2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Neural Reinforcement Learning Controllers for a Real Robot Application

Roland Hafner
Neuroinformatics
University of Osnabrueck
Email: roland.hafner@uos.de

Martin Riedmiller
Neuroinformatics
University of Osnabrueck
Email: martin.riedmiller@uos.de

***Abstract*— Accurate and fast control of wheel speeds in the presence of noise and nonlinearities is one of the crucial requirements for building fast mobile robots, as they are required in the MiddleSize League of RoboCup. We will describe, how highly effective speed controllers can be learned from scratch on the real robot directly. The use of our recently developped Neural Fitted Q Iteration scheme allows Reinforcement Learning of neural controllers with only a limited amount of training data seen. In the described application, less than 5 minutes of interaction with the real robot were sufficient, to learn fast and accurate control to arbitrary target speeds.**

I. Introduction

A. Overall control architecture

The RoboCup initiative (www.robocup.org) provides a highly competitive benchmark environment for the development of novel approaches of intelligent autonomous robot control. In the MiddleSize League, where up to 6 versus 6 mobile autonomous robots play soccer on a 8 times 12 m field, the game has become increasingly fast in the last couple of years. This requires the use of more and more elaborated controllers on every level of the control architecture. Speed and accuracy are crucial requirements for building a competitive team. Our 'Tribot' robot is based on an omnidirectional drive, where three DC-motor driven wheels are mounted on each corner of a triangular plattform (see figure 1). Control of this omnidirectional robot is possible in any direction with potential rotation at the same time. Every movement therefore means the appropriate control of the individual speeds of the DC motors driving the wheels.

The control architecture consists of several layers. Higher layers give commands for the movement of the whole robot, in terms of translational speed in x and y direction and in terms of rotational speed. By the use of an inverse kinematics model, this command is then transformed into individual wheel speeds. For each wheel, an individual controller controls the target speed. Only if these speed controllers work fast and accurately, the global movement of the robot can be achieved. However, since these motors are mechanically coupled to each others via the plattform, their emitted forces influence and disturb each other's control task. Usually, PID controllers are used for the task of low level speed control; however due to considerable nonlinearities in the motor behaviour, they are difficult to tune and typically not optimal over the whole working range.

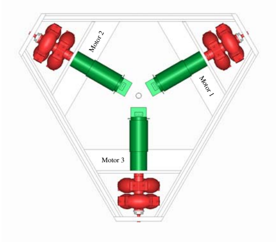

Fig. 1. The drive of the omnidirectional autonomous robot Tribot uses three omnidirectional wheels arranged in triangular shape. Each wheel is driven by a powerful DC motor.

Therefore the idea is to replace the motor speed controllers by learning controllers. Reinforcement learning controllers based on neural Dynamic Programming methods in principle allow to cope with the above mentiond requirements. They can at least approximately learn an optimal control law (e.g. a time optimal behavior) and are able to cope with nonlinearities and noise. However, when it comes to the application in real world with continuous state spaces, RL methods often suffer problems of a long learning time or of not converging at all.

The recently proposesd Neural Fitted Q Iteration framework (NFQ) [9], [10] overcomes the problem of needing irrealistic many training samples by storing and reusing previously seen transition tuples of the system in every learning iteration. We will show that using the NFQ method, we are able to learn highly effective control behaviour from scratch by direct application on the real robot.

In contrast to other work that applies RL to real world systems, we do not make use of a model or simulator to train our controllers (like e.g. in [1]). Also, no prior policy is required here (which is the case for policy gradient methods, used e.g. in [5] [6] [4]). Instead, a neural network based RL controller is learned from scratch only by the use of real transition data collected by interaction with the real system.

The paper is organized as follows: First, we show the application of an RL controller to a real, single DC motor. In particular we will focus on the fact, that using our recently developed NFQ algorithm , we are able to learn a highly effective control policy from only a very short period of

1-4244-0602-1/07/$20.00 ©2007 IEEE.

interaction (less than 5 minutes) with the real motor. We will further show, how by the use of an integrating output of the RL controller, we can achieve satisfying accuracy while avoiding to provide the controller with a large set of candidate actions. Finally, we show that the concept is powerful enough to be directly realized on a real omnidirectional robot. The additional difficulty there comes from the fact, that speed control has to be done in a kinematically and dynamically coupled system. Therefore, a large variety of different loads and forces disturbs the individual motor behavior. Additionally, the collected data is noisy. We both describe the data collection and the learning process.

II. Setting up the RL learning framework

A. Task specification

Our goal is to learn a controller for regulating the speed of each single DC motor. As a major demand, the controller should be able to regulate the motor fast and accurately to arbitrary target speeds.

To fit the controller directly to the real motor, learning will be based on interactions with the real motor only. Neither an analytical nor a simulated model will be used for learning.

In the following, we will describe the learning system set up in more detail.

B. Markovian Decision Processes

The control problems considered can be described as Markovian Decision Processes (MDPs). An MDP is described by a set S of states, a set A of actions, a stochastic transition function $p(s, a, s')$ describing the (stochastic) system behavior and an immediate reward or cost function $c : S \times A \to \mathbf{R}$. The goal is to find an optimal policy $\pi^* : S \to A$, that minimizes the expected cumulated costs

$$J^{\pi}(s) = E \sum_{t=0}^{\infty} c(s_t, \pi(s_t)), s_0 = s \qquad (1)$$

for each state. In particular, we allow S to be continuous and assume A to be finite for our learning system. The transition model p is assumed to be unknown to our learning system (model-free approach). Decisions are taken in regular time steps with a constant cycle time.

C. Choosing the actions

The accurate regulation of the motor speed at arbitrary target values would in principle require the output of continuous voltages by the controller. Therefore, even if we accept a certain tolerance in accuracy, a very large action set of control voltages is needed. However, dealing with large action sets means to have a large number of potential candidates for each decision, and this drastically increases the complexity of learning an appropriate control policy.

Therefore, we use an integrating output for our RL controller [7]. The idea is, that the controller does not output the voltage directly, but instead just decides about the decrease or increase of the voltage by a certain amount $\triangle U$. By this trick, a wide range of resulting voltages can be produced whereas the set of actions available to the RL controller remains relatively small. The price we have to pay for this, is that the state of the MDP that the controller sees, is increased by the current state of the integrating output. This adds one additional component to the input vector of the RL controller. The final action set for the controller is

$$\Delta U \in \{-0.3, -0.1, -0.01, 0.0, 0.01, 0.1, 0.3\}$$

in order to have both fast and accurate increase and decrease of the integrated ouput. The final output U of the controller (that is eventually applied to the plant) is given by $U_t = \tilde{U}_t$, where $\tilde{U}_t = \tilde{U}_{t-1} + \Delta U$ realizes the integration of the actions selected by the RL controller.

D. Determining the input representation

The input to the RL controller must represent the current state of the DC motor, such that the Markovian property of the task is captured.

For an *ideal* DC motor, the state can be sufficiently described by two variables, namely the current motor speed $\dot{\omega}$ and the armature current I.

Using an integrating output as described above requires the extension of the state representation by the actual voltage signal U. This additional input here also helps to cope with the discrepancies between the *real* and the *ideal* DC motor behavior.

Since our final controller has to deal with arbitrary target speeds, the information about the desired speed must also be incorporated into the input. In principle, we can do this by directly using the value of the target speed. However, here we are using the the error between the actual speed and the target speed, $E := \dot{\omega_d} - \dot{\omega}$. As a result, the final input to the controller consists of the four dimensional continuous vector $(I, U, E, \dot{\omega})$. Finally figure 2 shows the over all structure of the RL controller.

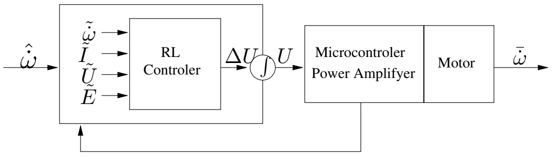

Fig. 2. Scheme of the Reinforcement Learning controller for the speed control of a single motor.

E. Choosing the immediate cost function

The immediate cost function $c : S \times U \to \mathbf{R}$ defines the eventually desired control behavior. Here, we are facing a non-episodic task, since no real final states exist, but instead regulation is an ongoing, active control task. The control objective here is to first bring the motor speed close to the target value as fast as possible and then to actively keep it at the desired level.

In terms of the immediate cost function, this can be expressed by the following choice of c:

$$c(s,a) = c(s) = \begin{cases} 0 & if|\dot{\omega}_d - \dot{\omega}| < \delta \\ c & else \end{cases} \qquad (2)$$

The first line denotes the desire to keep the motor velocity $\dot{\omega}$ close to its target value $\dot{\omega}_d$, where the allowed tolerance is denoted by $\delta > 0$.

The second line expresses the desire for the minimization of the time of the system being not close to its target value. Here $c > 0$ is a small constant value. The above framework specifies our demmand for a time-optimal controller to a region close to the target value, which reflects our desire for a fast and accurate control behaviour.

F. Neural Fitted Q Iteration (NFQ)

Neural Fitted Q Iteration [9], [10] belongs to the family of Fitted Q Iteration [2] schemes, which itself is derived from the idea of fitted value iteration [3].

The idea of classical Q-learning is to allow model-free Reinforcement Learning by iteratively learning an optimal value function over state-action pairs [13]. Typically, it is applied on-line, which means, that after each observation of a transition of the system, the corresponding value of the Q-function is updated by the following rule:

$$Q_{k+1}(s,a) := (1-\alpha)Q(s,a) + \alpha(c(s,a) + \gamma \min_b Q_k(s',b))$$

where s denotes the state where the transition starts, a is the action that is applied, and s' is the resulting state. α is a learning rate that has to be decreased in the course of learning in order to fulfill the conditions of stochastic approximation and γ is a discounting factor (see e.g. [12]). It can be shown, that under mild assumptions Q-learning converges for finite state and action spaces, as long as every state action pair is updated infinitely often. In the limit, the optimal Q-function is reached. The greedy exploitation of this optimal Q-function finally yields the optimal policy.

To deal with continuous state spaces, the above Q-learning rule can be adapted to be realized in a function approximator like a multi-layer perceptron (MLP). However, the problem with this approach is, that each update for one state-action pair might induce unforeseeable changes at the Q-values for other state-action pairs - disturbing or even destroying the effort done so far. Typically, this leads to long convergence times, requiring several ten thousands of trials [8].

The crucial idea underlying NFQ is the following: Instead of updating the neural value function on-line after each sample (which leads to the above problems), the update is performed off-line considering the entire set of transition experiences done so far. Experiences therefore are collected in triples of the form (s, a, s') by interacting with the (real) system. Here, s is the original state, a is the chosen action and s' is the resulting state. The set of experiences is called the sample set $\mathcal{D}$.

The consideration of the entire training information instead of updating on-line after each sample, has an important further consequence: It allows the application of advanced supervised learning methods, that converge faster and more reliably than online gradient descent methods.

```
NFQ_main() {
input: a set of transition samples $D$;
output: neural Q-value function $Q_N$
    k=0
    init_MLP() $\rightarrow Q_0$;
    DO {
        generate_pattern_set
        $P = \{(input^l, target^l), l = 1, \ldots, \#D\}$ where:
            $input^l = s^l, a^l$,
            $target^l = c(s^l, a^l, s'^l) + \gamma\, min_b Q_k(s'^l, b)$
        Rprop_training($P$) $\rightarrow Q_{k+1}$
        k:= k+1
    } WHILE ($k < N$)
```

Fig. 3. Main loop of NFQ .

The NFQ algorithm is displayed in figure 3. It consists of two major steps: The generation of the training set P and the training of these patterns within a multi-layer perceptron. The input part of each training pattern consists of the state s^l and action a^l of training experience l. The target value is computed by the sum of the transition costs $c(s^l, a^l, s^{l+1})$ and the expected minimal path costs for the successor state s'^l, computed on the basis of the current estimate of the $Q-$function, Q_k.

Since at this point, training the Q-function can be done as batch learning of a fixed pattern set, we can use more advanced supervised learning techniques, that converge more quickly and more reliably than ordinary gradient descent techniques. In our implementation, we use the Rprop algorithm for fast supervised learning [11]. The training of the pattern set is repeated for several epochs (=complete sweeps through the pattern set), until the pattern set is learned successfully.

The neural networks used through all the experiments were multi-layer perceptrons (MLPs), with an input layer of 5 neurons (4 as a state description, one for the action), 2 hidden layers with 10 neurons each, and a single output neuron (denoting the Q-value of the respective state-action pair).

III. REINFORCMENT LEARNING ON A REAL DC MOTOR

As a first experiment, we study the application of the RL controller to the speed control of a single MAXON DC motor. This motor is used in our mobile robot platform Tribot. One main reason for wanting a learning controller is the nonlinear behavior of the DC motor. The behavior is shown as the black line in figure 4. The diagram shows the non-linear relationship of the resulting (steady-state) speed depending on the applied motor voltage.

The cycle time between to control decisions is 33ms. The control signal to the motor is given in terms of motor voltage, given as a PWM signal.

For learning a controller from scratch, the NFQ framework as described in section II was used.

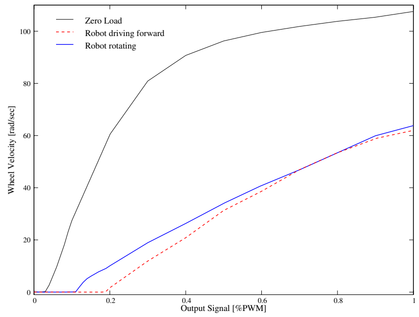

Fig. 4. Power amplifier input signal versus motor steady state velocity for motor 1 in 3 load conditions. For an ideal DC motor the correlation is linear for each load condition.

A. Data collection and learning

The NFQ learning procedure of the RL controller is based on samples of transitions, collected through the interaction with the real system. In principle, these transition experiences can be collected in arbitrary fashion. Even a random sampling procedure is possible (this will be shown later in section IV). For the following, we will do a special kind of sampling, based on an interplay of learning and data collection.

We start with a randomly initialized neural Q-function. The DC motor is then controlled by exploiting the current Q-function in an ϵ-greedy manner. As can be expected, at the beginning, the controller does not do anything reasonable - besides collecting transition samples. In figure 5 the system behavior in the initial learning phase is shown. The policy produces a highly varying output voltage $\hat{U}$ which causes a highly fluctuating motor speed.

After each 100 control cycles (corresponding to 3.3 seconds), the episode is terminated. A new target speed is randomly set and again the controller starts interacting with the real motor. The data collected is added to the transition data set. After each 2 episodes, the NFQ -procedure is called with the set of all transition samples collected so far. The resulting new Q-function is then greedily exploited within the RL controller for the following 2 episodes. After that, again the NFQ -procedure is called and so forth.

The parameters for NFQ are the following: We used an MLP with 5 input, 2 layers of hidden units with 10 neurons each and one output neuron. Each neuron used a sigmoidal activation function. Learning was repeated for 300 epochs using the fast supervised learning algorithm Rprop (the time for the 300 epochs depends on the number of transitions in the data set; the maximum time used here was less than 5 minutes on a Pentium PC). Exploration was done in 20 percent of the decisions while collecting data. The tolerated deviation from the target speed was set to $\delta = 5 rad/s$.

Using NFQ , learning was achieved very effectively: after only 60 episodes of interaction with the real system, a well performing RL controller was learned from scratch. This corresponds to only 198 s interaction time with the real motor, only little more than 3 minutes. Alltogether, 6000 transition samples were used for training. The resulting performance of the learned RL controller is highly satisfying, both with respect to speed and accuracy.

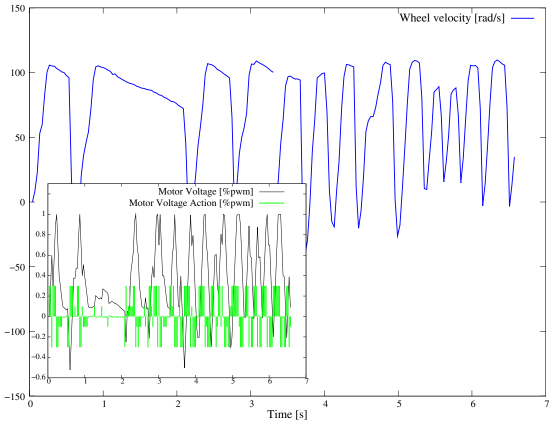

Fig. 5. Behavior of the RL controller in the early stages of learning. The behavior of the motor speed is rather arbitrary, since the controller has not learned a useful policy yet. All the transition data is collected, stored and used for training. The small figure show the working of the integration of the RL actions: the green lines denote the actions, that are selected by the RL controller which are integrated to result in the final voltage applied to the motor (black signal).

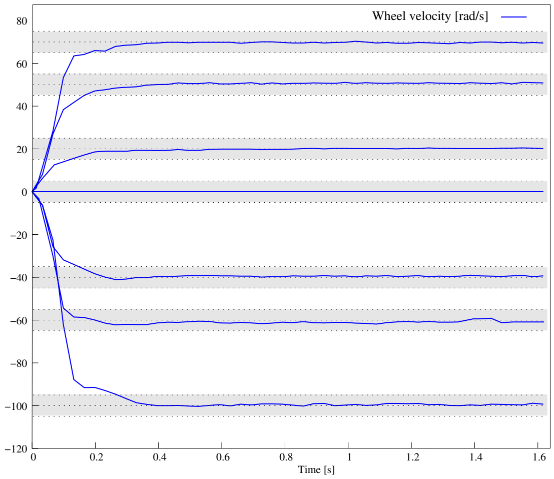

Fig. 6. RL control of the real DC motor on a wide range of target speeds. The controller works both fast and accurately.

B. RL Controller performance

As required by the specification, the RL controller should work for a wide range of target speeds. As figure 6 shows on a set of example settings for the target speed, the RL controller has perfectly learned to solve this task. The target speeds are reached quickly and reliably (the zones marked with grey denote the tolerated region around the target speed, where the immediate reinforcement signal is zero).

As already pointed out, to reach a wide range of target speeds, the controller must also be able to produce a large range of voltages that are applied to the DC motor. In order to circumvent a large set of potential candidate actions, we provided the RL controller with an integrating output (see section II-C). Obviously, the controller has learned to use this

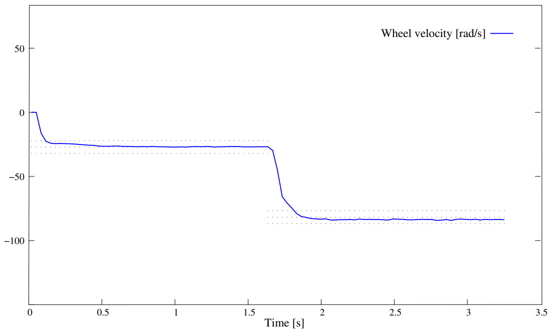

Fig. 7. Wheel speed characteristics of the learned controller for a sequence of two different set points.

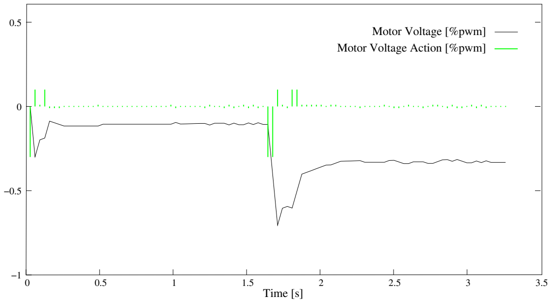

Fig. 8. The generated output voltage of the controller (black) produced by the actions plotted as green peaks for the control trajectory showed in figure 7.

instrument successfully. Figure 7 shows a sequence of two target speed settings, that are successfully achieved (the blue line denotes the motor speed). Figure 8 shows the control signal that is generated to perform this behavior. The black signal shows the integrated signal, that is applied to the motor, and the green lines show the actions, that are selected by the RL controller to generate the behavior. To achieve a quick regulation, first a highly negative voltage is produced (by the application of a large negative $\triangle U$, and the voltage is increased again to finally be kept at a voltage, that keeps the motor at the desired speed (note that a direct application of this final voltage would only lead to a slow reaching of the target speed only).

C. Comparison to a PID controller

In figure 9 one of the problems of a linear PID controller in this nonlinear control problem is shown: The DC motor must be controlled first to a low backward speed of approximately -28 rad/s followed by a target speed of approximately -82 rad/s. This covers a range from 30 percent to 80 percent of the maximal speed of the motor. As can be seen in figure 4 the system behavior is strongly nonlinear in this range. Tuning a linear PID controller therefore means to find a compromise between quick regulation and avoiding overshooting behavior - which is difficult to achieve, since the system itself behaves nonlinear. The red line in figure 9 shows this problem: whereas for a small negative speed we already observe an overshooting, for a large negative speed, the PID controller is somewhat too cautious and reaches its target slowly. Of course, enhanced classical control schemes can cover these effects in a better manner as shown here, but they do it by the price of much higher effort in modelling and tuning of controller parameters.

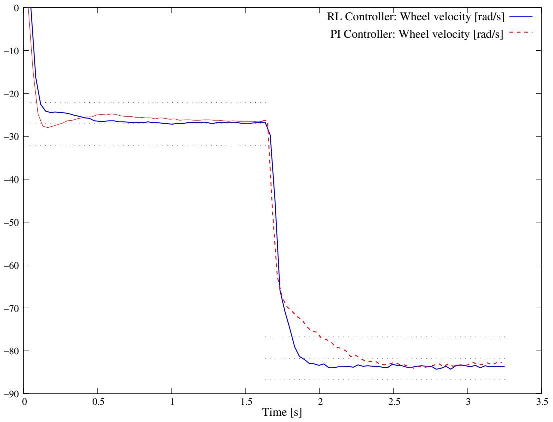

Fig. 9. Comparison of a PID controller and the learned RL controller for a sequence of two different set points.

The RL controller on the other side (blue line in figure 9 learned a highly effective control policy by pursuing its learning goal as specified in the immediate cost function ('minimum time control'), regardless of the nonlinearities of the plant.

IV. Reinforcement Learning on the real robot

When applying the RL motor speed controller as trained in the previous section to our omnidirectional robot, the performance is not satisfying. The controller manages to regulate the motor to a steady-state - but the resulting speed is much below the desired target speed. The reason for this can be found in figure 4. Under varying load (red and blue line), the relation between input voltage and resulting motor speed is drastically changed. Since the learning controller so far has only seen the motor behavior under no load, it is not surprising, that it is not suited for the loads that occur in the robot environment.

One possibility now would be to artificially produce varying loads on the DC motor and repeat the learning procedure of the previous section. Another way, which we will pursue here, is to directly collect data that are measured on the real robot. This has the advantage, that the load profile is generated by situations that actually occur on the real robot.

One problem is, that the data collected on the real robot is much more noisy than the data collected for a single DC motor in the load free case. Reason for this are the changing contact points of the omnidirectional wheels and also small misalignments in the horizontal axis. However, as shown in the following, the RL controller is still able to cope with all these problems and learn a highly satisfying control behavior.

A. Data collection and learning

The RL controller for the motor speeds is integrated at the lower level within the control software for our mobile robot.

The complete software runs on a laptop, mounted on the robot. Cycle time is 33 ms.

In contrast to the transition sampling procedure that interleaves learning phases and data collection, we decided to pursue another way here. Data was collected completely randomly, that means, random control signals were emitted to each of the three motors and the resulting transition samples were collected. This is a valid procedure, since NFQ does not rely for the samples to be collected in a certain fashion.

In particular, we run 50 episodes with a duration of 150 control cycles each (corresponding to a duration of 5 seconds), applying purely random control signals. This gives an overall of 50 * 150 = 7500 transition samples. Since the data was collected simultaneously for all three motors, this makes an overall of 3*7500 = 22500 transition samples that can be used for training within the NFQ framework. The whole process of data collection on the real robot needed only $50 * 5s = 250s$, which is little more than 4 minutes.

This data is then fed into the NFQ -learning procedure. For the neural network, the same MLP structure as before (5-10-10-1) was used. As before, the number of epochs was set to 300. Using the same parameters for varying tasks somehow underlines the robustness of the approach which is particular useful when it comes to application on real systems, as we do it here.

After only 30 iterations through the NFQ -loop, a highly effective controller was learned.

B. RL Controller performance

In figure 10 the learned controller is shown running on the real robot. The global drive commands used as a demonstration here are 'drive forward with 0.5 m/s' and then 'rotate by $2rad/s$'. The inverse kinematics are used to deliver the respective target speeds for each motor. The task of the learned controller is then to regulate each motor to the desired motor speed.

As shown in figure 10 the neural controller has learned to control the motors very quickly and reliably to their desired target speeds. A highly satisfying fact is, that the learned speed controller works reliably under the wide range of actual loads that occur within the real robot movement. It has even learned to deal with the considerable noise that occurs in the measured data.

V. Conclusion

We presented a succesful example of learning a highly effective control policy by the use of transistion data collected by interaction with a real robot system only. In contrast to other approaches that apply reinforcement learning to real systems, no initial knowledge, especially no model, no simulator and no initial policy, is required here. The NFQ procedure, which is used to solve the RL part, makes highly efficient use of the sampled data.

Using the NFQ approach we tackled the problem of a DC motor speed control application for a three wheeled omnidirectional mobile robot. In this application interesting problems of rapidly changing loads, nonlinear system behavior and noisy sensor data occur. Even in this scenario, the learned controller is able to regulate each wheel fast and accurately at the desired speed. Only a couple of minutes (less than 5 minutes) of random interaction with the real robot were needed to finally achieve a highly effective control policy from scratch.

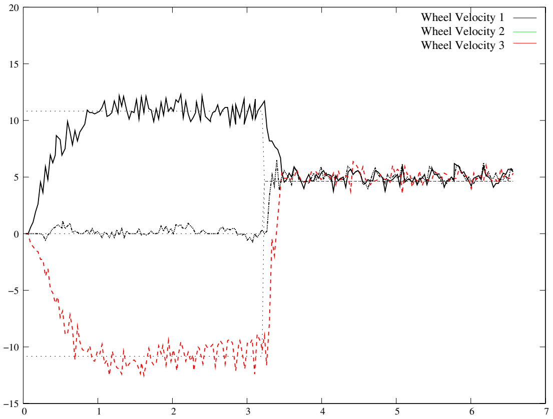

Fig. 10. The learned controller tested on a follow up control on the real robot. The robot was driven forward with $0.5\frac{m}{s}$ changing to a rotational velocity of 2 $\frac{rad}{s}$. The controller is able to achieve the velocities for all three motors under the presence of noise generated from the wheels.

References

[1] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control.* PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[2] D. Ernst and a. L. W. P. Geurts. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[3] G. J. Gordon. Stable function approximation in dynamic programming. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, San Francisco, CA, 1995. Morgan Kaufmann.

[4] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion, 2004.

[5] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots.*, 2003.

[6] J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2005)*, 2005.

[7] M. Riedmiller. Generating continuous control signals for reinforcement controllers using dynamic output elements. In *European Symposium on Artificial Neural Networks, ESANN'97*, Bruges, 1997.

[8] M. Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Neural Computing & Applications*, 8:323–338, 2000.

[9] M. Riedmiller. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proc. of the European Conference on Machine Learning, ECML 2005*, Porto, Portugal, October 2005.

[10] M. Riedmiller. Neural reinforcement learning to swing-up and balance a real pole. In *Proc. of the Int. Conference on Systems, Man and Cybernetics, 2005*, Big Island, USA, October 2005.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586 – 591, San Francisco, 1993.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning.* MIT Press, Cambridge, MA, 1998.

[13] C. J. Watkins. *Learning from Delayed Rewards.* Phd thesis, Cambridge University, 1989.