The Teaching-Box: A Universal Robot Learning Framework

Wolfgang Ertel, Markus Schneider, Richard Cubek, Michel Tokic†
University of Applied Sciences Ravensburg-Weingarten
Doggenriedstrasse 42
88250 Weingarten, Germany
<Firstname>.<Lastname>@hs-weingarten.de

Abstract— **There exist many powerful machine learning software libraries [15], which help the engineer to build robots that learn autonomously. However, engineering of an autonomous robot still is a challenging and time consuming task even with these learning libraries. With the open source Teaching-Box presented here, the "training" of a robot becomes easier due to the following features. The Java library of the Teaching-Box provides algorithms for reinforcement learning as well as for learning by demonstration (utilizing supervised learning algorithms) and data structures for exchanging policies between the different ways of learning. As an initial policy one can even take a manually coded behaviour and then improve it for example with reinforcement learning. A human trainer feedback (e.g. via the speech interface) can be used to increase the learning speed. The Eclipse based GUI facilitates the design of the robot learning projects and visualizes the learning process. For connecting the various modules of a project, open interface standards such as RL-Glue are used and an easy integration of the Teaching-Box into standard robot middleware is possible.**

I. Introduction

One of the reasons why powerful autonomous service robots cannot be bought off the shelf is their extremely high price. This is not surprising, because for every new application the complete robot software and often even hardware is being developed from scratch. The Collaborative Center for Applied Research on Service Robotics (ZAFH Service Robotics), funded by the German state of Baden-Württemberg, aims at making the development of robots cap-**able**, afford-**able** and depend-**able**. Within this research consortium one subproject develops the **Teaching-Box** which uses advanced machine learning techniques to relieve the robot developer from extensive (and expensive) programming of sophisticated robot behaviours. Furthermore, current machine learning research provides numerous examples (e.g. [18] of complex robot behaviours that today can be learned, but can not be classically programmed.

The Teaching-Box provides a rich and comfortable toolbox for easy experimenting with different algorithms and different ways of learning. For example, for teaching a robot arm to solve some task, creating a first policy by demonstration learning and then improving it by reinforcement learning may lead to a good behaviour. Similarly, it is possible to improve classically programmed policies with reinforcement learning. In cases with rare feedback from the environment, additional human trainer feedback may increase the learning speed. In such a spirit, the Teaching-Box will support the robot developer in training his robot with heuristic manual support and guidance in a pragmatic and easily configurable way. The intended users of the Teaching-Box are researchers as well as industrial robot engineers.

The Teaching-Box does not contain algorithms that remedy the curse of dimensionality inherent to reinforcement learning. However, since the Teaching-Box facilitates experiments that combine different learning methods and heuristics, it may help the engineer to solve problems which could not be solved with reinforcement learning only.

In particular, the Teaching-Box comes up with the following features:

1) Learning by demonstration: The robot developer demonstrates the task to the robot and the robot tries to imitate the demonstrated behaviour. For example, a robot arm may be trained by a human teacher to solve some task while the teacher manually guides the arm. During this process, at each discrete time step the state of the robot (e.g. the vector of the angles of all the joints) and the current action (e.g. the vector of the angular accelerations of all or some of the joints) are stored. Then an algorithm for supervised learning (e.g. a neural network or decision tree learning) is trained on these data in order to generalize from the finite data sample to the whole state space [8], [9]. To provide more appropriate feedback for the learning agent, the human teacher can give feedback to the robot either via the GUI or by means of a speech recognition component [19].

2) Reinforcement learning: The robot explores its environment by performing actions and receiving reward which may be delayed or missing for many time steps. Based on the reward, the robot continually improves its policy [17].

3) Programming: Manual programming of policies in Java is also possible. This is particularly interesting in combination with reinforcement learning.

4) Hybrid learning: All three programming techniques mentioned above can be combined to derive optimized robot policies. For example, a policy for a robot arm may first be learned by demonstration through a human teacher and

† Michel Tokic is also affiliated with the: Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

then refined by reinforcement learning. Or an initial simple walking policy for a four legged robot may be programmed manually and then improved by reinforcement learning. The Teaching-Box provides data structures for transferring policies from one method to another.

5) Java Library: Learning algorithms for supervised learning and reinforcement learning are provided in a Java library. We intend to add other open libraries such as the FANN neural network library as well as the WEKA library with algorithms for supervised learning to the Teaching-Box.

6) Eclipse based GUI: Simple robot learning projects can easily be developed in the GUI without programming. The GUI also provides tools for displaying graphs such as learning curves, cumulated rewards, etc. The use of the GUI is optional, since the Teaching-Box can be used as API only.

7) Open Interface Standards: Interfacing a Teaching-Box project with reinforcement learning algorithms in other languages can easily be done with the integrated RL-Glue components [5]. For connecting complex robots and other software such as image processing or sensor fusion programs to the Teaching-Box we provide interfaces to standard middleware such as SmartSoft [14] or PlayerStage [12].

8) Open Source: The Teaching-Box is open source and we want to invite the machine learning and the robot learning communities to contribute software to this platform. The Teaching-Box will be publicly available for the ICAR 2009 conference from `www.servicerobotik.hs-weingarten.de/teachingbox`.

II. State Of The Art

We examined already existing reinforcement learning frameworks in order to find out (1) if there are similar projects to ours and (2) how they differ to our framework. Because of its platform independence and meanwhile fast execution times, we prefer the JAVA programming language and considered only open-source frameworks that are under a license available for free. Another advantage of JAVA are the excellent third-party libraries such as Colt [6], which is a framework for high performance scientific- and technical-computing from CERN and others. We also examined the frameworks if they provide a RL-Glue interface [5], which is a standard interface to connect agents, environments and experiment programs together. The examination results under the mentioned constraints are two frameworks, that are briefly described in Section II-A and II-B. Other state-of-the-art machine learning libraries that we use within the Teaching-Box are described in Section II-C and II-D.

A. JRLF

The *Java Reinforcement Learning Framework* (JRLF) is a free software under the GNU General Public License (GPL) available from [10]. The framework comes up with a variety of learning agents, e.g. Q(λ)-Learning, SARSA(λ)-Learning, UTree, TTree, AMPS etc. In addition the software provides a graphical system to visualize algorithms and for evaluating the learning performance. Unfortunately, there seems to be no further development since the last (and also the first) version is from 2006. In the currently available version JRLF doesn't provide a RL-Glue interface.

B. PIQLE

Another project that is ongoing in the development by the authors and the open-source community is the ***P**latform for **I**mplementing **Q**-**L**earning **E**xperiments* (PIQLE), available as free software from [4] under the GNU Lesser General Public License (LGPL). This software is primarily designed by the authors for implementing and testing the algorithms and problems that are described in the book *Reinforcement Learning: An Introduction* [17]. The software comes with a RL-Glue interface that enables external programs to use the library without much effort.

C. FANN

A state-of-the-art library for learning with neural networks is the **F**ast **A**rtificial **N**eural **N**etwork Library (FANN) [2]. It is free software under the LGPL licence and used by the Teaching-Box, e.g. to approximate Q-Functions or Policies.

D. Weka Machine Learning Library

Weka is a well established Java-Library with machine learning and data mining algorithms which accompanies the data mining book [20]. At least the decision tree learning algorithms will be included in the Teaching-Box.

III. The Architecture

A simplified sketch of the Teaching-Box architecture is shown in Figure **??**. Here we see how the modules interact with each other and how the software can be used by third-party applications through the interfaces. The core of the Teaching-Box is a learning agent that interacts with an environment, which can be for example a robot or a simulation. The learning agent typically receives reward signals that indicate how valuable an action has been in terms of reaching an overall goal. The goal initially is unknown to the agent. An example of such a goal could be to successfully grasp a specific object.

A pure reinforcement learning agent that initially has zero-knowledge about its environment, typically requires a very long time for learning a good policy. In order to overcome long learning times, the Teaching-Box provides a "human trainer" interface that enables the user to roughly demonstrate an arm robots trajectory (learning by demonstration) which can subsequently be refined by the use of reinforcement learning methods [8], [9]. More precisely, the interface enables the user to teach transition pairs $(state, action)$ into the Teaching-Box, which are then learned by supervised learning methods such as neural networks or decision trees. Thereafter, the performance of the resulting policy can be evaluated and improved by reinforcement learning methods.

In addition, to further improve the learning process, the user can provide feedback to the learning agent via the human trainer module. The Teaching-Box offers an interface that enables the user to send reward signals by speech or by a joystick. The described sequence of learning steps is

an example of a learning project typical for the use of the Teaching-Box. It is illustrated in Figure 1.

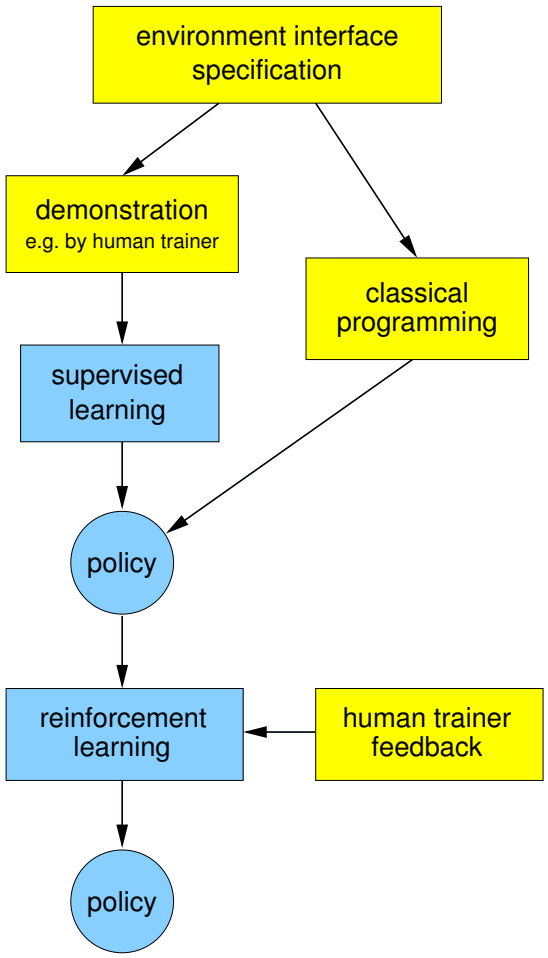

Fig. 1. Control flow in a typical Teaching-Box project.

The overall learning process is controllable by the "Experiment" module which enables the user, for example, to specify the number of steps within one learning episode. The learning results can then be observed through a visualization module which, for example, displays the immediate (mean) reward.

IV. Graphical User Interface (Editor)

One of the major goals of the Teaching-Box is to facilitate experiments with learning agents in research, development and education. Therefore, the user should not be forced to write program code to build, change or extend an experimental setup. This applies not only to simulated environments, but also to external ones such as a real robot arm. Based on the object oriented design of the underlying API, the GUI provides components to represent experiments, environments and agents. Policies, learning methods, Q-functions etc. as integral parts of agents are also represented by components. An experimental setup (in the following called project) can easily be built by drag and drop. Visualizations of the results can be called from the GUI.

A. Eclipse Graphical Editing Framework (GEF)

The Teaching-Box's GUI (the Editor) uses the Eclipse Graphical Editing Framework [1], written in Java. GEF is an eclipse based plug-in to build graphical user interfaces for models and processes in science and business, where visual representations typically lead to graphs and nested structures. It is a highly abstracted implementation of the Model-View-Controller architectural pattern. Well-known applications like KNIME (Konstanz Information Miner) [3] or jBPM (jBoss Business Process Management) are based on this framework.

B. Modeling an Experimental Setup

Every experiment consists of an agent and an environment as the fundamental parts of the agent-environment interface [17] and an experiment to trigger and control the interaction between them. These three basic components are represented as boxes in the GUI (Figure 2). Components are added to the project, can be connected among each other and edited to adjust parameters. This model also corresponds to the well-known RL-Glue framework [5], to which we will refer in Section V. Interacting components have to be connected by edges. The basic visual representation of a project is a tree, with the experiment as the root node and the agent and environment as its child nodes. The agent in turn can be hierarchically decomposed into child nodes, such as a policy or a learning method, which again can have child nodes, depending on the used methods and algorithms. All mentioned components are available in the editors component tree and can be added to the project by drag and drop. If the classes of environments, policies and Q-functions implement a visualization method, the graphical output can be called from the GUI. Experiments by themselves provide the visualizations of rewards and TD-errors over time.

C. Demonstration Learning

To enable demonstration learning, the environment components can send state-action pairs to the agent (Section V), where they are stored in a datastructure. The structure is represented by its own component in the project. This for example facilitates the comparison of different policy improvement algorithms for the learned behaviour.

D. Adding Custom made Components

The GUI and the underlying reinforcement learning API are designed with the intention for simple extension by new environments or agent parts. To add such a new component one has to implement it in the Java API (Section VI). Predefined Java interfaces specify, which components properties are manually adjustable. The new component will automatically be added to the editors component tree to be used in projects.

V. RL-Glue Compatibility and Interfaces to Robots or Simulations

RL-Glue is a common framework in the RL-community, that provides a standard interface to connect agents, environments and experiment programs together [5]. The editor

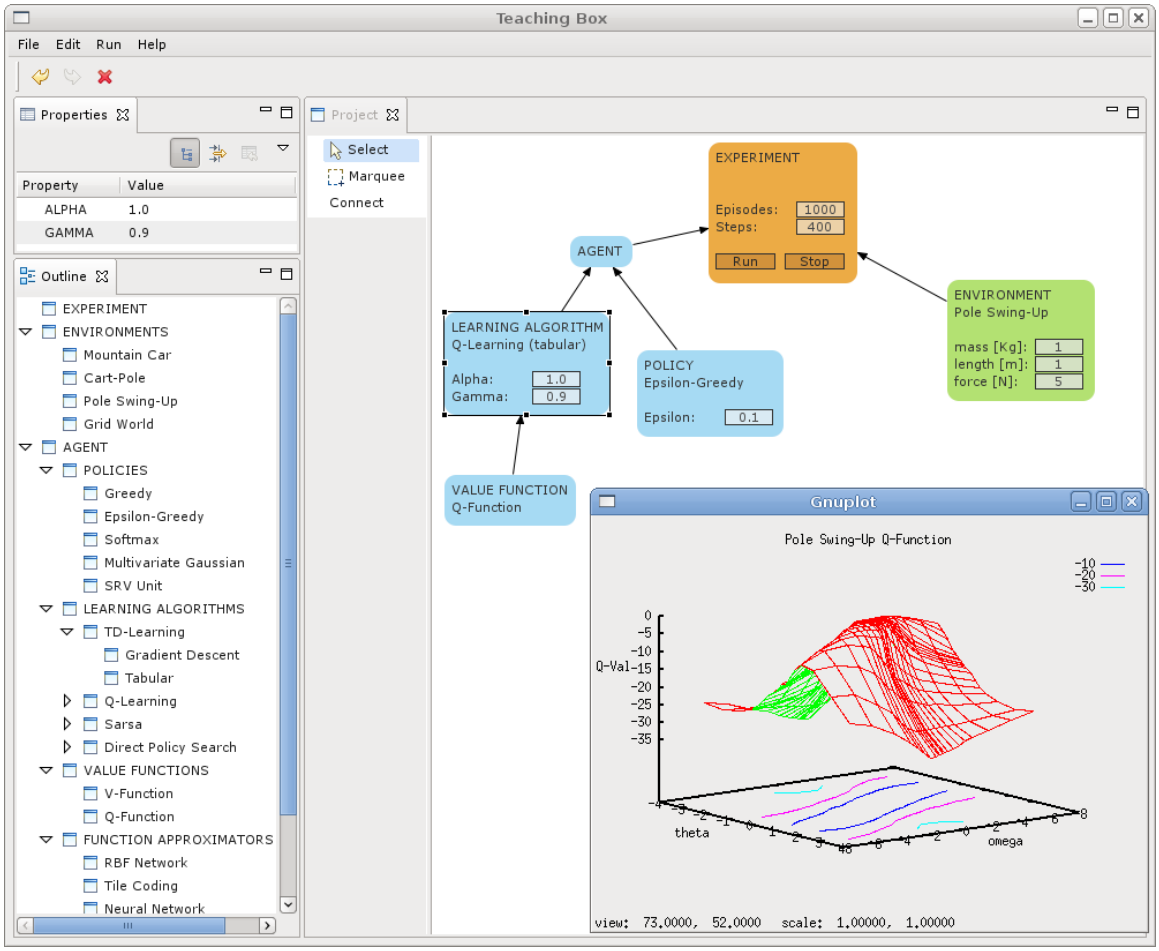

Fig. 2. The Teaching-Box GUI with the Pole Swing-Up example and visualization of the learned Q-function. The properties of the selected node (learning algorithm) can be edited in the property sheet (top left).

includes the three basic RL-Glue components, simply recognizable by the ”RL-Glue” prefix in their names. They are handled like other components with the only difference, that they have to be connected with a corresponding RL-Glue implementation. While starting the project, the Teaching-Box will automatically start the RL-Glue server and adapt all participating non RL-Glue parts by running them as RL-Glue components. Thus, existing RL-Glue programs can easily be used with the Teaching-Box.

The Teaching-Box, and consequently the graphical editor, provides a component which represents an interface to an external environment. That could be a simulation such as a Webots project or real hardware, like a robot arm. They can be connected to their editors component by interfaces from our framework, which are available in several languages and easy to use. Naturally, a connection via RL-Glue is also possible. An encapsulation of the Teaching-Box in components of the robot middleware SmartSoft is planned [14].

VI. Data Structures and Algorithms

In reinforcement learning, actions and states can be continuous or discrete, a scalar value or composed of several components. The Teaching-Box uses only vectors of real numbers. This is less overhead than introducing specific data structures for all the cases described above.

We split the reinforcement learning problem into agent, policy, environment, experiment and learning algorithm according to the description in [17]. As a result of this modular design we can guarantee the reusability of already developed software components for new algorithms and different optimization problems.

The system is adaptable, such that new algorithms can be added without much effort. There are easy to implement interfaces for the algorithms and function approximations. Therefore it is possible to use different functions (Tables, TileCoding, RBFs, Neural Networks) without the need to change the algorithm.

We implemented an already rich set of well known reinforcement learning algorithms such as Temporal Difference Learning, Dynamic Programming, Actor Critic Methods, Prioritized Sweeping and Policy Gradient Methods as described in [11], [16] and [17]. For evaluation purposes we offer the common benchmark problems like mountain car, the pole swing up task and the cart pole problem.

Evaluation of several algorithms in parallel is possible due to the use of observer patterns. Several learning algorithms

can be "attached" to the same experiment. All of them will receive the same information (s_t, a_t, s_{t+1}, r_t). The advantage of this method is, that the constraints (e.g. sensor noise, exploration) are the same for all learning algorithms.

There are many ways for an agent to treat a state from the environment. For example the state can be discretized, additional features can be added (e.g. the square of each component), use it with a function approximation like tile-coding or a any combination of them. In order to handle all of this, we introduce the concept of *FeatureFunction*. All modifications, adaptations and changes are defined as *FeatureFunctions* that can be combined in any way. In Figure 3 you can see an example of the pole balancing benchmark and the *FeatureFunction* concept. The state $(x, \dot{x}, \theta, \dot{\theta})$ observed from the environment is expanded to $(x, \dot{x}, \theta, \dot{\theta}, x^2, \dot{x}^2, \theta^2, \dot{\theta}^2)$, separated into discrete values and then used in a table as a value function.

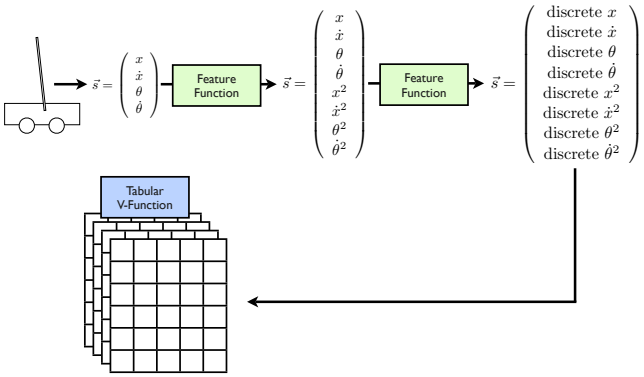

Fig. 3. FeatureFunctions can be used to transform sensor inputs.

In addition to the well known policies like greedy, ϵ-greedy and softmax action selection we provide interfaces for discrete and continuous action policies. Also every function approximation can be used to store information about actions (e.g. the likelihood to be chosen) and represent policies. This allows us also to learn a policy by demonstration and then use reinforcement learning to improve the learned behaviour.

A. Example

The following example solves the mountain car problem with gradient decent Q-learning. An adaptive radial basis function network (RBF network) is used as a function approximator. First the Q-Function and RBF network are created. Then we specify the environment and the policy to use. The gradient descent Q-leaning algorithm is attached to the agent and a new experiment is started for the given number of steps and episodes.

Listing 1. blabla

```
// default sigma for a new created RBF
final double[] sigma = new double[]{POS_SIGMA
    , VEL_SIGMA};

// create adaptive RBF network
AdaptiveNRBFNetwork net = new
    AdaptiveNRBFNetwork(sigma);

// setup Q-Function
QFeatureFunction Q = new QFeatureFunction(net
    , MountainCarEnv.ACTION_SET);

// setup environment
MountainCarEnv env = new MountainCarEnv();

// setup policy
GreedyPolicy pi = new GreedyPolicy(Q,
    MountainCarEnv.ACTION_SET);

// create agent
Agent agent = new Agent(pi);

// setup learner
GradientDescentQLearner learner = new
    GradientDescentQLearner(Q);

// learning parameters
learner.setAlpha(0.1);
learner.setGamma(0.99);
learner.setLambda(0.73);

// attach learner to agent
agent.addObserver(learner);

// setup experiment
Experiment experiment = new Experiment(agent,
     env, MAX_EPISODES, MAX_STEPS);

// run experiment
experiment.run();
```

B. Hierarchical Reinforcement Learning

One way to overcome the curse of dimensionality is the utilization of hierarchical reinforcement learning as shown in various applications [7]. The learning algorithms have to be generalized to semi Markov decision processes where the time between two subsequent actions is variable. The crucial idea for reducing the search complexity is the use of "macro-operators". In some restricted part of the state-action-space the agent learns a policy which may be optimal for that sub-space. This policy can then be stored and labeled as a macro-operator. After learning a sufficient number of macro-operators, the human trainer decides that learning on the next level starts. Here, the basic actions are the macro-operators and the state space may become larger, for example by allowing additional sensor inputs. Hierarchical Reinforcement Learning will become important for solving complex applications in the future. However, it will not be included in the first version of the Teaching-Box and it is not yet resolved which algorithms will be implemented.

C. Multi-Agent Learning

Another promising way to speed up reinforcement learning of complex tasks is cooperative multi-agent learning [13]. In the simplest case a number of agents try to solve a common task without cooperation. They receive a common global reward and each agent independently tries to optimize its

policy in order to maximize the global reward. Obviously, if the agents exchange information about their view of the world and their policies, better solutions can be achieved. Since all agents are learning and thus continuously changing, from the view of any particular agent, the environment is constantly chanching. All proposed multi-agent learning schemes have problems with convergence, even to suboptimal Nash-equilibria.

Initially the Teaching-Box will not include multi-agent learning since this area is relatively young and no favourite solution has emerged yet. Nonetheless, the open architecture of the Teaching-Box can easily be extended to include multiple agents and cooperation among them.

VII. Conclusions

The authors hope that the Teaching-Box will contribute to a faster dissemination of machine learning techniques into industrial automation and robot development.

In order to solve complex applications with high dimensional state- and action-spaces, the Teaching-Box continuously has to be adapted to the state of the art and approved new methods must be integrated. This can be achieved only if the research community cooperates and contributes their achievements to the Teaching-Box.

VIII. ACKNOWLEDGMENTS

This work has been conducted within the Collaborative Center for Applied Research on Service Robotics (ZAFH Service Robotics). The authors gratefully acknowledge the research grants of the state Baden-Württemberg and the European Union.

The authors also gratefully acknowledge Günther Palm, Christian Seemüller, Christian Bieber, Andy Stumpe, Arne Usadel, Phillip Ertle and Joachim Fessler. All the above mentioned people contributed to the success of the Teaching-Box either by giving valuable feedback or by developing use cases.

References

[1] Eclipse GEF (Graphical Editing Framework). Website. `http://www.eclipse.org/gef` [Online; accessed 03-March-2009].

[2] FANN: Fast Artificial Neural Network Library. Website. `http://leenissen.dk/fann/` [Online; accessed 03-March-2009].

[3] KNIME (Konstanz Information Miner). Website. `http://www.knime.org` [Online; accessed 03-March-2009].

[4] PIQLE: Platform Implementing Q-Learning. Website. `http://piqle.sourceforge.net` [Online; accessed 03-March-2009].

[5] RL-Glue (Reinforcement Learning Glue). Website. `http://glue.rl-community.org` [Online; accessed 03-March-2009].

[6] The Colt Project. Website. `http://acs.lbl.gov/~hoschek/colt/` [Online; accessed 03-March-2009].

[7] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Systems, Special issue on reinforcement learning*, 13:41–77, 2003.

[8] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, pages 1371–1394. Springer, 2008.

[9] H. Friedrich, S. Muench, R. Dillmann, S. Bocionek, , and M. Sassin. Robot programming by demonstration (rpd): Supporting the induction by human interaction. *Machine Learning*, 23(2):163–189, 1996.

[10] Mykel J. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour*. PhD thesis, School of Informatics, University of Edinburgh, July 2006. JRLF Website: `http://mykel.kochenderfer.com/jrlf` [Online; accessed 03-March-2009].

[11] V. R. Konda and J. N Tsitsiklis. Actor-critic algorithms. In *Neural Information Processing Systems 1999*. MIT Press, 2000.

[12] M. Kranz, R. B. Rusu, A. Maldonado, M. Beetz, and A. Schmidt. A player/stage system for context-aware intelligent environment. In *Proceedings of the System Support for Ubiquitous Computing Workshop (UbiSys 2006), at the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006)*, Orange County, California, 2006. `http://playerstage.sourceforge.net` [Online; accessed 11-March-2009].

[13] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[14] C. Schlegel. Communication patterns as key towards component-based robotics. *International Journal on Advanced Robotics Systems, Special Issue on Software Development and Integration in Robotic*, 3(1):49–54, 2006. `http://smart-robotics.sourceforge.net/index.html` [Online; accessed 11-March-2009].

[15] S. Sonnenburg, M. L. Braun, C.S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K. Mller, F. Pereira, C.E. Rasmussen, G. Rtsch, B. Schlkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, Oct 2007.

[16] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation, 1999.

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

[18] R. Tedrake. Learning control at intermediate reynolds numbers. In *IROS 2008 Workshop: Robotics Challenges for Machine Learning II*, Nice, France, 2008.

[19] A. L. Thomaz and C. Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 172(6-7):716 – 737, 2008.

[20] I. Witten and E. Frank. *Data Mining*. Hanser Verlag München, 2001. Von den Autoren in Java entwickelte DataMining Programmbibliothek WEKA: (`http://www.cs.waikato.ac.nz/~ml/weka`).