Compact Q-Learning for Micro-robots with Processing Constraints

Masoud Asadpour and Roland Siegwart

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

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 safe-wandering and light following tasks on 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.

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, communication networks and robotics. 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.
Miniaturizing robots introduces many new problems in behavior implementation. The robot parts must have low power consumption. This forces the designer to add parts e.g. sensors conservatively. Due to simplicity of hardware parts, the control program must handle all additional processing such as 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-time tasks such as learning.
The Alice micro-robot [2] is one of the successful micro-robot implementations. To test its capabilities, we demonstrate in this paper the feasibility of implementing an online learning algorithm.
Different learning algorithms have been designed by researchers on micro-robots such as evolutionary algorithms [6] and neural networks (e.g. [5] based on ROLLNET or [6] based on spike neurons). Unfortunately, all of them have problem in online learning, learning time, or task complexity. So it is very hard (if not impossible) to utilize them on autonomous micro-robots with limited power autonomy and acquire complex learning behaviors. We decided to use Reinforcement Learning since it has been one of the most successful learning methods in real robotics.
In this paper we describe how to tackle the learning problem practically and optimize it in program and memory consumption. The next section deals with Reinforcement Learning and one-step Q-learning algorithm. The third section discusses the problems when applying simplifications to Q-learning and introduces an optimized algorithm in size, process time, and memory consumption based on integer-calculation and low-level instructions. Section four presents the micro-robot Alice and its hardware and software features. In the next section the learning of safe-wandering is described and the experimental results are discussed. The sixth section describes extra learning experiments on learning of light following and the last section discusses the results and future works.

2 REINFORCEMENT LEARNING

Reinforcement learning [8] is one of the widely used online learning methods in robotics. With an online approach, the robot learns during action and acts during learning, which is neglected in supervised learning methods. With reinforcement learning the learner perceives the state of its environment (or conditions at high level), and based on a predefined criterion chooses an action (or behavior). The action changes the world's state and as a result the agent is given back 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 is positive (reward) or negative (punishment).
The reinforcement learning method that we use in this paper is the *one-step Q-learning* method [9][10]. 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 receives a scalar "reward" or "punishment".[1] The action value table, the 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:

[1] We do not cover delayed reward.

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

where τ is the temperature parameter which adjusts the exploration rate of the action selection decisions. 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 probable at the end.

After selecting the useful 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, γ $(0 \le \gamma \le 1)$ 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 [8] showed the 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 SIMPLIFIED 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 on integer operations only.

3.1 Integer vs. Floating point Operators

Floating-point operations take too much processing time and program memory. For the sake of comparison, we have listed in Table 1 the number of instructions generated by a C compiler[2] 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 and require just a few instructions. Moreover, we prefer to use unsigned operations to save memory bits, ease computations and reduce overflow-checking.

Table 1: Program memory consumption of floating-point and integer operations on Alice micro-robots

Floating point operator	# of Instructions	Call overhead	% of Alice memory for the first call
+, -	322	26	4.25
/	204	25	2.8
*	119	25	1.76
		Total (only for one call to each operator)	**8.81**
Integer operator	**# of Instructions**	**Call overhead**	**% of Alice memory for the first call**
+, -	3	0 (no function call)	0.04
/	21	7	0.34
*	37	7	0.54
		Total (only for one call to each operator)	**0.92**

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 processing power intensive because of the computation of the powers of *e.* It also needs a float-type memory cell (generally 4 bytes) to be assigned for 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 the difficulties 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 *β=m/n, γ=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:

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

[2] PCW Compiler from Custom Computer Services Inc.

to have an effect on 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 needs many bits.
Furthermore, there are many states and actions that do not give any reward but lead the learner to near-goal states. In (2) the responsibility of γ *v(y)* term is to increase the value of such actions. It can be shown that the value of *v(y)* must be at least:

$$\lceil nq / mp + cq / p \rceil = \lceil 1/\beta\gamma + c/\gamma \rceil \tag{5}$$

For a typical $\beta=0.1$ and $\gamma=0.9$, *v(y)* must be at least 12 to increment the Q-value bye one.
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[3] i.e.:

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

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[4]. The Q-values in the previous state-chain will be increased by a big integer number, which result 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 sum. We assume that Q-values are unsigned integers and have a minimum value of zero.
The probability assignment formula are changed to *roulette selection* as following:

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

where $|Actions|$ is the size of action set. Q-values are increased by one so that zero-valued actions have a small positive probability. This method has been widely used in Reinforcement Learning and Genetic Algorithm.
The formula 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 are raised and then the big value of the sigma term makes the 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 a 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 the randomly selected action with the probability distribution of (7). So there is no need to divide numbers to form probabilities chain in (7).
The policy update formula is changed to:

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

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

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

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 function *f* limits the unsatisfactory effect of high-value Q-cells discussed formerly.
We recommend using reinforcement signal *r,* as following:

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

because it needs very few bits and can be handled by increment and decrement operations 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 lacking 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 the 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 $\beta(r+f(x,a,y))$.

[3] Remember that e.g. 9/5 =1 in integer division
[4] Since Q-values are incremented by one this case easily happens

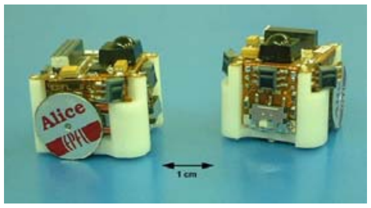

Figure 1: The Alice micro-robot

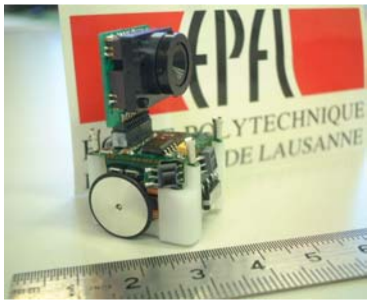

Figure 2: Alice equipped with a camera

4 THE ALICE MICRO-ROBOT

The Simplified Q-Learning Algorithm has been tested on safe-wandering and light-following tasks on the micro-robot Alice, equipped with a PIC micro-controller. Alice (Fig. 1) is one of the smallest autonomous mobile robots in the world [2]. Its high energetic autonomy of up to 10 hours makes it suitable for collective robotics and learning experiments. The autonomy can be increased by adding an extra battery if needed. In its basic configuration it has two motors for locomotion, four active infrared sensors, a micro-controller, a battery, and an IR TV remote receiver for communication.
Alice is a programmable and modular robot and a number of modules can be added to it, such as a camera (Fig.2), long or short radio or IR communication, touch sensors, interface to personal computers, and recharging pack. Table 2 details all the parts and characteristics of the Alice version 2002 used for this experiment. Thanks to its programmability and interface with personal computers, Alice has been used in various research [4] [2] and educational projects [3].
The micro-controller PIC16F877 from Microchip(R), whose entire package occupies 22 x 21 x 20 mm, is the central part of the electronics and control of the robot's behaviors. It directly drives the two low-power watch motors through 6 pins and reads the values of the analog-digital converters for proximity sensor measurement.
The PIC16F877 is an 8 bit RISC (Reduced Instruction Set Computer) micro-controller and has only a set of 35 simple instructions, including bit wise, sum, conditional and unconditional branch operations. To save energy, the clock speed is set to 4 MHz. Each instruction takes four clock-cycles, so each instruction takes 1 us (except for jumps which take 2 us). The specification of PIC16F877 is detailed in table 3.
A high percentage of the memory resources is taken by the management of 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 observer. Currently there are four commands for 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 to continue the learning task in case 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 do the selected action correctly
5. Updating the policy and learning, and
6. Computing statistics and quality measures and writing them to EEPROM for later evaluation purposes.

5 SAFE-WANDERING

The first learning task is safe-wandering in the two cross- 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 the walls have a greater distance to each other. The task of the robot is to wander in the mazes with a preference to forward movement and 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 corresponds to one bit in the state bits and a threshold is defined for the value of sensors to show the presence or absence of obstacles. The number of states is 16 but it can be reduced to 15 since no state exists where all sensors show an obstacle.
To decrease the size of the Q-table and save memory, we choose only 3 actions for robot: move forward, turn right, and turn

Table 2:Features and components of the Alice robot

Dimensions	22 x 21 x 20 mm
Weight	5 g
Velocity	Up to 40 mm/s
Power consumption	12 - 17 mW
System autonomy	From 2 to 10 hours
Mechanical structure	Plastic frame and PCB
Motors	2 bi-directional watch motors
Motion	2 wheels on the minute axis
CPU	PIC16F877 @ 4 MHz
Energy source	NiMH rechargeable battery
Sensors	4 infrared proximity sensors
Communication	IR TV remote receiver(one way)

Table 3: Specifications of PIC16F877 micro-controller

Flash Program Memory	8 K x 14-bit words
RAM Data Memory	368 x 8 bits
EEPROM Data Memory	256 x 8 bits
I/O Ports	5
Timers	3
A/D modules	10 bit, 8 in/out channels
Instruction set	35
Operation Frequency	DC-20MHz
Power consumption	Less than 0.6 mA @ 3V, 4.0 MHz

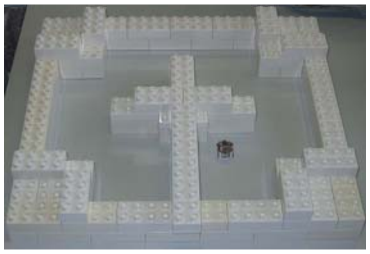
Figure 3 -The Cross-maze

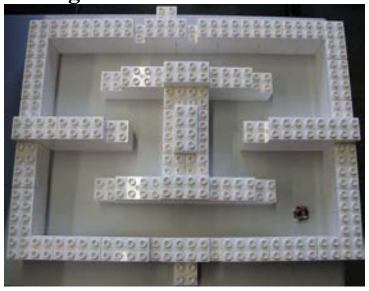
Figure 4 -The H-maze

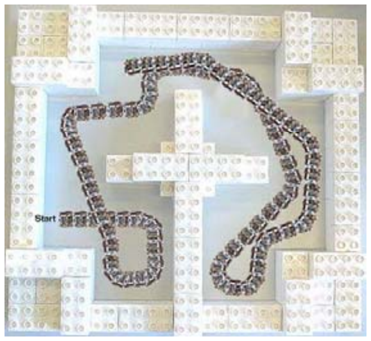

Figure 5- A sample of the learned safe-wandering behavior

left with maximum possible speed. Each cell of Q-table has one-byte length (15x3=45 bytes totally) and the values ranges from 0 to 240. Setting a maximum value is to avoid overflows in add or subtract operations. The reward function is defined as following:

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

We define another threshold to detect hitting 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 discount function (θ_Q) is 100 and the discount function is defined as follows:

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

The learning cycles are repeated every 200 ms. The program reads the sensor values, and then after state detection and action selection sets the velocities of the wheels[5]. During the 200 ms motors are controlled in one of the six phases to do the selected action correctly. At the end the program detects the next state, computes the reward and updates the policy.
Sum of the received rewards is written every 15 seconds in the EEPROM so that it can be downloaded after learning for drawing the learning curve. Saving the learned policy on EEPROM is on-demand dependent upon evaluator 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 could be saved and if the learning is incomplete it can be resumed from the last stopped point.
A sample of the learned safe-wandering behavior is shown in fig. 5. The figure shows that the Alice robot has learned to avoid obstacles and move straight efficiently. The robot insists on moving forward and turns just near the walls.
The graph in fig.6 shows changes of received rewards during 15 minutes of learning experiment in the cross-maze. The experiment could be stopped at 7 minutes but we intend to show stability of the results. Each point in graph denotes sum of the received rewards in 15 seconds periods. The maximum received reward during this time could be 75 i.e. if the robot moves forward during all 15 seconds. 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 converges and the curve vibrates mainly between 20 and 40. Vibrations in the received rewards at some steps are because the robot is in different situations of the maze.
The graph for H-maze is drawn in fig. 7. In this graph the experiment takes 30 minutes and each point shows the sum of the received rewards throughout each minute. The sum changes from –213 at the introductory steps to 33 at the final steps. The minimum peak at the 5th 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 get many rewards. The narrow space between the walls is a very important limit for movements in H-maze. The robot states only show presence or absence of wall (0 or 1) and do not deal with more details. As

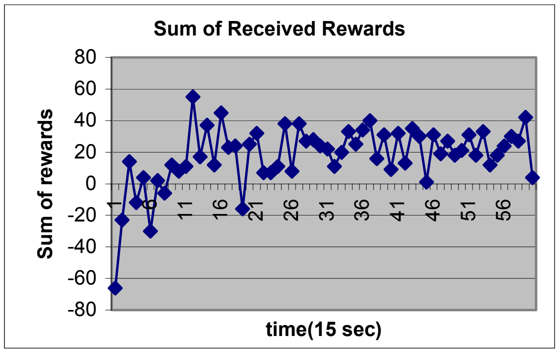

Figure 6- Changes in sum of the received rewards during each 15 seconds in the Cross-maze

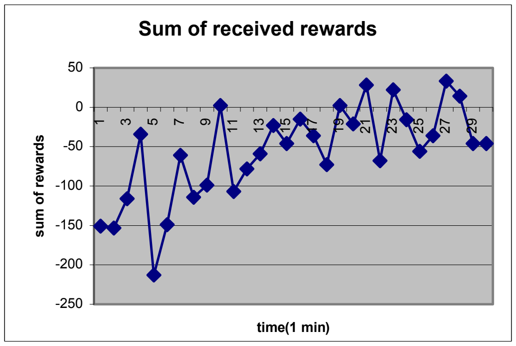

Figure 7- Changes in sum of the received rewards during each minute in the H-maze

[5] Actually we do not set the velocity, the control of wheels is more complex and is done by setting pause parameters.

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.
The two learning tasks converge in 7 and 30 minutes respectively. Comparing to 10-hour autonomy of the Alice robots, it seems that an implementation of more complex and time-consuming learning tasks is possible. Moreover, the whole learning program plus the operating system takes 70 % of data and 33% of program memory. Therefore a large volume of them remains empty. It is important to know, that the operating system takes already about 37% of data and 20% of program memory. So the learning program plus the save, load, and evaluation procedures occupy only 13% of program memory. Recall that only 3 floating-point operations (i.e. +, *, and /) consume 8.8% of program memory (see Table 1).

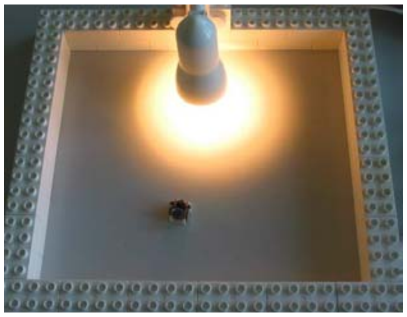
Figure 8- Light following task

6 Light-following

To test reliability of the simplified algorithm, another learning task is implemented and verified. In this task, the Alice learns to follow a moving light source, which is moved manually around a simple rectangular maze with a similar speed as Alice (fig.8). The IR sensors are used to measure the ambient light close to the robot. The state of the robot is denoted by the most illuminated side (totally 4 states). Actions of robot are the same as in safe-wandering (3 actions). The robot is rewarded when it faces toward the brightest side and gets punished otherwise.
The learning curve drawn in fig.9 shows effectiveness of algorithm. This task is simpler than the former and takes less than 3 minutes to converge. The learned behavior is nearly the same as human-written program. Unlike safe-wandering, the curve shows no sensible fluctuation in learning curve due to independence of learning task and geometrical shape of maze.

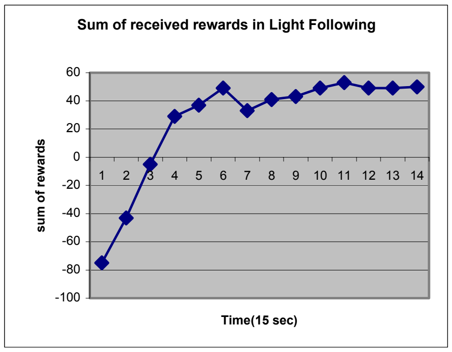

Figure 9-Changes in the sum of received rewards in the light following task

7 Conclusion and future works

In this paper we described the problems we faced in programming micro-robots for learning tasks. The main problems in micro-robots are limitations in power autonomy, processing power and memory (both in program and in data memory). We showed how these limitations forced us to avoid floating-point operations and rely only on integer-based ones, mainly additions or subtractions. We also determined some problems that occur in regular reinforcement learning algorithms when they are limited to integer numbers.
We proposed a simple and fast reinforcement learning algorithm optimized for data and program memory consumption. It was then verified on safe-wandering (with two different configurations) and light following tasks. The fast convergence of algorithm made it possible to save at least 95% of power autonomy. It offers the potential for more time-critic and complex behaviors. The small number of required instructions left around 70% of program memory unused. Since, data memory size depends mainly on Q-table representation, using integers for Q-values saves at least 75% of the required memory space.
The 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 and implement cooperative learning among groups of robots.

8 References

[1] Bonabeu, E.; Dorigo, M.; Theraulaz, G. : Swarm Intelligence : From Natural to Artificial Systems, Oxford University Press,1999.
[2] Caprari,G.; Balmer, P.; Piguet,R.; and Siegwart, R., The autonomous micro robot alice: A platform for scientic and commercial application. In Proceedings of the 9th International Symposium on Micromechatronics and Human Science. 231–235, 1998.
[3] Caprari,G.; Arras,K. O.; and Siegwart, R., 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.
[4] Caprari,G.; Arras,K. O.; and Siegwart, R., Robot navigation in centimeter range labyrinths. In Rückert, U.; Sitte,J.; and Witkowski, U.,eds., Proceedings of the 5th International Heinz Nixdorf Symposium: Autonomous Minirobots for Research and Edutainment, AMiRE 2001.
[5] Dean F. Hougen, Maria Gini, and James Slagle, "Rapid Unsupervised Connectionist Learning for Backing a Robot with Two Trailers", IEEE International Conference on Robotics and Automation, April 1997.
[6] Floreano, D.; Schoeni, N.; Caprari, G.; and Blynel, J.: Evolutionary Bits'n'Spikes. In R. K. Standish, M. A. Beadau and H. A. Abbass, eds. Artificial Life VIII: Proceedings of the Eight International Conference on Artificial Life, MIT Press, 2002.
[7] Siegwart,R.; Wannaz, C.; Garcia,P .; and Blank, R., Guiding mobile robots through the web. In Workshop Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, Victoria, Canada, 1998.
[8] Sutton ,R.S.; and Barto, A.G., Reinforcement Learning:An Introduction, MIT Press, Cambridge, MA, 1998.
[9] Watkins, C. J.C.H., Learning from Delayed Rewards, Ph.D. thesis, King's College, 1989.
[10] Watkins ,C.J.C.H.; and Dayan P., Q-Learning (technical note), In: Sutton R.S.(ed.), "Machine Learning: Special issue on reinforcement learning", vol. 8, 1992.