On an educational approach to behavior learning for robots

Michel Tokic
Institute of Neural Information Processing
University of Ulm
James-Frank-Ring
89069 Ulm, Germany
Email: michel.tokic@uni-ulm.de

Arne Usadel, Joachim Fessler, Wolfgang Ertel
Institute of Applied Research
University of Applied Sciences Ravensburg-Weingarten
Doggenriedstrasse 38
88250 Weingarten, Germany
Email: firstname.lastname@hs-weingarten.de

***Abstract*—This paper introduces a system for teaching biologically-motivated robot learning in university classrooms that might be used in courses such as *Artificial Intelligence* and/or *Robotics*. For this, we present a simple hardware robot that is able to learn a forward walking policy on basis of a reinforcement signal. Students are able to conduct experiments on a PC with a software called the Teachingbox that controls the robot. This software offers the possibility to control the learning method's parameters throughout the learning process, which allows observing the effects of such parameters on a real robot. Furthermore, learning on the hardware robot is very fast since forward-walking policies are usually learned in about 30 seconds. Due to this quick learning process nearly no waiting time is caused, and in return this fact often impresses the audience and leads to the question: "How does it work?".**

I. Introduction & Motivation

As robots or the environment of a robot become more and more complex, the way of programming robots in the classical supervised way also becomes more difficult. As a consequence, engineers often program just a "working" behavior of a robot, but which can be far away from an "optimal" behavior, e.g. movements of a robot that maximize the forward walking velocity. One possible solution to this general problem is offered by learning behaviors from scratch—in the same way as humans or animals do—instead of manually programming the robot. In literature, learning in such way is called *trial-and-error learning* which has been first studied in the domain of psychology and animal learning [1]. Nowadays, the research domain of *reinforcement learning* (RL) [2] aims to mimic *trial-and-error learning* in a machine-learning approach based on a reward signal (or reinforcement signal) which strengthens or weakens action selections in certain situations with the goal of maximizing the cumulative reward. Since robot learning is just one possible application of RL, the knowledge about this research domain broadens an engineer's skill on behavior programming that can also be applied to other applications.

Neller et. al. said: *"Simple examples are teaching treasures. Finding a concise, effective illustration is like finding a precious gem. When such an example is fun and intriguing, it is educational gold."* [3]. At the University of Applied Sciences Ravensburg-Weingarten, we were looking for such kind of illustration that enables to teach RL within a narrow time-slot of about four lessons of an *Artificial Intelligence* introductory course. Within that given time-slot, we introduce the value-iteration and Q-learning algorithms on discrete state and action spaces and explain the exploration/exploitation problem. In order to explain to the students the field of RL, we found a crawling robot with a simple two-DOF arm as sketched in Figure 1 appropriate that has been proposed by Kimura et al. [4] and built in hardware by Tokic [5]. Furthermore, the reason why we also favor a robot instead of a theoretical problem is due to the fact that robots seem to be encouraging motivators for students as also recently reported by Kay [6].

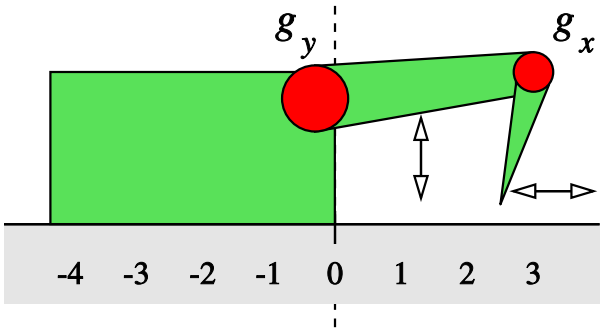


Fig. 1. A model of the crawling robot with its two joints g_x and g_y.

In case of discrete positions and small movement angles of the joints, the state space of the robot arm can be approximated by a grid world. In order to move forward, the robot has to repeatedly perform a cycle of moves as shown in Figure 2 or in the sequence shown in Table I. The task for the learning algorithms is to find a policy (which might be such a cycle) that maximizes the cumulative reward. For this, the reward is the speed of the robot, i.e. the distance that the body of the robot moves forward per time step. Consequently, a move forward gives positive reward whereas any backward move yields negative reward.

In the following we describe the robots architecture and elaborate on simple experiments that students can conduct with a software called the "Teachingbox" [7], which is an open-source framework written in Java. By using this software tool, students are (1) able to send action commands to the robot

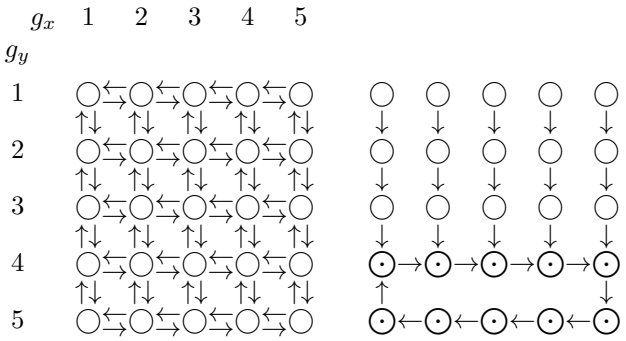


Fig. 2. The 5×5 grid-world model (left) and a cyclic walking policy (right). States within the cycle are labeled as $\odot$.

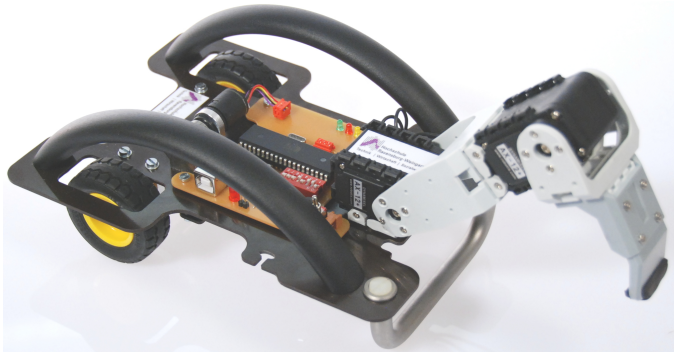
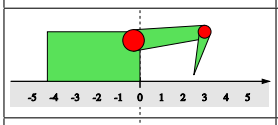
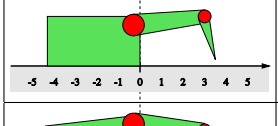
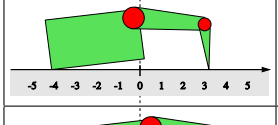
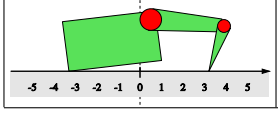


Fig. 3. The crawling robot we use in our laboratory tutorials.

TABLE I
FOUR STEPS OF A SIMPLE CYCLIC FORWARD-WALKING POLICY.

robot	time t	state g_y	state g_x	reward x	action a_t
	0	up	left	0	right
	1	up	right	0	down
	2	down	right	0	left
	3	down	left	1	up

in order to observe the robot's behavior and (2) are also able to learn policies on the basis of observed rewards from the robot's environment.

II. Hardware Robot

A prototype of the robot that we use in our "Laboratory on Artificial Intelligence" is depicted in Figure 3. Basically, this robot is controlled by an ATmega32 microcontroller board that is mounted on top of the robot. This board controls the joints of the robot which are driven by Dynamixel AX-12 actuators. These servos communicate with a half-duplex asynchronous packet-protocol on TTL-level with up to 1,000,000 bps. The maximum holding torque is about 1.17 Nm.

The speed of the robot is measured by an optical incremental encoder that is connected via a non-slip belt transmission to a (rigid) wheel axle. The controller board also comes up with outlets for the servos, an outlet for the encoder and a DIP switch for setting up several parameters. For instance, one of these parameters inverts the encoder signal and results the robot to learn a backward-moving strategy instead of moving forward.

On top of the controller board, there also exists a RF04 ER400TRS serial transceiver module, which is used for communication with the Teachingbox software on the PC side. This module is directly attached to the ATMega's serial port and operates by a speed of 19,200 baud. On the PC side we use a RF04 USB telemetry module for communicating with the controller board over a standard RS232 COM port.

III. Reinforcement Learning

We consider the reinforcement learning framework [2] where an agent interacts with a Markovian decision process (MDP). At each discrete time step, $t \in \{0, 1, 2, ...\}$, the agent is in a certain state, $s_t \in \mathcal{S}$—for example, the angular position of the robot's joints. After the selection of an action, $a_t \in \mathcal{A}(s_t)$, the agent receives a reward signal, $r_{t+1} \in \mathbb{R}$, from the environment and passes into the successor state s_{t+1}. The decision which action is selected in a certain state is characterized by a policy, $\pi(s) = a$, that could also be stochastic: $\pi(a|s) = Pr\{a_t = a|s_t = s\}$. A policy that maximizes the cumulative reward over time is denoted as π^*.

In practice, there exist several approaches by which a policy for the robot can be learned. For this, we recently proposed using the value-iteration algorithm [8] with an online model-learning of the environment in parallel which was derived from the *Dynamic Programming* approach. In order to save additional memory required by the model-learning task, we now propose using a different learning algorithm within this paper that belongs to the family of *Temporal-Difference Learning* methods.

In order to learn an optimal policy π^* for the robot, we use Watkins's Q-learning algorithm [9] as depicted in Algorithm 1. This algorithm basically works by assigning a numerical value to each state-action pair (s, a), where each state-action value, $Q(s, a) \in \mathcal{Q}$, is an estimate of the expected cumulative reward, R_t, for following the current policy by starting in state s and taking action a:

$$\begin{aligned} Q_\pi(s,a) &= E_\pi\{R_t|s_t = s, a_t = a\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right\} , \end{aligned}$$

where $0 < \gamma < 1$ denotes a discounting factor that specifies the influence of rewards received more far in the future.

Furthermore, the parameter $0 < \alpha < 1$ specifies a learning rate that determines how much the value-function estimate is being adapted w.r.t. to the current *temporal-difference error*:

$$\delta = r + \gamma \max_{b \in \mathcal{A}(s')} Q(s', b) - Q(s, a) \ . \tag{1}$$

The affect of both algorithm parameters on the learning process is explored by the students during the conduction of experiments as described in Section V.

Algorithm 1 Q-LEARNING ON ROBOT WITH ε-GREEDY

Initialize Q arbitrarily, e.g. $Q(s, a) = 0$ for all s, a
Initialize start state arbitrarily, e.g. $s \leftarrow (g_x = 1, g_y = 1)$
loop
    $\xi \leftarrow \text{rand}(0..1)$
    if $\xi < \varepsilon$ **then**
        $a \leftarrow$ random action from $\mathcal{A}(s)$
    else
        $a \leftarrow \text{argmax}_{b \in \mathcal{A}(s)} Q(s, b)$
    end if
    select action a
    observe reward r and successor state s'
    $a^* \leftarrow \text{argmax}_{b \in \mathcal{A}(s')} Q(s', b)$
    $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \delta$
    $s \leftarrow s'$
end loop

The robot's action selection policy, which is based on the Q-function learned throughout the interaction with the environment, works as follows. Since the robot is faced with an unknown environment after switching it on, a tradeoff between exploration (long-term optimization) and exploitation (short-term optimization) has to be done [2], [10]. A very simple and commonly used technique for this is ε-Greedy exploration [9], where at each time step the agent selects an action at random with probability $0 \leq \varepsilon \leq 1$ (exploration). With probability $1 - \varepsilon$ (exploitation) the agent selects an action that is greedy with respect to the current value-function estimates:

$$\pi(s) = \begin{cases} \text{random action from } \mathcal{A}(s) & \text{if } \xi < \varepsilon \\ \text{argmax}_{a \in \mathcal{A}(s)} Q(s, a) & \text{otherwise,} \end{cases} \tag{2}$$

where $0 \leq \xi \leq 1$ is a uniform random number drawn at each time step. If there is more than one action having the highest estimated value in state s, a random action of this set of best actions is chosen.

In order to speed-up learning, a commonly used approach is to reduce the exploration rate ε over time. In this case ε is set to a high value at the beginning of the learning process which is decreased by a constant fraction at each time step. This results that the agent is more explorative at the beginning of the learning process, when the environment knowledge is unknown, as later the agent becomes pure exploitative. The final outcome of the learning algorithm where the robot interacted some time with the real world is shown in Figure 4.

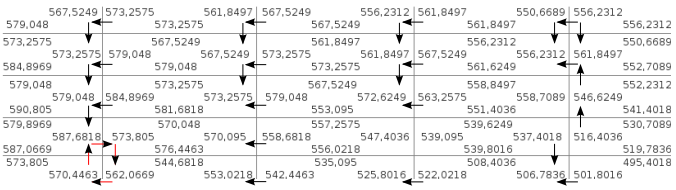


Fig. 4. The Q-values and (greedy) policy learned by Q-learning from a real-world interaction of the walking robot (learned with $\gamma = 0.99$). The corresponding rewards are shown in Figure 5.

IV. THE TEACHINGBOX

When students should learn to understand the behavior of an algorithm, it is didactic supportive to perform experiments with a simple demonstrator. With such it should be possible to easily play with, e.g. in terms of algorithms parameter variations which enable to observe the affect of such parameters on the learning progress. Furthermore, such a demonstrator should also be usable without much effort in order that students focus only on relevant things.

Our recently presented software framework, the Teachingbox (TB) [7], aims at providing a rich library of implemented algorithms for robot learning in a universal robot learning framework. Hereby, the main purpose of this open-source Java framework is to support the development of autonomous agents with learning capabilities. The TB comes up with algorithms for RL, Learning-by-Demonstration, the possibility of manually programming policies and a build-in grid-world editor for modeling simple two-dimensional grid worlds. In particular, the RL-part of the TB currently consists of implementations of the most popular learning algorithms such as value-iteration, Q-learning and SARSA-learning with the support for Softmax action selection and ε-greedy policies [2]. Furthermore, the TB also supports eligibility traces as well as gradient-descent learning of value functions, e.g. by CMACs or radial basis function networks. In order to visualize the learned behavior of an agent, the TB also provides a plotting library for value functions and learned policies.

In our "Laboratory on Artificial Intelligence" students conduct experiments with the TB and the crawling robot by writing simple Java programs. A typical program code that demonstrates the usage of the TB with learning on the hardware robot is depicted in Algorithm 2. At first, a Q-function with tabular approximation is instantiated. Then, the environment (the hardware robot) and the policy to be used are specified. Finally, the Q-learning algorithm (learner) is configured, attached to the agent and a new experiment is started for 1 episode with 300 time steps.

Immediately after the experiment is started, the TB's grid-world editor appears to the user that visualizes the current state of the robot and the rewards observed from the environment in real time, (Figure 5). Furthermore, also a policy window

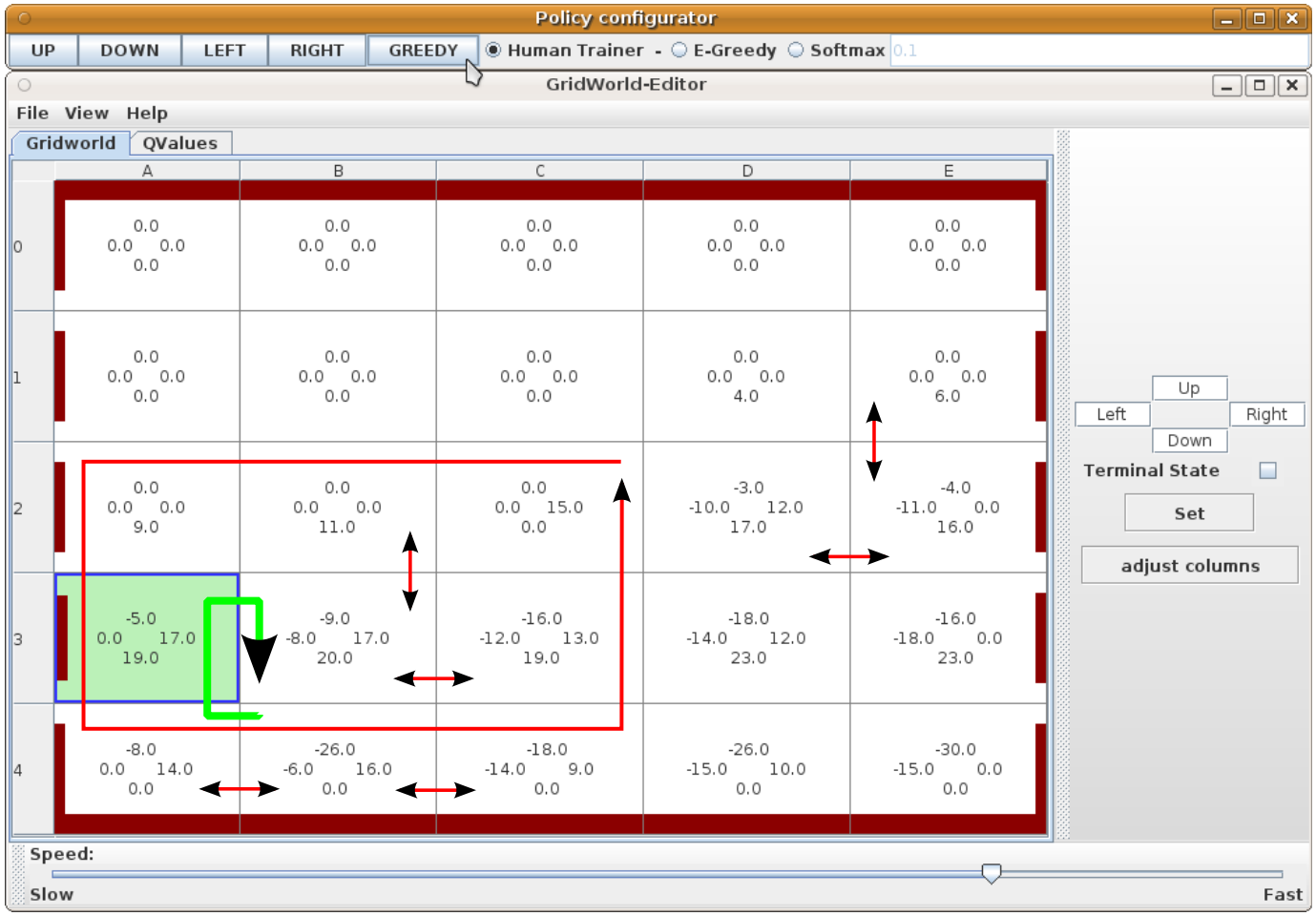


Fig. 5. This figure shows the grid-world editor of the Teachingbox where on top of the window the user is able to configure the policy. Numbers in cells indicate the reward $r(s,a)$ observed from the environment. The cycle A3→B3→B4→A4 indicates the optimal cycle having an average reward of $\bar{r} = \frac{17+20-6-8}{4} = 5.75$ per action. All other marked cycles indicate examples of sub-optimal cycles that have a lower average reward/action compared to the optimal cycle. The cell having the surrounded border (A3) indicates the current state of the robot.

appears (the upper window of Figure 5) in which the user is able to configure the exploration/exploitation policy to be used by the agent. Additionally to the standard policies such as ε-greedy and Softmax, the user can also control the robot "by-hand" when selecting the "Human-Trainer" policy. With this, the robot's actions are controlled by the cursor keys (up, down, left, right) whereby it's also possible to select the "Greedy" action with respect to the currently learned Q-function.

It is easy to adapt the Java code for the use with other environments. For example, if learning should be based in an arbitrary $m * n$ grid-world environment modeled by the user, then only line 2 of Algorithm 2 needs to be adapted to:

GridworldEnvironment env =
new GridworldEnvironment(m,n);

which simply replaces the agent's environment and also demonstrates the flexibility of the Teachingbox which is based on the use of Java Interfaces. This approach standardizes methods of policies, environments and learners with the goal of being interoperable with each other. For example, each environment in the TB implements an `Environment` interface that standardizes important methods such as:

- `double doAction(Action)`
- `State getState()`
- `boolean isTerminalState()`.

where `State` and `Action` are `double` vectors in order to be compatible with each component of the Teachingbox. Another example for the use of interfaces are policies that have to implement a `Policy` interface in order to standardize the methods:

- `Action getAction(State)`
- `Action getBestAction(State)` .

V. Experiments with the robot and the Teachingbox

In the laboratory tutorial on RL, the first task for the students is to model the rewards of a simulation of the crawling robot by using the TB's grid-world editor. After the modeling of an environment, the policy must be learned by a learning algorithm such as Q-learning, Sarsa or value iteration. During

Algorithm 2 SIMPLE Q-LEARNING JAVA EXPERIMENT

```
1: // initialize new Q-Function with Q(s,a)=0 by default
   TabularQFunction Q = new HashQFunction(0);
2: // establish serial robot link (baudrate, port)
   CrawlerEnvironment env =
       new CrawlerEnvironment(19200, "/dev/ttyUSB0");
3: // setup policy configurator
   PolicyConfigurator pi =
       new PolicyConfigurator ( Q,
           CrawlerEnvironment.ACTION_SET);
4: // create agent
   Agent agent = new Agent(pi);
5: // setup experiment with 300 time steps
   Experiment experiment =
       new Experiment(agent, env, 1, 300);
6: // setup Q-function learner
   TabularQLearner learner = new TabularQLearner(Q);
   learner.setAlpha(0.3);
   learner.setGamma(0.9);
7: // attach learner to agent
   agent.addObserver(learner);
8: // start experiment
   experiment.run();
```

this process, students have to conduct several experiments with variations of the learning algorithm parameters α and γ as well as with the policy parameter ε. These experiments lead to the observation that, for example, the discounting-factor $\gamma \in [0, 1)$ has an important influence on the quality of learned policies. For example, if γ is chosen very small, then the agent is more near-sighted and takes not rewards into account received from actions more far in the future, and which often results in learning sub-optimal policies. In comparison, large settings of γ make the agent more far-sighted, but in turn to this, the speed of learning can also be slowly at the beginning of learning since the convergence of the Q-function requires more transition experiences.

While learning the Q-function, the TB can memorize the function on the computer hard-disk, which enables reusing it in other experiments. After the successful learning of the Q-function, students have to evaluate the learned policy from the simulated environment on the real hardware robot. For this, the `GridWorldEnvironment` in the Java code has to be replaced by the `CrawlerEnvironment`. Furthermore, the exploration/exploitation policy for the robot has to be a pure greedy policy ($\varepsilon = 0$), which results that in a given state the action with the highest Q-value is selected. It is important that throughout this experiment learning is disabled in the Java-code, i.e. no `Learner` is attached to the `Experiment`. This results that only the policy of the simulated environment runs on the hardware robot. The idea behind this approach is to enable observing how well the environment has been modeled by the students and how the resulting policy looks like by observing the robots behavior. Therefore, after the selection of an action, the robot transmits the actual state as well as the reward for the most recent selected action to the Teachingbox that visualizes these values in the grid-world editor.

Next, students conduct experiments with the environment of the real hardware robot. In this experiment the `Learner` has to be attached to the `Experiment` again, which enables learning from the robot's environment instead of learning from the simulated environment. Again, each state and reward received from the robot is visualized in the grid-world editor.

Throughout the experiment with the hardware robot, the students task is to vary the learning rate $\alpha \in [0, 1)$ of the Q-learning algorithm that determines how fast the learner adapts the Q-function with respect to the current TD-error δ. The understanding of this parameter is especially important, because the robot interacts in a non-deterministic environment with a noisy reinforcement signal due to the sensor and which also varies due to irregularities of the robot's surface. On the one hand, a large setting of the learning rate causes a fast adaption to environmental changes, for example, when the hardware robot walks from tar to grass. But on the other hand, large settings of α can also be problematic because sensor noise as part of the reinforcement signal may cause the robot to leave a learned "optimal" cycle. In contrast, when the learning rate is relative small, learning of policies takes more time due to the slow adaption of the (more accurate) Q-function.

The last experiment evolves the understanding for the need of balancing exploration and exploitation, which is also conducted on the real hardware robot and where students vary the policy parameter $\varepsilon \in (0, 1)$ of the ε-greedy method (policy configurator on top of Figure 5). As a result, students will find out that without any exploration, i.e. $\varepsilon = 0$, the agent is very likely to stick in sub-optimal cycles of the state space (due to local minima of the Q-function) that in sum yield to a relative small cumulative reward than compared with the optimal cycle. Such a sub-optimal behavior is observable as the forward walking velocity of the robot that might be significantly slower as in comparison to the optimal cycle. The reason for this behavior on the hardware robot is often due to the fact that the robot hasn't walked through every state transition of the state space and thus actually doesn't know about other (better) cycles. If such behavior occurs throughout the learning process, one can simply solve this misbehavior in the TB by increasing the exploration parameter ε in order that the robot also tries other actions which are not greedy with respect to the Q-function. Furthermore, one may also use the "Human-Trainer" policy and guide the robot into parts of the state space which haven't been explored yet.

Finally, after the experiments with Q-learning, students can perform the same kind of experiment with other learning algorithms, for example, with value iteration or Sarsa and then compare the speed of learning.

VI. Educational Experiences & Conclusions

In order to get an overview whether or not the robot is a good demonstrator for reinforcement learning, we asked our students to participate in a pilot survey. Until the deadline of this paper, the online questionnaire was filled out by 5 of 10 students which represent 50% of the participants of our latest AI course. Because of this low return of responses, the quality of the following results indicates just a rough direction and must be seen as preliminary.

The results of our questions reveal that the students are of the opinion that the robot is a good object of study in general and it seems that the understanding of how reinforcement learning works and how learning method parameters affect the robot's speed (policy quality) got conveyed. Despite of being a good demonstrator, the students also remarked that the robot's hardware is still not comprehensive enough. From our point of view, this answer is not surprising since we cover reinforcement learning in about four lessons (each 1.5 hours), where we're limited in teaching just a small scope of the overall research field, i.e. we just explain discrete state and action spaces and do not consider the continuum. Furthermore, the problem of delayed rewards that exists in real world applications (e.g. the outcome of board games) is not given by the robot example. Anyway, interested students may write their own environments within the Teachingbox, which is possible due to the public availability of the source code on SourceForge. Alternatively, students may also play with standard use-cases such as the mountain-car or inverse-pendulum problem, which are already implemented in the Teachingbox.

The results also indicate that the students sustainably enhanced their skills on differentiating between the two main algorithms we convey: value-iteration and Q-learning. In turn, we couldn't obtain the same good result on the importance of the exploration/exploitation problem. Nevertheless, the students' feedback on both the algorithms and the exploration/exploitation problem was still positive and so we achieved our main goal: To present reinforcement learning in a lively, interesting manner and to support the students learning success. Finally, the students were enthusiastic to see that a behavior which has been learned in a simulation could be transferred to a real robot.

From our teachers' point of view, the robot demonstrator is a versatile instrument which greatly enhanced our lessons on reinforcement learning in order to present behavior learning for robots to the students. With the Teachingbox, we have a reliable software tool that also supports more complex hardware demonstrators that eventually will be constructed in the future. Furthermore, we also provide the robot's construction plans, printed circuit board diagrams and videos on our website[1] and thus enable other interested institutions to rebuild the robot by themselves.

[1] http://ailab.hs-weingarten.de/

Acknowledgment

The authors like to thank Markus Schneider, Richard Cubek, Tobias Fromm, Tobias Bystricky and Andy Stumpe from University of Applied Sciences Ravensburg-Weingarten which were involved by the co-development of the Teachingbox. The authors also thank Günther Palm, Mohamed Oubatti and Friedhelm Schwenker from Ulm University for the scientific discussions on the crawling robot and also acknowledge Philipp Ertle from University of Applied Sciences Ravensburg-Weingarten for proof reading and discussing a first draft of the paper.

References

[1] E. L. Thorndike, *Animal Intelligence*. The Macmillan company, New York, 1911.

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

[3] T. W. Neller, C. G. M. Presser, I. Russell, and Z. Markov, "Pedagogical possibilities for the dice game pig," *Journal of Computing Sciences in Colleges*, vol. 21, no. 6, pp. 149–161, 2006.

[4] H. Kimura, K. Miyazaki, and S. Kobayashi, "Reinforcement learning in POMDPs with function approximation," in *Proceedings of the 14th International Conference on Machine Learning (ICML'97)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 152–160.

[5] M. Tokic, "Entwicklung eines lernenden laufroboters," Diploma thesis, University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany, 2006.

[6] J. Kay, "Robots as recruitment tools in computer science: The new frontier or simply bait and switch?" AAAI Press, Tech. Rep., 2010.

[7] W. Ertel, M. Schneider, R. Cubek, and M. Tokic, "The Teaching-Box: A universal robot learning framework," in *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009)*, 2009, pp. 1–6, http://www.teachingbox.org.

[8] M. Tokic, W. Ertel, and J. Fessler, "The crawler, a class room demonstrator for reinforcement learning," in *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference (FLAIRS'09)*, H. C. Lane and H. W. Guesgen, Eds. Sanibel Island, Florida, USA: AAAI Press, 2009, pp. 160–165.

[9] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, England, 1989.

[10] S. B. Thrun, "Efficient exploration in reinforcement learning," Carnegie Mellon University, Tech. Rep., 1992.