Evaluation of Machine Learning Methods on a Swinging Humanoid

Hannes Gräuler, Robin Kogelberg, Thomas Lampe, Jessica Meyer, Philipp Niermann, Mareike Paul, Roland Hafner, Sascha Lange, Martin Riedmiller

***Abstract*— We show that, given a specific task, a variety of machine learning algorithms can be applied. The approaches are evaluated in terms of performance in a simulated environment and applicability to a real-world task. We argue that no approach performs optimally in all aspects considered.**

I. Introduction

Generally, there exists a multitude of fundamentally different machine learning approaches to deal with a given task. There are two major distinctions that can be made: On one hand regarding the learning mechanism between evolutionary algorithms and reinforcement learning; on the other hand into feedforward and feedback architectures that are learned.

In this paper we contrast three learning approaches for the same tasks. The application performed for the comparison was a swinging task in which a humanoid robot was to perform simple acrobatics on a high-bar. This task was chosen due to the fact that it is not straightforward to find an optimal solution manually. Different hand-crafted strategies, like sinus oscillation and changing servo activation at the point of inflection or the point of lowest altitude, were tested on the real robot and in simulation prior to this study. Sinus oscillation induced a movement although not very high, while the strategy of changing servo action in point of inflection did not manage to initiate swinging but gave high results, when the robot was manually started by being pushed. This led to two basic, apparently non-correlated tasks: reaching an appointed height as fast as possible in order to even begin swinging, and reaching as high an amplitude as possible. An additional task was coping with an external disturbance.

Three machine learning algorithms were used for learning these tasks: A standard evolutionary algorithm (cf. [5]) was used in a feedforward setup, while Neural Fitted Q-Iteration (NFQ, cf. [11]) represented the recently proposed category of fitted value iteration algorithms [4], such as Fitted Q Iteration [3] and LSPI [8]. In addition, an approach that learned neural oscillation with evolutionary algorithms was attempted, initially using a feedforward mechanism as well but being extended to include feedback information.

II. Aufbau

A. Tasks

performing swinging taks with different aspects of evaluation There were three different tasks the three machine learning methods had to fulfill:

Institute for Computer Science, University of Osnabrück, Germany hgraeule@uos.de

Speed The first task for the robot was to reach an appointed height (0.3 rad $\approx 17.2°$) as fast as possible. The trial length was limited to 400 cycles.

Height As second task, the robot had to perform a movement as high as possible. Time is irrelevant for the benchmark, but the trials were cut off after 400 cycles.

Noise The third task was testing the reaction of the robot to a glitch; the robot should not be tripped up after it. Here the disturbance was a loss of communication in cycles 100 through 107. This task was not trained as both tasks above, but was tested with some strategies learned in task one or two.

B. Configuration

1) Real robot: For the experiments on a real humanoid robot a Robotis Bioloid[1] – a 32 cm tall humanoid robot with 16cm long arms – in standard configurationwas fixed with his hands to a rotating high bar (height 64 cm).

The movement of the robot only was induced by moving knee and hip joint servos, all other joints are blocked.

The rotation of the high bar axis was measured to calculate the deflection θ (see Fig. 1). Joints used in the tasks were the hip (α) and knee (β).

In the simulation each program cycle possessed a length of 100 ms, whereas it was 250 ms on the real robot.

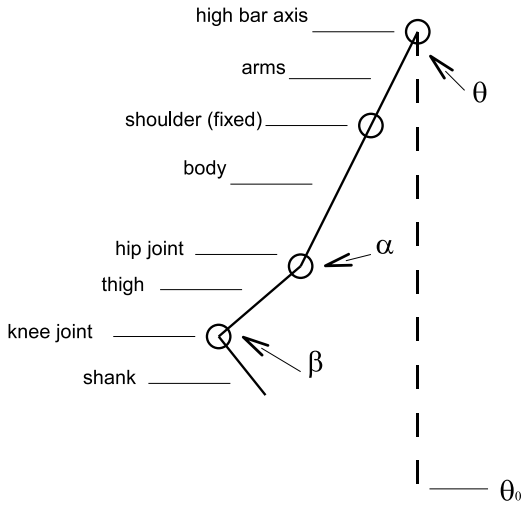

Fig. 1. Sketch of real robot and angles

[1] http://www.robotis.com

2) Simulation: To prevent too much load on the real robot, most testing was done in a simlulation environment called Simloid[2], which is based on the Open Dynamics Engine (ODE[3]) and was developed at Humboldt-University Berlin to simulate the Robotis Bioloid. The setup in the simulation was slightly different: the hands were not attached to the high bar, but the shoulders were fixed to the bar using invisible rods; also the arrangement of the torso servos was different, although the legs were identical.

III. Comparison

In this section the three different approaches are described.

A. Evolutionary algorithms

1) Setup: The first approach investigated was to use evolutionary algorithms (EA, cf. [1], [5]), which are often used in optimization. Several parts of evolutionary algorithms, such as crossover and mutation, have to be implemented depending on the problem, described below.

The amount of individuals is set to 26, the number of generations is 20. Each individual stores information about the time when the legs of the robot shall change their position, and two positions of the servos for the knees and the hips. The legs are either at the front or at the back, positions in between are not stored. We assume that the points, when the robot shall change positions, can be described as a cubic function the coefficients of which shall be learned.

We tried three different approaches to perform the tasks described above. In the first one we learned the coefficients of the cubic function, the second one was to learn the positions of the servos. These two approaches were combined in the third one: First the coefficients of the function are learned. The results are used to learn the position of the servos, which in turn are used to learn the time again, and the cycle repeats three times.

As selection routine we chose the $(\mu + \lambda)$-evolutionary strategy with $\mu = \lambda = 25$ (cf. [2]).

For crossover we computed for each coefficient the average value of the parents' value. For the servos, the crossover ist done by combining the positions of each servo. Mutation is done on the children by multiplying one variable with either 0.9 or 1.1. After each reproduction the worst 25 individuals are replaced by the children as generally described in a steady-state evolutionary algorithm (cf. [9]).

The fitness function depends on the task: for the height task, the fitness is the height achieved. Reaching a certain height as fast as possible leads to a fitness showing the number of cycles needed until the height is reached. For the third task, the final results of the other tasks are run with a glitch and the fitness values are compared.

2) Results: The results for the speed task are shown in Fig. 2. The lower lines belong to the approach to learn the function to given servo positions, the upper ones show the results for the approach to learn the positions. The approach last mentioned does not implicate any result: The achieved height is less than 0.3. Thus combining the two approaches is comparable to running the algorithm for learning coefficients three times and is therefore not considered any more.

[2]http://www.robocup.de/AT-Humboldt/simloid.shtml
[3]http://ode.org/

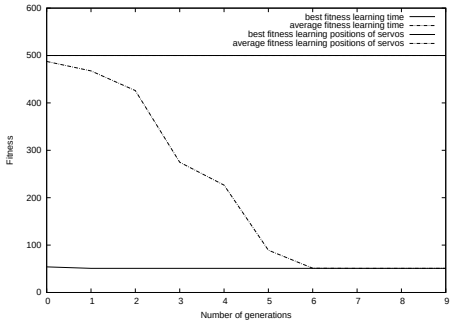

Fig. 2. Fitness achieved with EA learning times or positions for swinging fast. The upper line combines the average and the best fitness when learning positions. The lower the lines, the better the the strategy.

Fig. 3 shows the fitness results of an evolutionary algorithm performing the height task. The upper lines are the result for learning the time, the lower ones visualize the results for learning servo positions. As it meight by seen easily learning times shows better results than learning positions. This may be due to the fact that the coefficients of the function are set to parameters not optimal for the problem. Thus it is interesting to have a look at the results when combining the two approaches. Fig. 4 illustrates the results.

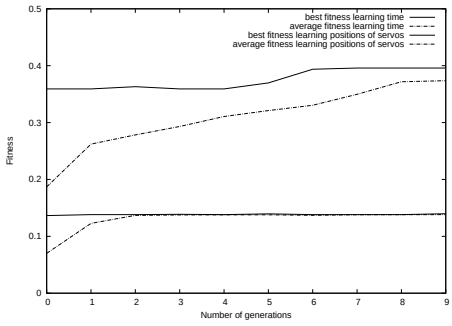

Fig. 3. Fitness achieved with EA learning time or positions for swinging high. The lower lines illustrate the results for learning positions, theupper lines show the results for learning time. The higher the line, the better the result.

The last task was to react to the glitch. For this task we took the best results of the first tasks. Except for the height task when learning positions, all results with the disturbance are worse than the trials without. When performing the speed task, no deterioration of the fitness can be achieved as the glitch occurs in cycle 100 which is never reached.

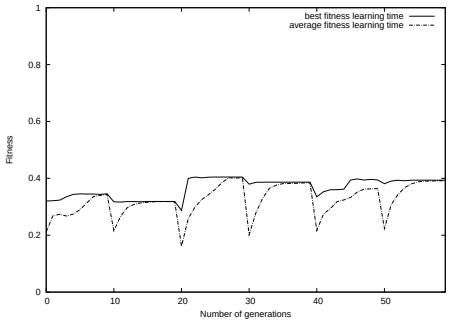

Fig. 4. Fitness achieved with EA learning times and positions for swinging high. The optimization of each approach was repeated three times, taking the results of the run before and consisting of 10 generations.

TABLE I

RESULTS WITHOUT AND WITH A GLITCH

Task	Approach	Without glitch	With glitch
Height	Learning Time	0.395854	0.33877
Height	Learning Positions	0.139738	0.153971
Height	Hybrid	0.404473	0.334218

B. Neural oscillations

1) Setup: Provided that the swing-up task can be solved by an oscillatory movement of the hip servos, the applicability of neural oscillations was investigated as a second approach.

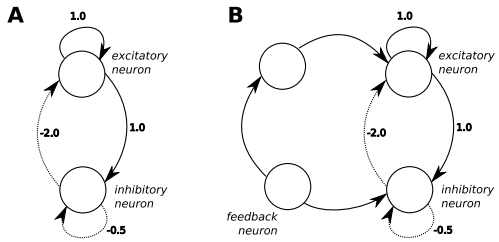

Fig. 5. Two designs of oscillatory neural networks with initial weights. Network A is capable of producing a simple oscillation that can be used for feedforward control. In contrast, network B is able to integrate feedback information through additional neural units.

Using recurrent networks as neuro-controllers for robots has previously been studied [6]: A way to model such a neural controller is shown in Fig. 5 as network A. The top neuron has only excitatory outputs, the bottom one only inhibitory. The dynamics of both neurons is described with a differential equation with certain parameters. The neural potential from the excitatory neuron is then utilized as set point α of the hip servo. The knee joint angle β is being kept fixed.

There exists a framework for neuro-evolution of complete network topologies [12] to adapt a network to a specific task, but we implemented a straightforward evolutionary algorithm that does not change the network topology, but simply modifies the neural parameters and connection weights (and thus the oscillation amplitude and frequency). For the height task, the fitness f_1 is the maximum of all reached angles θ_t. For the speed task, the fitness f_2 equals f_1 in case that it is less than 0.3, or $0.3 + \frac{400-x}{400}$ otherwise, where x is the cycle where 0.3 is reached for the first time.

Obviously, the oscillation generated by such a network is a feedforward control, since it incorporates no feedback from the external world. Due to that fact, the network is expected to perform badly when a glitch is induced. We modeled such a glitch by not updating the internal network state for the cycles in question.

In order to handle such disturbances properly, the network needs to be fed with some kind of external feedback. A second network, whose layout corresponds to network B in Fig. 5, was implemented. Two additional neurons were introduced and the potential of the *feedback neuron* was always set to the current deflection angle θ. The weights and internal parameters of all neurons were then learned again by the evoluationary algorithm.

2) Results: The fitness progression for the height task over 400 generations is shown in Fig. 6. While a maximum deflection angle of 1.3 was achieved for the height task, the maximum fitness for the speed task was 1.12. This implies that the network was able to reach an angle of 0.3 after 72 cycles.

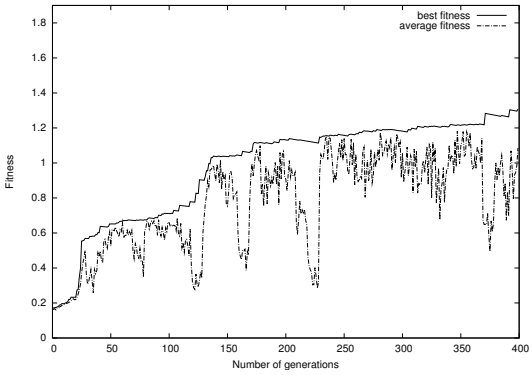

Fig. 6. Fitness progression of network A for the height task over 400 generations.

As expected, network A was not able to cope with the an induced glitch in a satisfying manner. Fig. 7 shows the servo set point together with the deflection angle for the first 300 cycles of a run with induced glitch.

The results achieved with network B were promising: Not only is the network now capable of dealing with an induced glitch (see Fig. 8) but the network also performs much better in the height task as it reaches a maximum angle θ of 1.74 (see Fig. 9). However, the performance in the speed task was weaker – network B needs 90 cycles to reach an angle of 0.3.

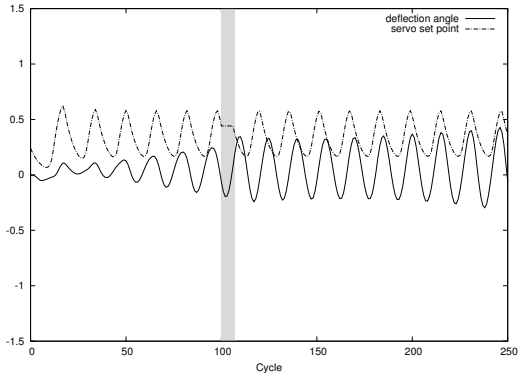

Fig. 7. Network A performs badly when a glitch is induced. The shaded area corresponds to the time where the network state is not updated.

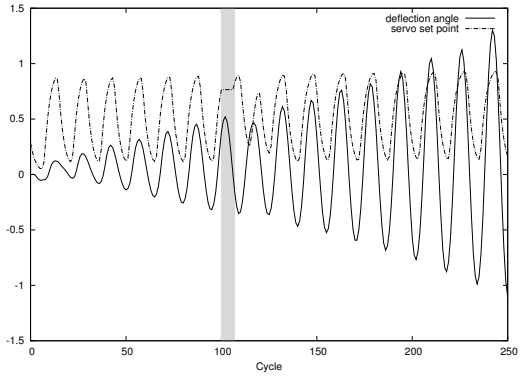

Fig. 8. Network B was able to cope with the disturbance. After just two periods the set point oscillation was back in sync.

C. Neural Fitted Q Iteration

1) Setup: The same setup as described in section II-B was used once more, this time utilizing the NFQ algorithm [11] for learning. As NFQ is a reactive approach, it requires knowledge of the learning system's current state at every decision point in order to select an appropriate action. Here, the state consisted of the current and previous amplitude θ_t and θ_{t-1}, as well as the velocity $\Delta\theta_t$, the acceleration $\Delta\Delta\theta_t$, the hip servo position α_t and its revolution speed $\Delta\alpha_t$. The possible actions were binary, corresponding to two poses of knee and hip joints: one with both being angled backwards, and one with the knees straight and legs angled forward. State and action were scaled in the interval $[0; 1]$ and passed to a neural network with two hidden layers of size 10 each, which approximated the expected path costs $Q(s, a)$ and was simulated using the n++ framework [10].

During learning, the costs $Q_\epsilon(s_t, a_t)$ for each observed transition $s_t \xrightarrow{a_t} s_{t+1}$ were estimated by direct transition costs of $0.02 \cdot (1 - \frac{|\theta_t - \theta_0|}{|\epsilon - \theta_0|})$ – weighted with the how close the system was to the goal heights $\theta_0 \pm G$, in order to encourage high amplitudes in general – added to $J(s_{t+1})$, which was discounted with factor $\gamma = 0.95$. Costs of 0 were assigned if the system had entered a terminal state, which was defined as any state with $|\theta_t - \theta_0| > \epsilon$. The margin ϵ was set to

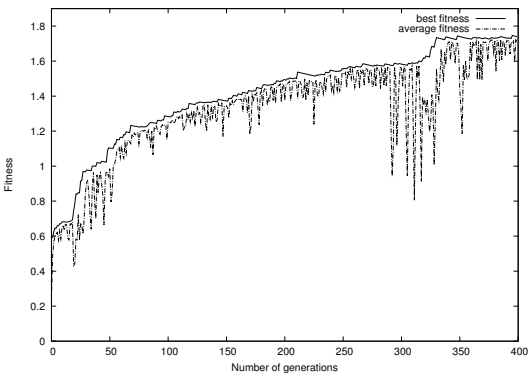

Fig. 9. Fitness progression for the height task with feedback network B.

0.3 in the speed condition; in the height condition, the costs were calculated as $0.4 \cdot Q_{0.3} + 0.6 \cdot Q_{\theta_{max}}$ – this was done in order to aim for a rotation as high as possible, while still retaining an easy goal needed to begin swinging in the first place.

The learned policy of the robot was represented only implicitly in the Q-function. The two possible actions were evaluated online during the trials and the one with lower expected costs were chosen. There were two distinct stages during each trial: A test stage lasting 400 program cycles, during which the policy learned thus far was performed in order to determine the current performance of the system; as well as a learning stage, during which exploration was performed over the optimal action. The agent would use an exploratory action with a probability of $t^{-\frac{1}{8}}$ in trial t, thus decreasing over time; the action chosen was a_{max} or a_{min} with a chance of 20% each, or a_{t-1} otherwise. The exploration stage ended as soon as the robot entered a terminal state.

2) Results: Fig. 10 shows the development of the maximum amplitude for each test stage during the height condition. While an increase in performance can be seen particularly in the decreased occurrence of low-performance policies, the learned strategy is clearly not stable.

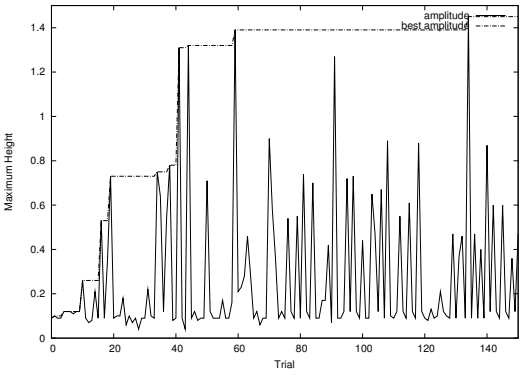

Fig. 10. Amplitude achieved with NFQ during the height condition.

The speed condition shows a very similar development.

Strategies that are capable of bringing the system into a goal occur as early as after the fifth trial, although similar as before, the fitness does not converge within the given experiment. Non-proper policies occur throughout the entire run, and the policy does not converge within the allocated time frame. Achieving an optimal time within the experimental run took much longer with 237 trials. The evaluation of strategies showed a layer-like pattern, with results occurring only around a few values of the fitness function. This behaviour results from the task itself: Each layer represents a certain distinct number of turns needed to reach the terminal state.

In the final noise condition, the strategy learned during trial 135 of the height condition – a time when the strategy was expected to generalize far enough to deal with unknown regions of the state space – was used to test the susceptibility to interruptions. Fig. 11 displays the amplitude and chosen action throughout one test trial, both with and without disturbance. The interference produces virtually no effect regarding the system's amplitude, safe for the arc directly following it. Afterwards, the system immediately compensates, increasing its amplitude again.

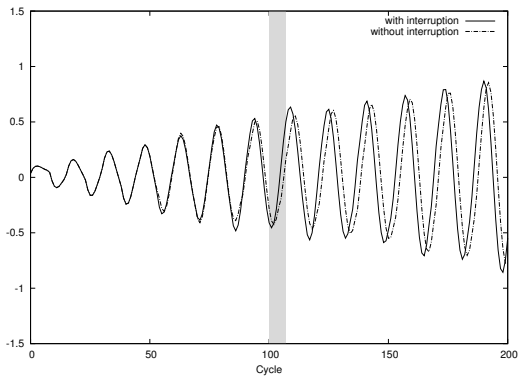

Fig. 11. Amplitude during a single trial with an interruption between cycles 100 and 107.

Since the approach used here had produced visible results in a reasonably low number of trials, it was additionally evaluated on the real Bioloid system. The setup was virtually the same, with the only difference being the shorter trial duration of 200 cycles, due to the increased cycle length. Only the speed condition was performed, with the results shown in Fig. 12. Like in the simulation, a proper policy could be achieved very quickly – trial 2 in this case, with a required time of 3.75 seconds – while the performance remained equally unstable overall; although the duration of the experiment was kept very short, due to the high susceptibility to mechanical failure of the robot, making convergence unlikely either way.

Also, the system remained equally tolerant against disturbances, as can be seen in Fig. 13, which plots an interruption for the policy obtained in trial 16. Again, the interruption does not produce any major effect.

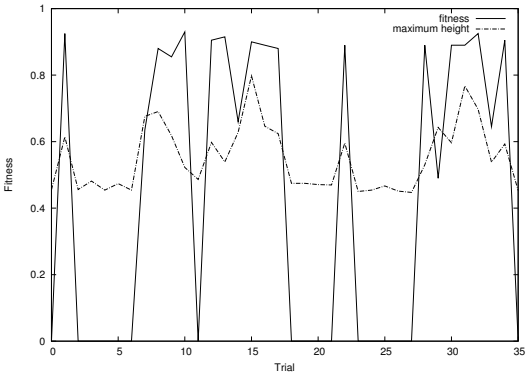

Fig. 12. Fitness achieved with NFQ during the speed condition on the real robot. The fitness score is inversely proportional to the number of steps needed, with 200 or more steps yielding fitness 0. The height reached in each trial has been included for reference, but was not used for learning.

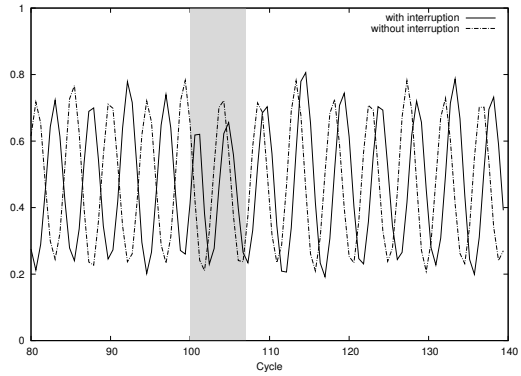

Fig. 13. Amplitude during a single trial with an interruption between cycles 100 and 107 for the real robot. The fluctuation in swinging height results from the low communication frequency of the hardware.

IV. Conclusions and Future Works

A. Conclusions

When comparing the preceding results of the different learning approaches as shown in Table II, one major difference, particularly in the light of the intended application on a real robot (see Fig. 14), was the time required to learn the task at hand. NFQ clearly outperformed the competition in this respect, reaching viable policies within only a few trials, compared to hundreds of generations consisting of tens of trial each for the evolutionary approaches.

The drawback of the approach becomes apparent as well. The quality of the solution varies wildly over trials, with only a marginal trend for improvement visible. Even though one might hope for more stable results if the experiment

TABLE II

	Condition		
	Speed	Height	Noise
Evolution	49	0.40	-
Oscillation A	72	1.30	-
Oscillation B	90	1.74	+
NFQ	32	1.45	+

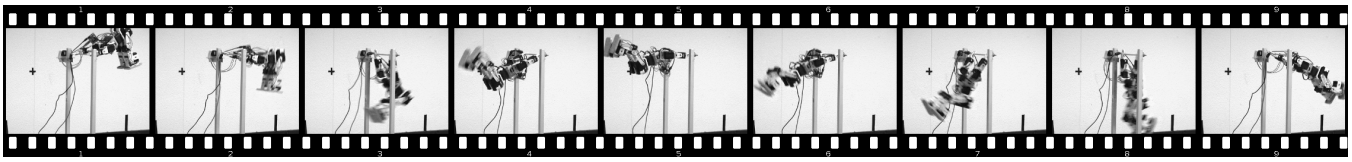

Fig. 14. Example swinging cycle on the Bioloid, illustrating a strategy achieving maximum height.

had been continued longer (which was not feasible here, due to the steadily increasing duration of each trial's learning stage as the number of observed transitions grew larger), convergence would not be guaranteed for the approach. Thus, one is forced to manually select the policy that scored best during the experimental run - which is fortunately viable in tasks such as the one considered here, where the quality of a strategy is easy to assess on-line and the best one can be retained.

In terms of performance, feedback-based neural oscillation produced clearly the best results in the height condition, although it remains unclear whether NFQ could eventually reach the same height given more time. Still, the latter always suffers from the drawback that unlike the oscillating system, it does not possess the a priori knowledge that a sinus-like pactivity pattern should be used to achieve swinging. Even later policies tend to revert to a strategy of remaining in one pose and switching only momentarily to the other upon reaching a certain height, as seen in Fig. 15.

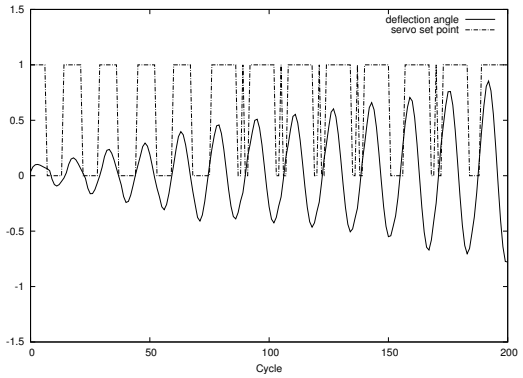

Fig. 15. Amplitude and servo control for the NFQ-policy achieving best results during the height condition.

However, both evolution an particularly NFQ reached the goal height during the speed task considerably faster than neural oscillation of either type. This is likely owed to their ability to perform even non-rhythmic actions, whereas neural oscillation was dependent on moving in a pattern that even if not neccessarily sinus-like, was nevertheless cyclic.

Disturbances in the system were predictably not handled easily by the feedforward control systems. However, it could be seen that the ability to adapt to external influence is in no way unique to classical reactive approaches such as NFQ, but could also be handled with an adapted design of neural oscillation.

B. Future Works

There are several possible approaches to improving the speed and performance of the learning systems described here. Some effort has been made to overcome the main drawback of the NFQ algorithm observed, namely the lack of policy convergence. [7] describe an approach where the cost approximators learned over successive iterations are eventually polled offline and a policy generated based on a majority vote, yielding a strategy performing better than those in each individual step.

Another possible route to take lies in the transfer of simulation results to the real system. While a policy learned in the simulation could not be directly applied on the Bioloid due to differences in joint configuration, cycle length, swing setup and encoder ranges, the results thus acquired might at least serve as a starting point for learning in a real system, potentially cutting down the required time to acceptable levels.

REFERENCES

[1] T. Bäck and H. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Journal of Evolutionary Computation*, Vol. 1, No. 1:1–23, 1993.

[2] H.-G. Beyer. *The Theory of Evolutionary Strategies*. Springer, Berlin, Heidelberg, New York, 2001.

[3] D. Ernst, P. Geurts, L. Wehenkel, and L. Littman. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:2005, 2005.

[4] 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.

[5] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[6] M. Hülse, S. Wischmann, and F. Pasemann. Structure and function of evolved neuro-controllers for autonomous robots. *Connection Science*, 16(4):249–266, 2004.

[7] T. Kietzmann and M. Riedmiller. Selecting the right patterns and policies in reinforcement learning for a real world application. In *26th International Conference On Machine Learning*, submitted.

[8] M. G. Lagoudakis, R. Parr, and L. Bartlett. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:2003, 2003.

[9] V. Nissen. *Einführung in Evolutionäre Algorithmen*. Vieweg, Braunschweig, Wiesbaden, Germany, 1997.

[10] M. Riedmiller. N++: Simulator für mehrschichtige neuronale netztopologien, 1997.

[11] M. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *16th European Conference on Machine Learning*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328, Porto, Portugal, Oktober 2005. Springer.

[12] K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1757–1762, Washington, DC, USA, 2002. IEEE Computer Society.