Proceedings of the Twenty-Second International FLAIRS Conference (2009)

The Crawler, A Class Room Demonstrator for Reinforcement Learning

Michel Tokic and **Wolfgang Ertel** and **Joachim Fessler**
ZAFH Servicerobotik
University of Applied Sciences Ravensburg-Weingarten
Doggenriedstrasse 38
88241 Weingarten, Germany

Abstract

We present a little crawling robot with a two DOF arm that learns to move forward within about 15 seconds in real time. Due to its small size and weight the robot is ideally suited for classroom demonstrations as well as for talks to the public. Students who want to practice their knowledge about reinforcement learning and value iteration can use a wireless connection to a PC and monitor the internal state of the robot such as the value function or the reward table. Due to its adaptivity, depending on the surface properties of the underground the robot may surprise its audience with unexpected but efficient walking policies. The GUI is open source and the robot hardware is available as a kit from the authors.

Introduction

In a classical introductory AI class, reinforcement learning has a narrow time slot of a few hours only. In our AI class we introduce value iteration (Bertsekas 1987) and Q-Learning (Watkins 1989) on discrete state- and action spaces and explain the exploration problem. Finally we give a short outlook on continuous problems and other algorithms.

In order to give a vivid presentation, we were looking for an example of a simple robot that enables us to directly demonstrate the learning algorithms. The state space of such a robot has to be two dimensional and the actions of the robot should be moves to neighbor states in the two dimensional grid world. A robot that very nicely fulfills these requirements was presented as a simulation in (Kimura, Miyazaki, and Kobayashi 1997). Figure 1 shows an outline of the little crawling robot with a two degree of freedom arm. In case of discrete positions and small movement angles of the two joints the state space can be approximated by a cartesian grid world as shown in Figure 2.

To move forward, the robot has to repeatedly perform a cycle of moves. An example for such a cycle is shown in Figure 2 on the right or in the sequence shown in Table 1. The task for the learning algorithms is to find a policy (which might be such a cycle) that maximizes the long-term cumulative reward. The reward is the speed of the robot, i.e. the distance the body of the robot moves forward per time step. Consequently, a move forward gives positive reward whereas any backward move yields negative reward.

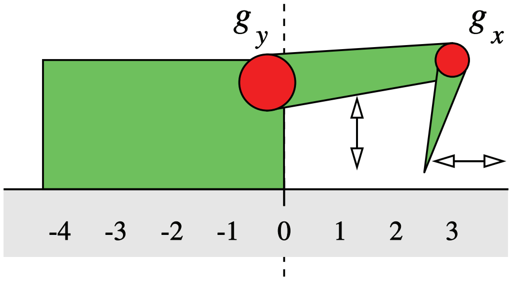


Figure 1: The robot with its two joints g_x and g_y.

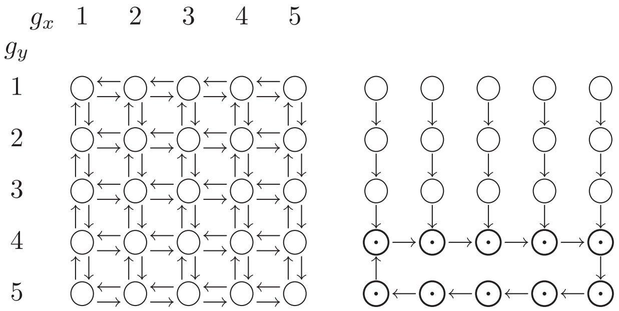


Figure 2: The cartesian 5×5 grid-world model (left) and a cyclic walking policy (right). States within the cycle are labled as $\odot$.

As a last but important requirement, the robot has to be small, lightweight and autonomous, such that it can easily be taken to the class room in a brief case and be demonstrated without an external power supply.

In the following sections we describe the robot hardware, the robot software and a graphical user interface (GUI) on the PC with a wireless connection to the real robot (Tokic 2006). For didactic purposes the GUI is very helpful because it allows us to view the value table and reward table and to watch the learning process of the robot.

The Robot

Hardware Setup

Our crawling-robot prototype as depicted in Figure 3 is controlled by an ATmega32 microcontroller board which

---

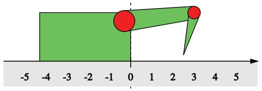
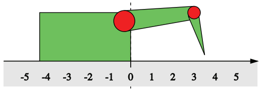
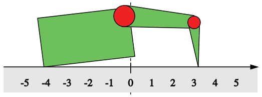
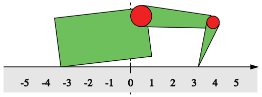
robot	time	state		reward	action
	t	g_y	g_x	x	a_t
	0	up	left	0	right
	1	up	right	0	down
	2	down	right	0	left
	3	down	left	1	up

Table 1: Four steps of a simple cyclic forward walking policy.

is mounted on top of the robot. The joints of the robot are driven by servos and the robot's movement is measured by an optical incremental encoder attached to one of the wheels. The board has outlets for the servos, a serial wireless transceiver, an outlet for the encoder and a DIP switch to setup several parameters. For instance one of these parameters inverts the encoder signal such that the robot learns a backward-moving strategy instead of moving forward.

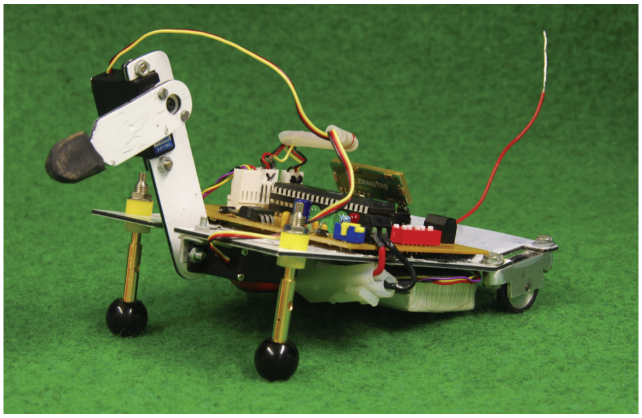


Figure 3: The crawling-robot prototype.

In addition to this prototype we developed a new version of the crawling robot as shown in Figure 4. Due to its robustness this robot is more suitable for lectures and laboratory tutorials. It has two wheels at the rear with a rigid axle. The axle is connected via a non-slip belt transmission to the encoder. This assembly reduces the sensor noise to a minimum. The board has the same features as the board above but with a bus interface for Dynamixel AX-12 servos in addition. 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.

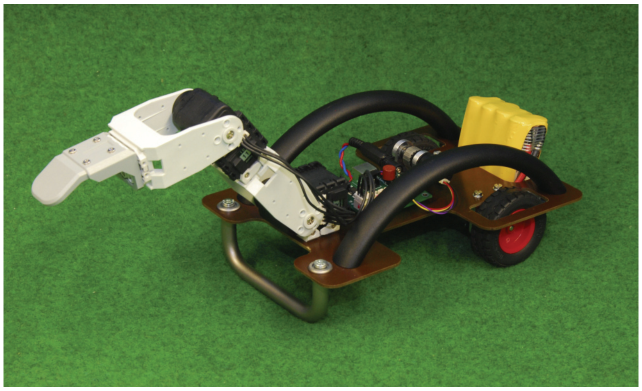


Figure 4: The new crawling robot for use in our laboratory tutorials.

The Learning Algorithm

We consider the reinforcement learning framework, see for example (Sutton and Barto 1998), where an agent, in our case the crawling robot, interacts with a Markovian decision process (MDP). At each time $t \in \{0, 1, 2, ...\}$ the agent is in a certain state $s_t \in \mathcal{S}$. After executing action $a_t \in \mathcal{A}$ the agent receives a reward signal $\mathcal{R}^a_{ss'} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}$ by passing into a successor state s_{t+1} with probability $\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s'|s_t = s, a_t = a\}$, $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}$. $\delta(s, a)$ denotes a transition function and represents the successor state s' after action a has been selected in s. The decision which action a is chosen in a certain state s is characterized by a policy, $\pi(s) = a$, which could also be stochastic $\pi(s, a) = Pr\{a_t = a|s_t = s\}$. A policy which maximizes the average reward over time is denoted as π^*. On the crawling robot we use the value-iteration algorithm (Bertsekas 1987; Sutton and Barto 1998; Powell 2007) for learning a cyclic deterministic policy $\pi(s)$ by which the robot moves forward. The algorithm value iteration which we use on the robot basically works by assigning a numerical value $V(s) \in \mathcal{V}$ to each state $s \in \mathcal{S}$ where each state value represents the expected cumulated reward over time when following $\pi(s)$. Based on these state values $\mathcal{V}$ in conjunction with $\mathcal{R}^a_{ss'}$ a deterministic policy $\pi(s)$ can be derived.

We adapted the value-iteration algorithm to perform on the crawling robot as depicted in Algorithm 1. Here a discounting factor $0 < \gamma \leq 1$ is used to specify the portion of influence of neighbor-state values $V(\delta(s, a))$ on $V(s)$. Since the robot is faced with a zero-knowledge environment after switching it on, a tradeoff between exploration (long-term optimization) and exploitation (short-term optimization) has to be done (Thrun 1992; Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998). A very simple exploration technique is ε-Greedy exploration. Here the agent selects an action at random with a probability ε, uniformly and independently of $V(s)$, instead of following $\pi(s)$. Heuristically ε could be set to a high value at the beginning of the learning process and then be decreased over time. This ensures much exploration at the beginning and pure exploitation starting at some time t.

Algorithm 1 VALUEITERATION ON ROBOT

```
Initialize $V$ arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}$
Initialize $\mathcal{R}^a_{ss'}$ arbitrarily, e.g., $r(s,a) = 0$, for all $r \in \mathcal{R}^a_{ss'}$
$state \leftarrow (g_x = 1, g_y = 1)$
loop
    $\xi \leftarrow \mathrm{rand}(0..1)$
    if $\xi < \varepsilon$ then
        $a \leftarrow \mathrm{rand}(\mathcal{A}(state))$
    else
        $a \leftarrow \mathrm{argmax}_a \mathcal{R}^a_{ss'} + \gamma V(s')$
    end if
    $successorState \leftarrow \delta(state, a)$
    observe $r(state, a)$ and update $\mathcal{R}^a_{ss'}$
    for all $s \in \mathcal{S}$ do
        $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} r(s,a) + \gamma V(\delta(s,a))$
        $\pi(s) \leftarrow \mathrm{argmax}_{a \in \mathcal{A}(s)} r(s,a) + \gamma V(\delta(s,a))$
    end for
    $state \leftarrow successorState$
end loop
```

As the value-iteration algorithm requires the rewards $r(s,a)$ for each action a in state s, we simply save them into a memory table $\mathcal{R}^a_{ss'}$. After executing action a in s, we update the corresponding record $r(s,a)$ in $\mathcal{R}^a_{ss'}$.

The Robot Software

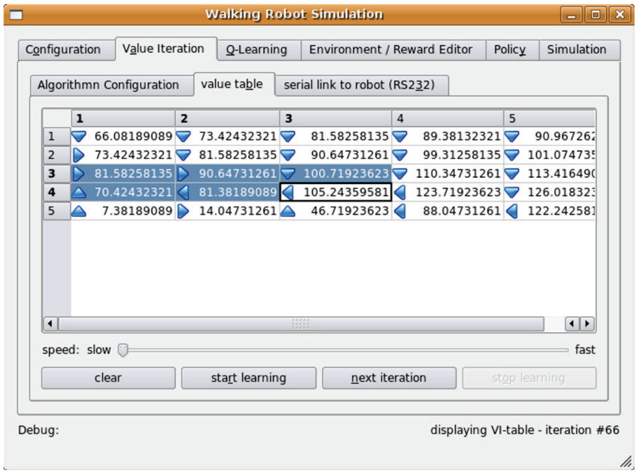
The robot's software architecture runs autonomously in a single thread on the ATmega32 microcontroller. This microcontroller comes with four "pulse-width modulated" (PWM) pins. We use two PWM-pins associated with Timer1A/B to directly position the joint-servos. The optical incremental encoder which is connected to an external-interrupt pin provides the reward signal $r(s,a)$. An interrupt-service routine (ISR) counts the distance the robot moved within a small delay time t. It is activated subsequently after setting up the Timer1A/B registers by which we position the servos with respect to state s and action a.

In an endless loop the robot executes actions either with respect to the learned policy $\pi(s)$ or exploration steps. After each action a value-iteration step for all $s \in \mathcal{S}$ is performed. Subsequently the serial interface is polled to check if the GUI sent a command to which the microcontroller should respond. For instance such a command can be a request for transmitting $\mathcal{V}$ and $\mathcal{R}^a_{ss'}$ to the GUI in order to display the robot's "brain".

Walking Robot Simulation

To demonstrate and test reinforcement learning algorithms on the two-dimensional grid-world, we developed a graphical user interface (GUI) which allows control of the real hardware robot and a simulated version of it. In addition to an animation of the robot's x-z-projection the GUI comes up with a reward editor, a display of the current value table and a view of the current policy, Figures 5 and 6. A configuration editor enables the user to select the size of the state space and to specify the learning algorithm parameters, for example the discounting factor γ. In the configuration dialog it is also possible to configure a neural network for approximating the V-Function using back- or resilient-propagation (Rumelhart, Hinton, and Williams 1986; Riedmiller and Braun 1992).

In the view of the value table the user is able to control value iteration step-by-step using the "next iteration"-button. As well it is also possible to run the learning algorithm in a time-delayed or full-CPU-speed loop. Due to this feature the user can observe the robot's learning progress over some time period. For this in each robot state s, which is a cell in the value table, we also display an arrow respresenting the robot's current policy $\pi(s)$ as shown in Figure 5.



	1	2	3	4	5
1	▼ 66.08189089	▼ 73.42432321	▼ 81.58258135	▼ 89.38132321	▼ 90.96726
2	▶ 73.42432321	▼ 81.58258135	▼ 90.64731261	▼ 99.31258135	▼ 101.07473
3	▶ 81.58258135	▶ 90.64731261	▼ 100.71923623	▼ 110.34731261	▼ 113.41649
4	▲ 70.42432321	◀ 81.38189089	◀ 105.24359581	◀ 123.71923623	▼ 126.01832
5	▲ 7.38189089	▶ 14.04731261	▲ 46.71923623	◀ 88.04731261	◀ 122.24258

Debug:

displaying VI-table - iteration #66

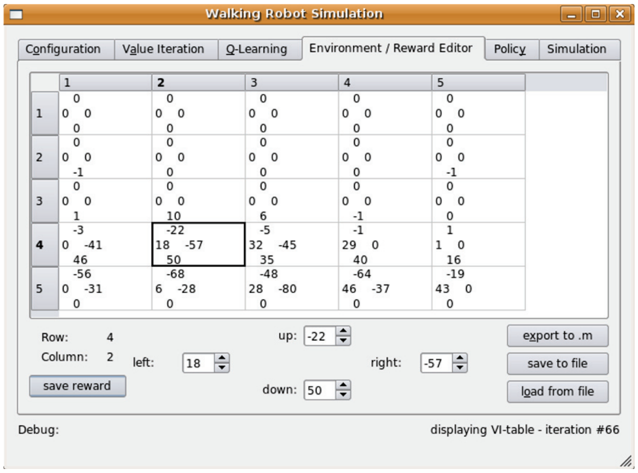
Figure 5: The simulator's value table view with arrows for the current policy in each state. Please note the marked cyclic policy which the hardware robot learned after executing 66 value-iteration steps with respect to the environment as displayed in Figure 6.

Another important feature of the simulator is the reward editor as depicted in Figure 6. It allows the user to model the agent's environment and thus to simulate the real hardware robot on different grounds. Alternatively the hardware robot's reward table $\mathcal{R}^a_{ss'}$ is displayed in this view when the simulator is connected to the robot. In combination with the robot's value table this enables the user to get an insight into the robot's "brain". In the next section we will show how the simulator can be used by students to model different robot environments.

Apart from simulating the crawler, the simulator can also be used to simulate arbitrary two dimensional grid worlds with the four actions left, right, up and down.

Serial Robot Interface

One of the GUI's main features is the serial robot interface by which the user is able to communicate wireless with the hardware robot. The interface enables the user to analyze the



	1	2	3	4	5
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2	0 0 0 -1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 -1
3	0 0 0 1	0 0 0 10	0 0 0 6	0 0 0 -1	0 0 0 0
4	-3 0 -41 46	-22 18 -57 50	-5 32 -45 35	-1 29 0 40	1 1 0 16
5	-56 0 -31 0	-68 6 -28 0	-48 28 -80 0	-64 46 -37 0	-19 43 0 0

Row: 4
Column: 2
left: 18
up: -22
right: -57
down: 50

Debug: displaying VI-table - iteration #66

Figure 6: The simulator's reward editor displaying the hardware robot's reward table $\mathcal{R}^a_{ss'}$.

robot's learning progress and to setup several learning process specific parameters. The link is realized by a RF04/400 USB telemetry module on the PC side, connected to the USB port. On the microcontroller side, we use an Easy-Radio ER400TRS transceiver which is directly connected to the microcontroller's USART pins and which is also a part of the RF04/400 USB telemetry module. The two transceivers operate at the 433MHz frequency band with a serial baudrate of 19200. The GUI view provides to the user the following features:

- send control commands to the robot, e.g. "start walking" or "stop walking"
- retrieval of $\mathcal{R}^a_{ss'}$ and $\mathcal{V}$ from the robot
- transfer $\mathcal{R}^a_{ss'}$ and $\mathcal{V}$ from the GUI to the robot
- configuration of the robot's discounting factor γ
- configuration of the robot's exploration parameter ε
- retrieval of a counter, c, that indicates what distance the robot moved forward since the last counter reset
- reset of the distance counter c

Class Room Experience

We used the crawler in the AI class for the computer science master students. At the very beginning of the section on reinforcement learning we give a demonstration of the crawler learning to walk within about 20 seconds. This simple demo leads to a high motivation: the students want to know "how it works". Then, after introducing value iteration, the students can perform their own experiments on the robot. Some of the possible experiments are described in the following.

In our laboratory tutorials a typical task for a student is to observe the robot's learning behaviour on different surfaces. A part of this exercise is the examination of the robot's learning progress by loading $\mathcal{V}$ and $\mathcal{R}^a_{ss'}$ from the robot into the GUI. This enables the student to identify the policy cycle and to verify it. This is done by computing the average reward which is the ratio of the cumulated reward and the number of actions within the cycle. In the example of Figures 5 and 6 the robot's average reward is $\frac{0+0+6+32+18-3}{6} = 8.833$. Afterwards the students simulate the behaviour with a different discounting factor γ to get a sense about the effect of γ on learning times. For example a low γ leads to short learning times because value iteration converges faster. But unfortunately this leads to the problem that the optimal cycle is eventually not learned and suboptimal cycles are learned instead of the optimal one. Since $V(s)$ respresents the expected reward in the future, this effect is caused by too little information about states farther away from s. However solving this problem with a high $\gamma \lessapprox 1$ leads to longer learning times because value iteration converges slower. For the hardware robot we recommend $\gamma = 0.9$ as the discounting factor since in all our experiments we got short learning times with it and we were able to find the optimal cycle.

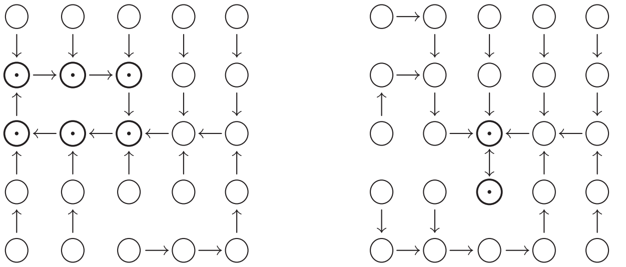


Figure 7: Left: The robot's policy learned on the laboratory carpet as depicted in Figure 3. Right: The robot's policy learned on a table with a slippery surface as depicted in Figure 8. States within the cycles are labeled as $\odot$.

Another parameter which can also be explored by the students is the exploration-rate parameter $0 \leq \varepsilon \leq 1$. The choice of ε also impacts the learning time. For example, $\varepsilon = 1$ leads to 100% exploration and the robot performs random actions all the time. On the other hand if $\varepsilon = 0$, then the robot exploits greedily the current policy without exploring if there are other actions that yield more reward in the future. This effect is observable by the students as the robot will mostly stick in sub-optimal walking policies if $\varepsilon = 0$ is configured. During the experiments the students find out that $\varepsilon = 0.1$ is a good choice for balancing exploration and exploitation on this little crawling robot.

Some results of our experiments are shown in Figure 7 where the robot learned different policies due to different surface properties. We stopped the robot after a while of learning and examined the policy. The videos showing this learning process can be found on our laboratory website (AI-Lab HRW 2008).

Another very valueable exercise for the students is the inverse of the above task. Here the student has to design patterns of immediate rewards that simulate different surfaces on which the crawler has to move. Then the student uses the simulator to train a policy with value iteration. Now he/she can check if the policy meets the requirements.

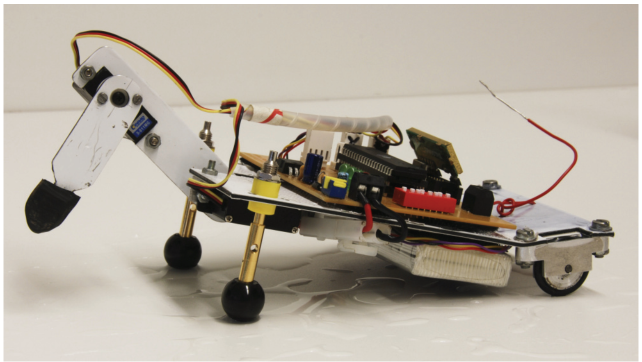
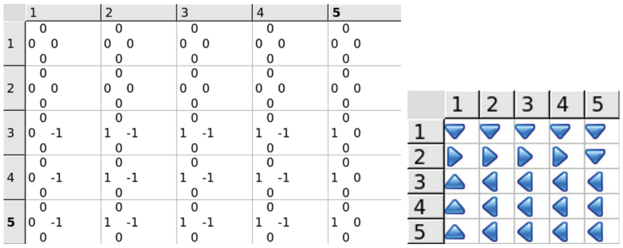


Figure 8: The crawling-robot prototype on a desk with a slippery surface.

As a first task, a flat surface is given. On such a ground, whenever the robot lifts its body a little bit, it can move effectively. An appropriate 5 × 5 reward pattern together with the resulting walking policy is given in Figure 9. With minimal vertical lifting, but maximal horizontal movement, the robot makes “big flat steps”.

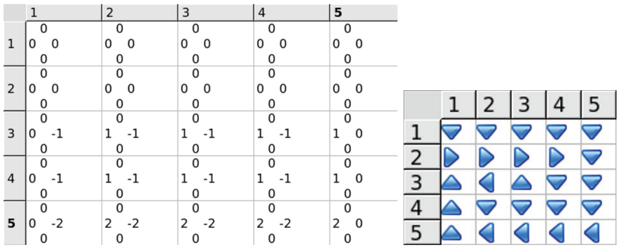


	1	2	3	4	5
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
3	0 0 -1 0	0 1 -1 0	0 1 -1 0	0 1 -1 0	0 1 0 0
4	0 0 -1 0	0 1 -1 0	0 1 -1 0	0 1 -1 0	0 1 0 0
5	0 0 -1 0	0 1 -1 0	0 1 -1 0	0 1 -1 0	0 1 0 0

	1	2	3	4	5
1	▼	▼	▼	▼	▼
2	▶	▶	▶	▶	▼
3	▲	◀	◀	◀	◀
4	▲	◀	◀	◀	◀
5	▲	◀	◀	◀	◀

Figure 9: Simulated reward pattern for a flat surface (left) and the corresponding learned policy (right).

As a next task, the surface is rough, like it is if a human or animal walks in deep snow. Here the legs have to be lifted up high. Otherwise, the resistance or friction is very high. This can be modelled with high rewards if the body is lifted up high and low (but non-zero) rewards with small lifts. Figure 10 shows such a reward pattern together with the resulting walking policy.



	1	2	3	4	5
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
3	0 0 -1 0	0 1 -1 0	0 1 -1 0	0 1 -1 0	0 1 0 0
4	0 0 -1 0	0 1 -1 0	0 1 -1 0	0 1 -1 0	0 1 0 0
5	0 0 -2 0	0 2 -2 0	0 2 -2 0	0 2 -2 0	0 2 0 0

	1	2	3	4	5
1	▼	▼	▼	▼	▼
2	▶	▶	▶	▶	▼
3	▲	◀	▲	▼	▼
4	▲	▼	▼	▼	▼
5	▲	◀	◀	◀	◀

Figure 10: Simulated reward pattern for a rough surface (e.g. deep snow) and the corresponding learned policy.

Finally, the student can compare the simulated policies and reward patterns with the results from the experiments on the real robot. Differences are due to the fact, that for large angles of the robot joints, the geometry of the crawler movements is far from the Euclidean (rectangular) geometry in the simulation. This leads to the insight that even a robot as simple as our crawler can “invent” policies which a programmer never would have designed. For the students this is a very convincing motivation for the study of learning algorithms for robots.

Conclusions and Future Work

Due to the relatively small state- and action space the learning times on the real robot with its ATmega32 microcontroller are very short. Usually after about 10 to 30 seconds the value-iteration algorithm has learned a nice walking policy. Due to this impressive learning speed and its animal like shape and movements, in many lectures and talks the little crawling robot was able to attract the attention of students and spectators. Especially for an abstract and mathematically challenging subject such as reinforcement learning this is the best way of motivating students we can imagine.

Even for us developers it was surprising to observe how sensible this very simple robot adapts its behaviour to the surface properties of the underground. The videos at (AI-Lab HRW 2008) show five policies learned on different surfaces. This can easily be demonstrated in the classroom e.g. by running the robot on a table or on a carpet and observing the difference.

Since this is an ongoing project we currently implement Q-Learning on the hardware robot as well as in the simulator as an alternative to value iteration. Once it is implemented the students are able to perform observations of the learning speed with respect to different learning algorithms. As a further extension we plan to implement different exploration/exploitation techniques, for example counter-based exploration as described in (Thrun 1992), since this will also impact the learning times.

Last but not least we want to encourage other universities to use this little robot for communicating the fascination in robot learning to their students. On our website (AI-Lab HRW 2008) we offer a hardware kit, an open-source simulator and the hardware robot’s open-source software.

Acknowledgements

We like to thank Hans-Peter Radtke, Markus Schneider, Arne Usadel and Tobias Bystricky for their contributions to the source code. We also thank Andy Stumpe and Christian Bieber for their support in taking measurements, video recordings and photos of the walking robot and for bringing the software to our website.

This work has been conducted within the *ZAFH Servicerobotik* (ZAFH 2008). The authors gratefully acknowledge the research grants of the state Baden-Württemberg and the European Union.

References

AI-Lab HRW. 2008. University of Applied Sciences Ravensburg-Weingarten, Germany. Website. `http:`

`//ailab.hs-weingarten.de` [Online; accessed 20-February-2009].

Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. P. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kimura, H.; Miyazaki, K.; and Kobayashi, S. 1997. Reinforcement Learning in POMDPs with Function Approximation. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, 152–160. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley-Interscience.

Riedmiller, M., and Braun, H. 1992. RPROP- A Fast Adaptive Learning Algorithm. Technical report, Proc. of ISCIS VII), Karlsruhe University, Germany.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: foundations* 318–362.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Thrun, S. B. 1992. Efficient Exploration In Reinforcement Learning. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA.

Tokic, M. 2006. Entwicklung eines lernenden Laufroboters. Diploma thesis, University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany.

Watkins, C. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, University of Cambridge, England.

ZAFH. 2008. Zentrum fuer Angewandte Forschung an Fachhochschulen. Website. `http://www.zafh-servicerobotik.de` [Online; accessed 20-February-2009].