.(ed.), Machine Learning: Special issue on reinforcement learning, vol. 8, 1992.

List of figures:

List of tables:

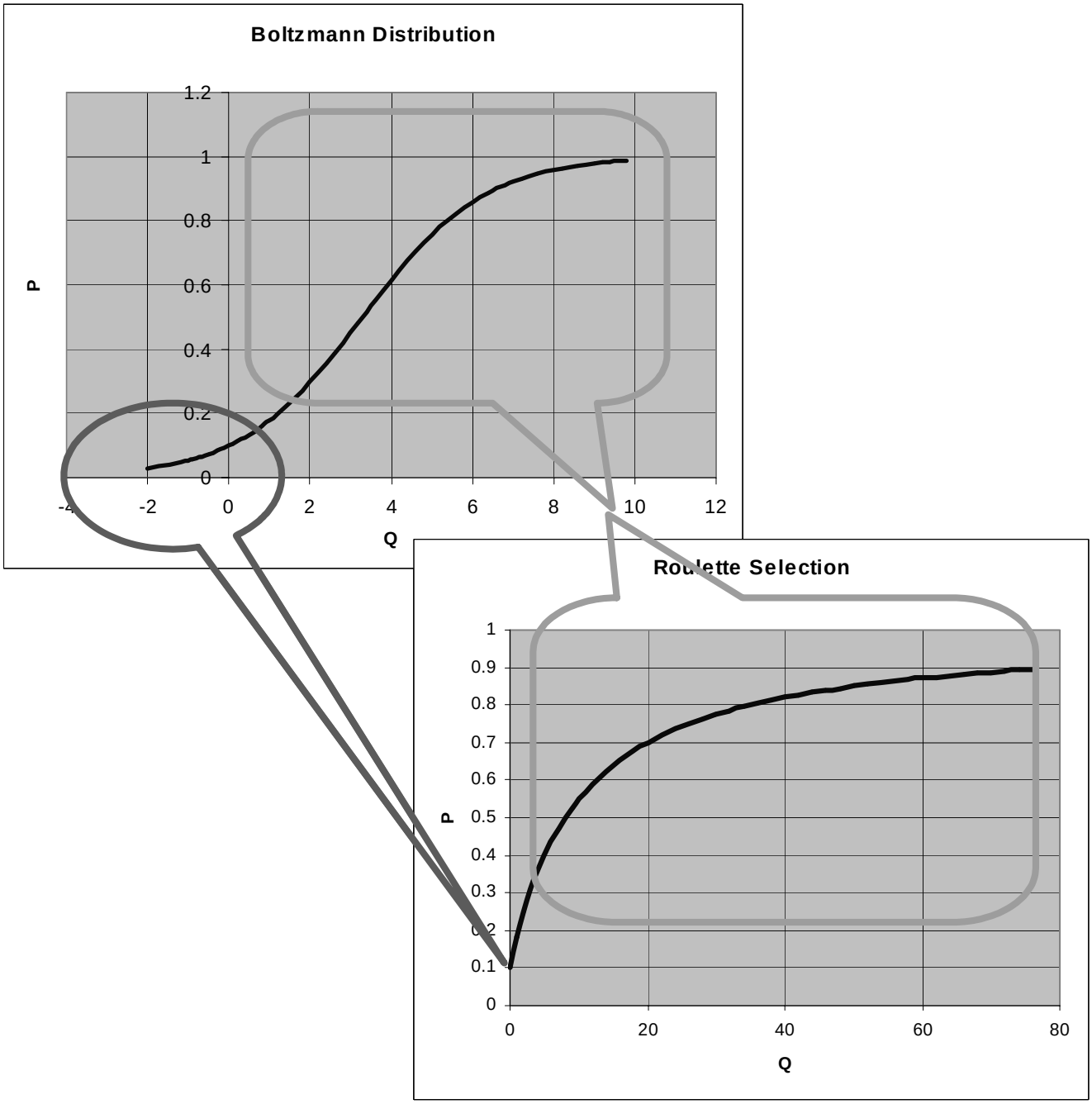


Figure 1- Simplification of Boltzmann Selection to Roulette Selection

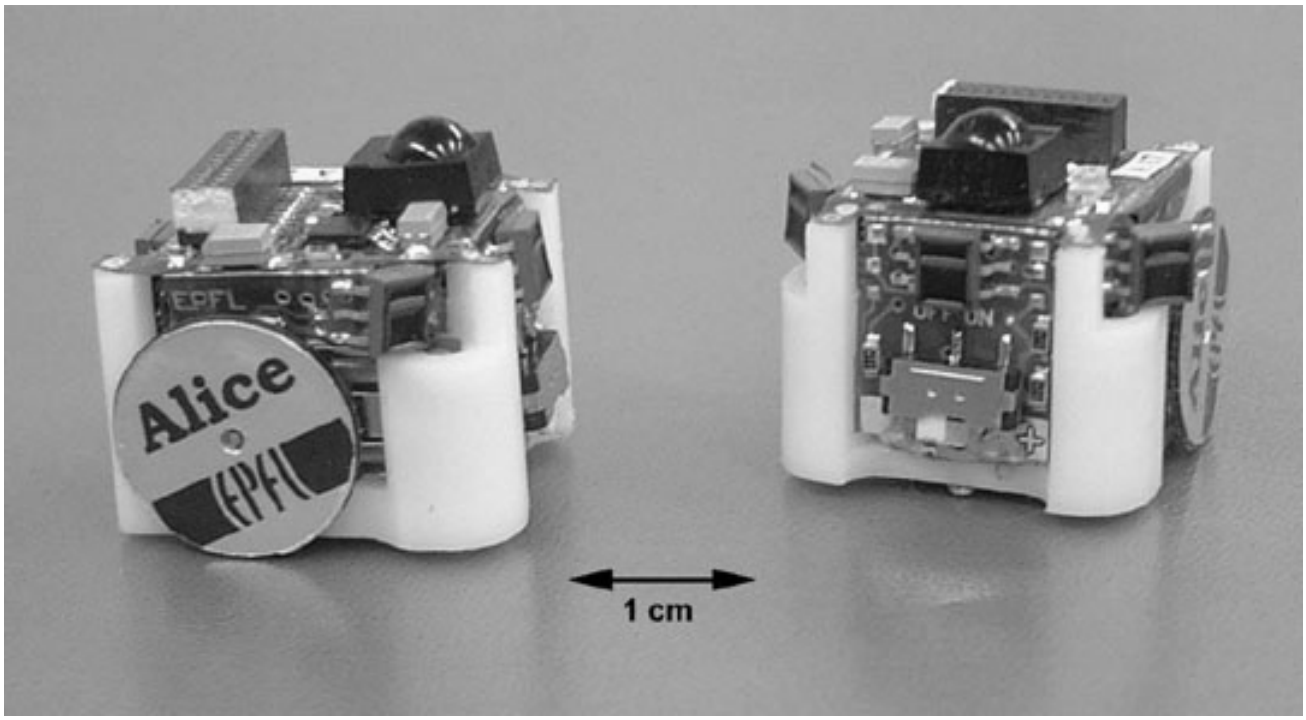


Figure 2: The Alice micro-robot

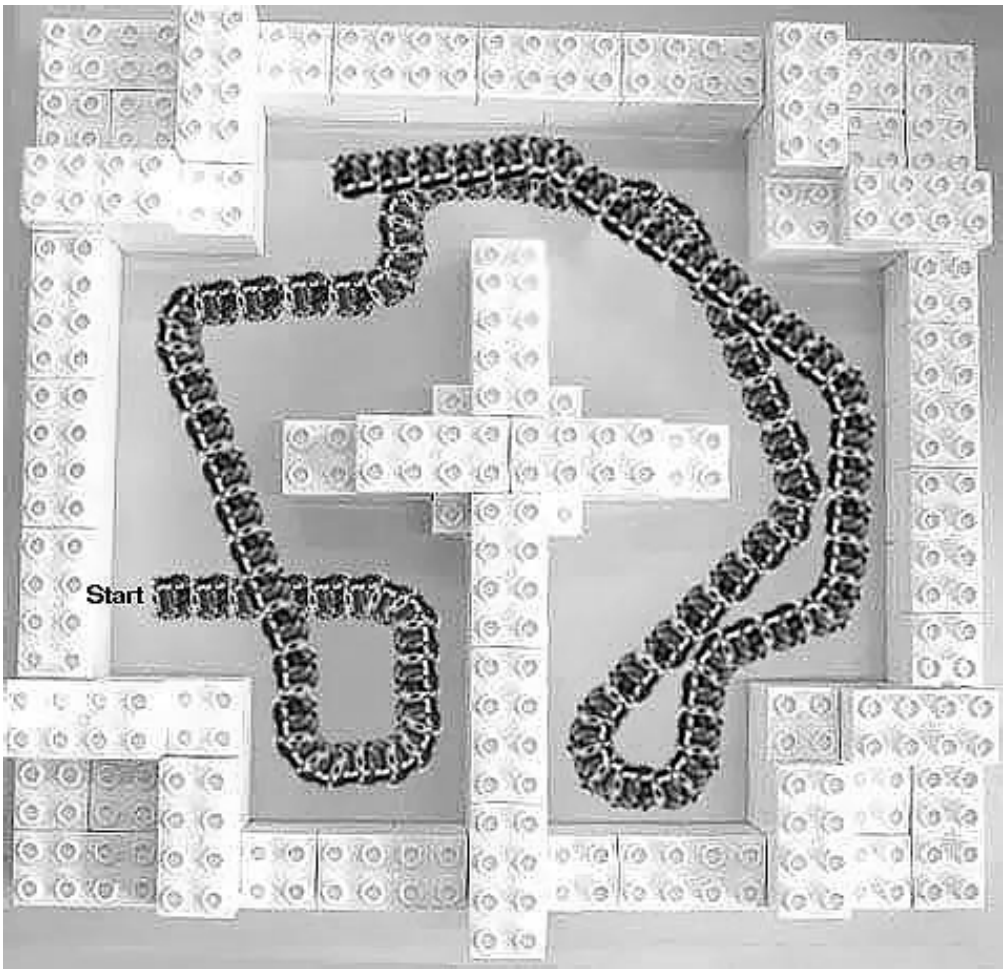


Figure 3- A sample of the learned safe-wandering behavior in X-maze

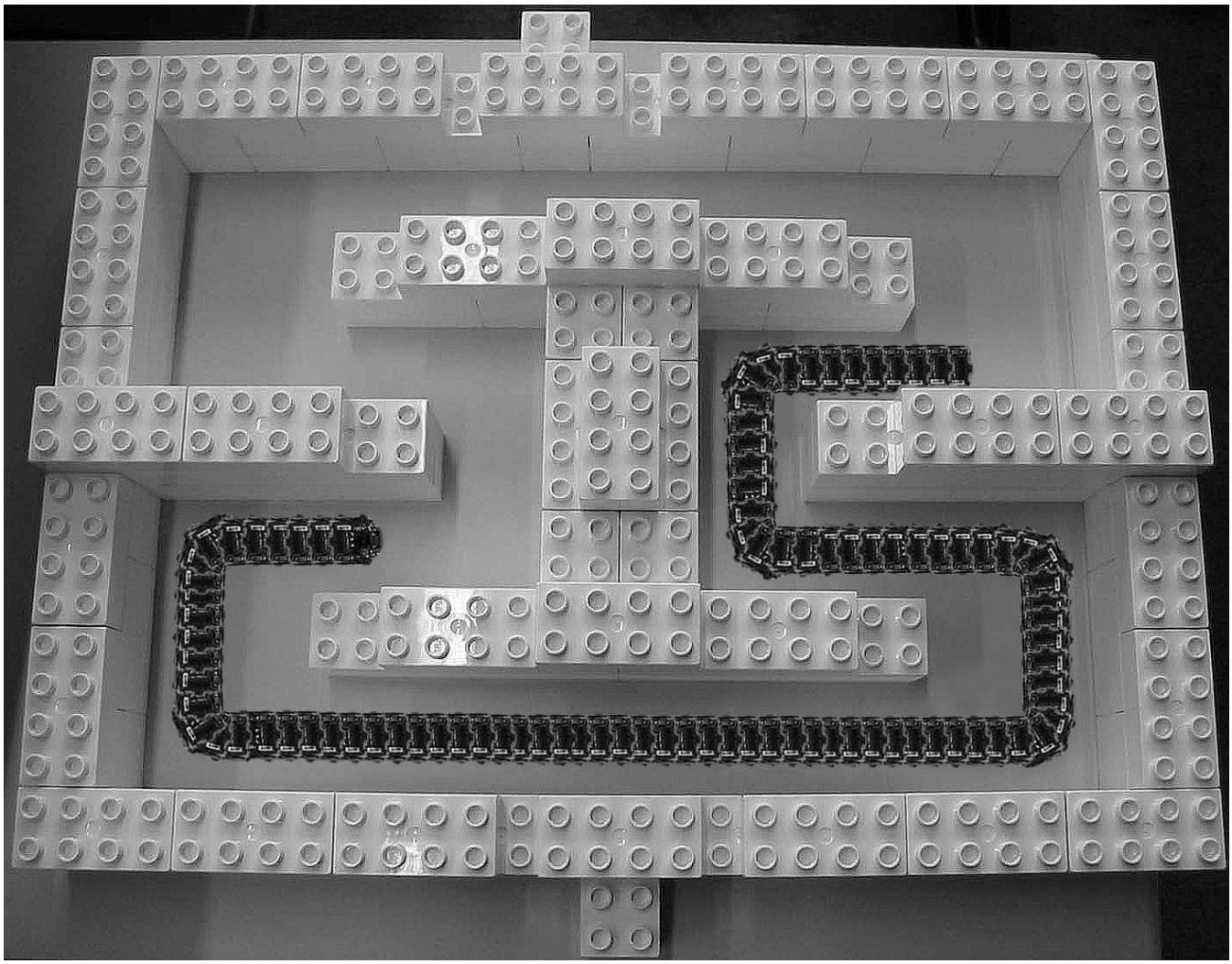


Figure 4-A sample of the learned safe-wandering behavior in H-maze

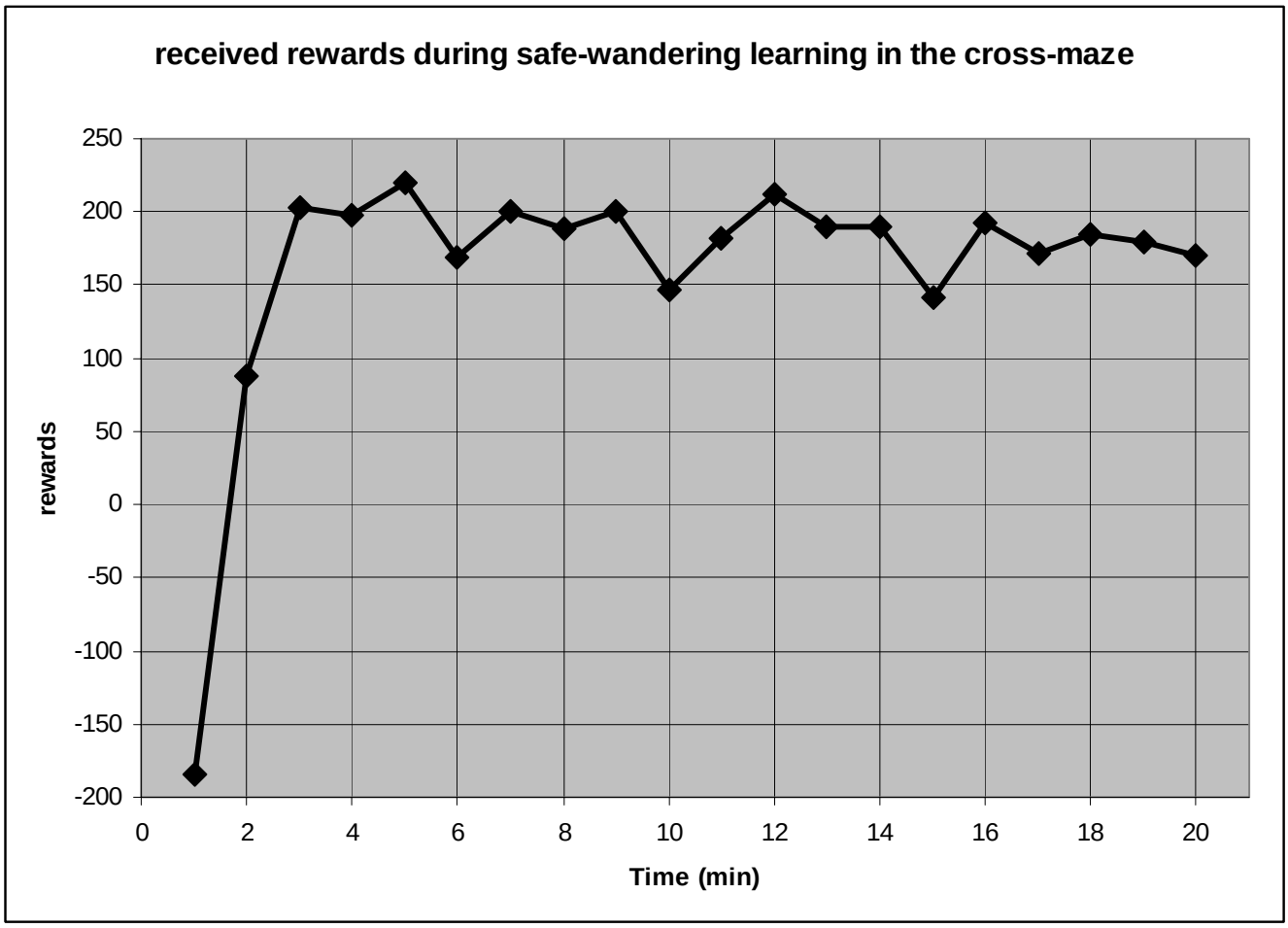


Figure 5- Received rewards during safe-wandering learning in the Cross-maze

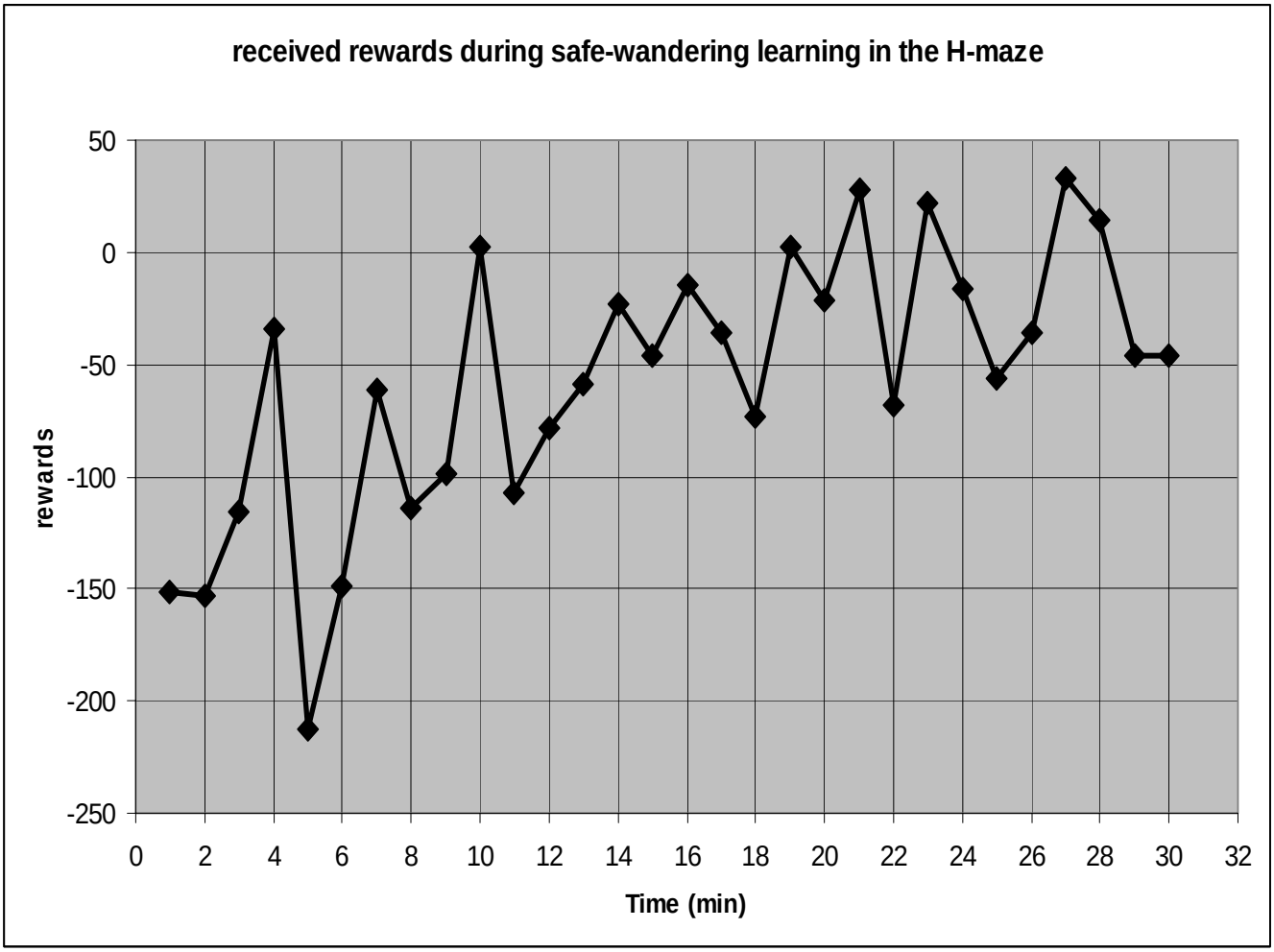


Figure 6- Received rewards during safe-wandering learning in the H-maze

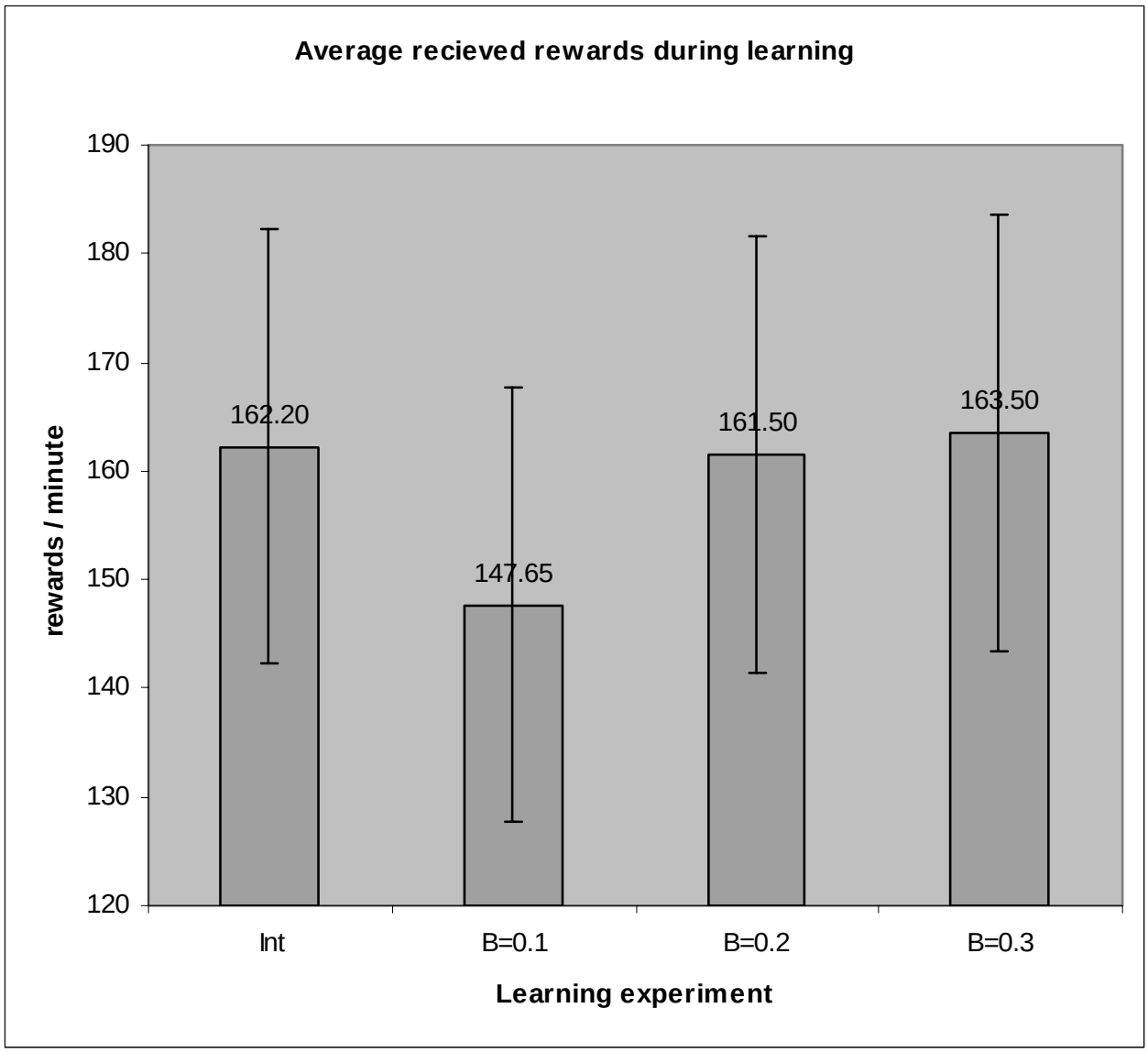


Figure 7- Average received rewards during learning phase

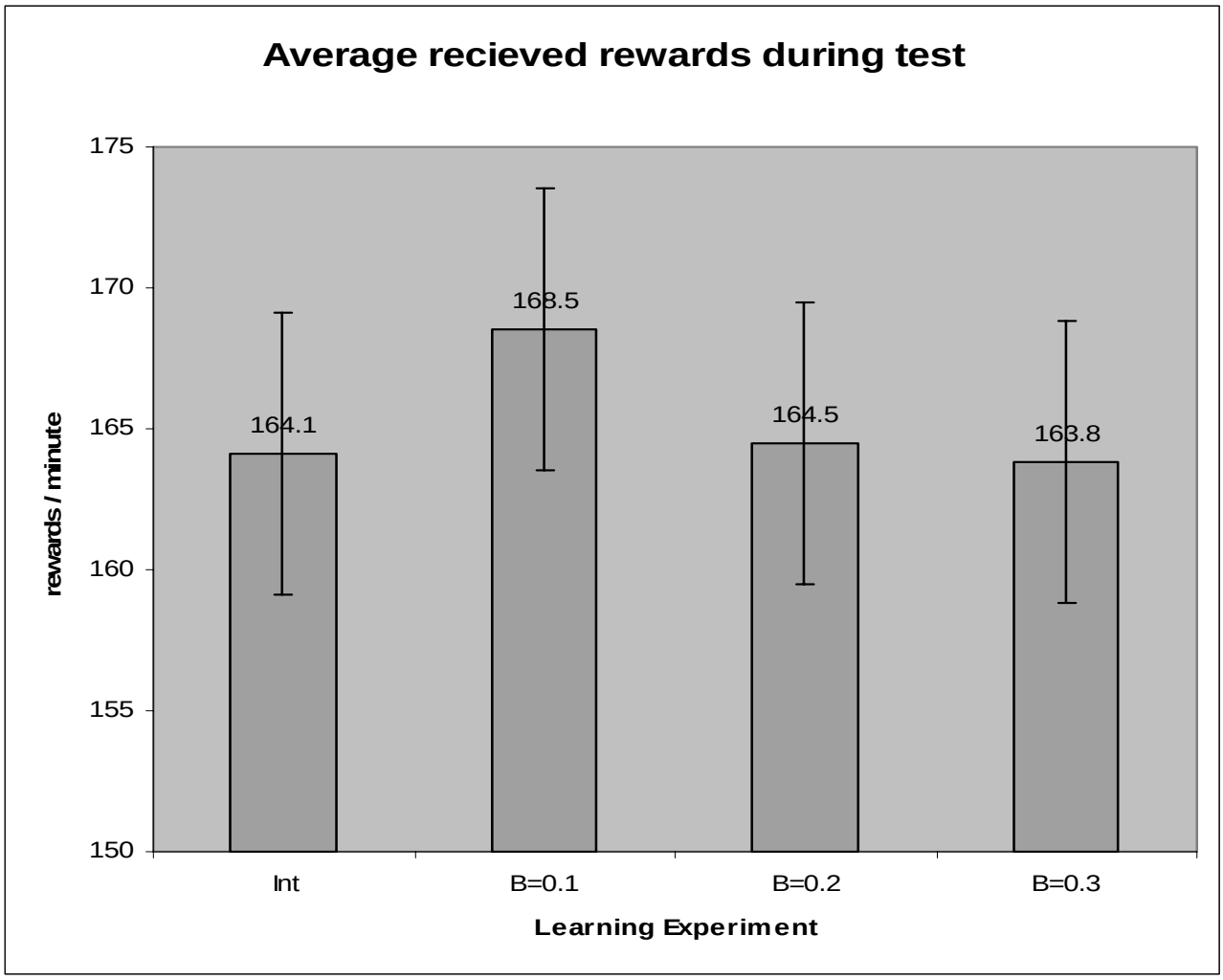
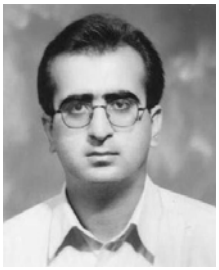


Figure 8-average received rewards during test

Table 1: Comparing floating-point and integer operations on Alice micro-robots

Floating point operator	**# of Instructions**	**Call overhead**	**% of Alice memory for the first call**	**Average Execution Time (us)**
+, -	322*	26	4.25	154
*	119	25	1.76	613
/	204	25	2.8	1121
Total (only for one call to each operator)			**8.81 %**	**1888**
8 bit Integer operator	**# of Instructions**	**Call overhead**	**% of Alice memory for the first call**	**Average Execution Time (us)**
+, -	3	0 (no function call)	0.04	3
*	37	7	0.54	43
/	21	7	0.34	87
Total (only for one call to each operator)			**0.92 %**	**133**

*Float numbers are represented in mantissa-exponent form. The generated code for float + becomes larger than * and / since, in +, the second operand is changed to have the same exponent as the first one.



Masoud Asadpour received his B.Sc. degree in computer software engineering from Sharif University of Technology, Tehran, Iran, in 1997, and his M.Sc. degree in AI and robotics from University of Tehran, in 1999. He has been researcher at Intelligent Systems Research Center, Institute for Studies on Theoretical Physics and Mathematics (IPM), Tehran for two years. Currently he is PhD student at Autonomous Systems Lab, EPFL University, Switzerland. His research interests are cooperative learning, collective robotics and multi-agent systems.

Roland Siegwart (1959) received his M.Sc. ME in 1983 and his Doctoral degree in 1989 at the Swiss Federal Institute of Technology (ETH) Zurich. After his Ph.D. studies he spent one year as a postdoc at Stanford University where he was involved in micro-robots and tactile gripping. From 1991 to 1996 he worked part time as R&D director at MECOS Traxler AG and as lecturer and deputy head at the Institute of Robotics, ETH. Since 1996 he is a full professor for Autonomous Systems and Robots at the Swiss Federal Institute of Technology, Lausanne (EPFL), and since 2002 also vice-dean of the School of Engineering. He leads a research group of around 25 people working in the field of robotics and mechatronics. Roland Siegwart published over 100 papers in the field of mechatronics and robotics, is an active member of various scientific committees and co-founder of several spin-off companies. He was the general Chair of IROS 2002 and he is currently VP for Technical Activities of the IEEE Robotics and Automation Society.