Learning to Drive and Simulate Autonomous Robots with Reinforcement Learning

Alexander Gloye, Fabian Wiesel, Cüneyt Göktekin, Anna Egorova, Mark Simon, Oliver Tenchio, and Raúl Rojas

Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany
http://www.fu-fighters.de

Abstract. We show how to apply learning methods to two robotics problems, namely the optimization of the on-board controller of an omnidirectional robot, and the derivation of a model of the physical driving behavior for use in a simulator.
In the first part of the paper, we show that optimal control parameters for a PID controller can be learned adaptively by driving the robot on a field while evaluating its behavior. A computer adjusts the parameters after each driving experiment using reinforcement learning. After training, the robots can follow the desired path faster and more elegantly than with manually adjusted parameters.
In the second part of the paper, we show how to learn the physical behavior of a robot. Our vision system tracks mobile robots and records its reaction to driving commands. The system learns to predict the position of the robot in future camera frames according to its reaction to the commands. The learned behavior can then be used in our simulation of the robots. Instead of having to adjust the physical simulation model whenever the mechanics of the robot changes, we can just relearn the behavior of the modified robot in a few minutes. The updated simulation reflects then the modified physics of the robot.

1 Learning in Robotics

When a new robot is being developed, it is necessary to tune the on-board control software to its mechanical behavior. It is also necessary to adapt the high-level strategy to the characteristics of the robot. Usually, an analytical model of the robot mechanics is not available, so that analytical optimization or a physical simulation are not feasible. The alternative to manual tuning of parameters and behaviors (expensive and error-prone *trial and error*) is applying learning methods and simulation (cheap but effective *trial and error*). We would like the robot to optimize its driving behavior after every mechanical change. We would like the high-level control software to optimize the way the robot moves on the field also, and this can be best done by performing simulations which are then tested with the real robot. But first the simulator must learn how the real robot behaves, that is, it must synthesize a physical model out of observations. In this paper we tackle both problems: the first part deals with the "learning to drive" problem, whereas the second part deals with the "learning to simulate" issue.

1.1 Learning to drive

When autonomous mobile robots move, they compute a desired displacement on the floor and transmit this information to their motors. Pulse width modulation (PWM) is frequently used to control their rotational speed. The motor controller tries to bring the motor to speed — if the desired rotational velocity has not yet been reached, the controller provides a higher PWM signal. PID (proportional, integral, diferential) controllers are popular for this kind of applications because they are simple, yet effective. A PID controller can register the absolute difference between the desired and the real angular velocity of the motor (the error) and tries to make them equal (i.e. bring down the error to zero). However, PID control functions contain several parameters which can only be computed analytically when a perfect analytical model of the hardware is available. In practice, the parameters are set experimentally and are tuned by hand. This procedure frequently produces suboptimal parameter combinations.

In this paper we show how to eliminate manual adjustments. The robot is tracked using a global camera covering the field. The method does not require an analytical model of the hardware. It is specially useful when the hardware is modified on short notice (adding, for example, weight or by changing the size of the wheels, or its traction). We use reinforcement learning to find the best PID parameters. An initial parameter combination is modified stochastically — better results reinforce good combinations, bad performance imposes a penalty on the combination. Once started, the process requires no human intervention. Our technique finds parameters so that the robot meets the desired driving behavior faster and with less error. More precise driving translates in better general movement, robust positioning, and better predictability of the robot's future position.

1.2 Learning to simulate

Developing high-level behavior software for autonomous mobile robots (the playbook") is a time consuming activity. Whenever the software is modified, a test run is needed in order to verify whether the robot behaves in the expected way or not. The ideal situation of zero hardware failures during tests is the exception rather than the rule . For this reason, many RoboCup teams have written their own robot simulators, which are used to test new control modules in the computer before attempting a field test. A simulator saves hours of work, especially when trivial errors are detected early, or when subtle errors require many stop-and-go trials, as well as experimental reversibility.

The simulator of the robotic platform should simulate the behavior of the hardware as well as possible. It is necessary to simulate the delay in the communication and the robot's inertia; heavy robots do not move immediately when commanded to do so. The traction of the wheels, for example, can be different at various speeds of the robot, and all such details have to be taken into account. An additional problem is that when the robots are themselves being developed

and optimized, changes in the hardware imply a necessary change in the physical model of the robot for the simulation. Even if the robots does not change, the environment can change. A new carpet can provide better or worse traction and if the model is not modified, the simulation will fail to reflect accurately the new situation. In practice, most simulation systems settle for a simplistic "Newtonian" mass model, which does not really accurately corresponds to the real robots being used.

Our approach to solve this modelling problem is to learn the reaction of the robots to commands. We send driving commands to a mobile robot, that is the direction, the desired velocity and desired rotation. We observe and record the behavior of the robot when the commands are executed using a global video camera, that is, we record the instantaneous robot's orientation and position. With this data we train predictors which give us the future position and orientation of the robots in the next four frames, from our knowledge of the last six. The data includes all commands sent to the robots. The predictor is an approximation to the physical model of the robot, which covers many different situations, such as different speeds, different orientations during movement, and start and stop conditions. This learned physical model can then be used in our simulator providing the best possible approximation to the real thing, short of an exact physical model which can hardly be derived for a moving target, such as robot being developed and modified every day.

2 Related Work

We have been investigating reinforcement learning for wheeled robots for some time and also the issue of learning the physical behavior of a robot [4]. Recently we started applying our methods to PID controllers.

The PID controller has been in use for many decades, due to its simplicity and effectiveness [6]. The issue of finding a good method for adjusting the PID parameters has been investigated by many authors. A usual heuristic for obtaining initial values of the parameters is the Ziegler-Nichols method [16]. First a value for the P term is found, from which new heuristic values for the P, I, and D terms are derived. However, much additional tuning is still needed with this and other methods, and therefore engineers have recently concentrated on developing adaptive PID controllers which automatically adjust their parameters to a modified environment or plant load [2], [1]. Most of the published methods have been tested only with computer simulations, in which an analytical model of the control system is provided. When an analytical model is not available, stochastic optimization through genetic programming [11] or using genetic algorithms is an option. Our approach here is to use reinforcement learning, observing a real robot subjected to real-world constraints. This approach is of interest for industry, where often a PID controller has to tune itself adaptively and repetitively [15].

The 4-legged team of the University of Texas at Austin presented recently a technique for learning motion parameters for Sony Aibo robots [10]. The Sony

robots are legged, not wheeled, and therefore some simplification is necessary due to the many degrees of freedom. The Austin team limited the walking control problem to achieving maximum forward speed. Using policy gradient reinforcement learning they achieved the best known speed for a Sony Aibo robot [10]. We adapted the policy reinforcement learning method to omnidirectional robots by defining a quality function which takes into account speed and accuracy of driving into account. This makes the learning problem harder, because there can always be a compromise between accuracy and speed, but we succeeded in deriving adequate driving parameters for our robots.

With respect to simulations, the usual approach is to build as perfect a model of the robot and feed it to a simulation engine such as ODE (Open Dynamics Engine). This is difficult to do, and the simulated robot probably will not behave as the real robot, due to the many variables involved. In an influential paper, for example, Brooks and Mataric identify four robotic domains in which learning can be applied: learning parameters, learning about the world, learning behaviors, and learning to coordinate [5]. They do not mention learning what kind of robot you have, which in a sense, is our goal here. We are not aware, at the moment, of any other RoboCup team using learned physical behaviors of robots for simulations. We think that our approach saves time and produces better overall results that an ODE simulation.

3 The Control Problem

The small size league is the fastest physical robot league in the RoboCup competition, all velocities considered relative to the field size. Our robots for this league are controlled with a five stages loop: a) The video image from cameras overlooking the field is grabbed by the main computer; b) The vision module finds the robots and determines their orientation [13]; c) Behavior control computes the new commands for the robots; d) The commands are sent by the main computer using a wireless link; e) A Motorola HC-12 microcontroller on each robot receives the commands and directs the movement of the robot using PID controllers (see [7]). Feedback about the speed of the wheels is provided by the motors' impulse generators.

For driving the robots, we use three PID controllers: one for the forward direction, one for the sideward direction (in the coordinate system of the robot) and one for the angle of rotation. The required Euclidian and angular velocity is transformed in the desired rotational speed of three or four motors (we have three and four-wheeled robots). If the desired Euclidian and angular velocity has not yet been achieved, the controllers provide corrections which are then transformed into corrections for the motors.

3.1 Micro-controller

The control loop on the robot's micro-controller consists of six segments (see Fig. **??**). The robot receives from the off-the-field computer the target values

for the robot's velocity vector v_x, v_y and the rotational velocity ω, in its local coordinate system. The HC-12 micro-controller, which is constantly collection the current motor speed values by reading the motors' pulse generators, converts them into Euclidian magnitudes (see Section 3.2). The PID-Controller compares the current movement with the target movement and generates new control values (Section 3.3). These are converted back to motor values, which are encoded in PWM signals sent to the motors(Section 3.2).

3.2 From Euclidian Space to Wheel Parameters Space and vice versa

The conversion of the robot velocity vector (v_x, v_y, ω) to motor velocity values w_i of n motors takes place according to Eq. (1).

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \frac{1}{r} \begin{pmatrix} x_1 & y_1 & b \\ x_2 & y_2 & b \\ \vdots & \vdots & \vdots \\ x_n & y_n & b \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega_n \end{pmatrix}. \tag{1}$$

The variable r is the diameter of the omnidirectional wheels, b is the distance from the rotational center of the robot to the wheels, and $F_i = (x_i, y_i)$ is the force vector for wheel i. The special case of three wheels at an angle of 120^o can be calculated easily.

For the opposite direction, from motor velocities to Euclidian velocities, the calculation follows from Eq. (1) by building the pseudo-inverse of the transformation matrix:

$$\begin{pmatrix} v_x \\ v_y \\ \omega_n \end{pmatrix} = \frac{1}{r} \begin{pmatrix} x_1 & y_1 & b \\ x_2 & y_2 & b \\ \vdots & \vdots & \vdots \\ x_n & y_n & b \end{pmatrix}^{+} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}. \tag{2}$$

We map the values of n motors to the three dimensional motion vector. If the number of wheels is greater than three, the transformation is overdetermined, giving us the nice property of compensating the pulse counter error of the wheels (by a kind of averaging).

3.3 PID Controller

As explained above, we have programmed three PID controllers, one for the forward (v_x) and one for the sideward velocity (v_y), as well as one for the desired angular velocity (ω). Let us call $e_x(t)$ the difference between the required and the actual velocity v_x at time t. Our PID controller computes a correction term given by

$$\Delta v_x(t) = o_x + Pe_x(t) + I(\sum_{k=0}^{\ell} e_x(t-k)) + D(e_x(t) - e_x(t-1)) \tag{3}$$

There are several constants here: o_x is an offset, P, I, and D are the proportionality, integration, and difference constants, respectively. The correction is proportional to the error (modulated by P). If the accumulated error is high, as given by the sum of past errors, the correction grows, modulated by the integral constant I. If the error is changing too fast, as given by the difference of the last two errors, the correction is also affected, modulated by the constant D. A controller without I and D terms, tends to oscillate, around the desired value. A controller with too high an I value does not oscillate, but is slow in reaching the desired value. A controller without D term can overshoot, making convergence to the desired value last longer.

The error value used in the above formula is multiplied by a scaling constant before plugging its value in the formula. This extra parameter must also be learned. It depends on the geometry of the robot.

3.4 Learning the PID parameters

We solve the parameter optimization problem using a policy gradient reinforcement learning method as described in [10], [14]. The main idea is based on the assumption that the PID parameters can be varied independently, although they are correlated. Thus, we can modify the parameter set randomly, calculate the partial error derivative for each parameter, and correct the values. Note that, in order to save time, we vary the whole parameter set in each step and not each parameter separately.

The parameter set $\mathcal{P}$ consists of 18 elements $(p_1, \ldots, p_{18})$. The number of parameters is independent from the number of wheels, because we use one PID controller for each degree of freedom and not for each wheel. The standard hand-optimized parameters are used as the starting set. In each step, we generate a whole new suit of n parameter sets

$$\begin{aligned} \mathcal{P}^1 &= (p_1 + \pi_1^1, \ldots, p_{18} + \pi_{18}^1) \\ \mathcal{P}^2 &= (p_1 + \pi_1^2, \ldots, p_{18} + \pi_{18}^2) \\ &\vdots \\ \mathcal{P}^n &= (p_1 + \pi_1^n, \ldots, p_{18} + \pi_{18}^n). \end{aligned} \tag{4}$$

Whereby the value π_i^j is picked with uniform probability from the set $\{-\epsilon_i, 0, +\epsilon_i\}$ and ϵ_i is a small constant, one for each p_i.

The evaluation of one parameter set consists of a simple test. The robot has to speed up from rest into one particular direction, at an angle of 45^o realtive to its orientation. It has to drive for some constant time, without rotations and as far as possible from the starting point. Then the robot has to stop abruptly, also without rotating and as fast as possible. During this test phase, the robot doesn't get any feedback information from the off-the-field computer.

Each test run is evaluated according to the evaluation function $\mathcal{Q}(\mathcal{P}^j)$, which is a function of following criteria: the deviation of the robot to the predetermined direction, the accumulated rotation of the robot, the distance of the run, and

the distance needed for stopping. The only positive criterion is the length of the run; all other are negative.

We evaluate the function $\mathcal{Q}(\mathcal{P}^j)$ for all test parameter sets $\mathcal{P}^j$, where $j = 1, \ldots, n$. The sets are collected according to the π constants for every parameter into three classes:

$$\begin{aligned} \mathcal{C}_i^+ &= \{\mathcal{P}^j | \pi_i^j = +\epsilon_i, j = 1, \ldots, n\}, \\ \mathcal{C}_i^- &= \{\mathcal{P}^j | \pi_i^j = -\epsilon_i, j = 1, \ldots, n\}, \\ \mathcal{C}_i^0 &= \{\mathcal{P}^j | \pi_i^j = 0, j = 1, \ldots, n\} \end{aligned} \tag{5}$$

For every class of sets, the average quality computed:

$$\mathcal{A}_i^+ = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^+} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^+\|}, \mathcal{A}_i^- = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^-} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^-\|}, \mathcal{A}_i^0 = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^0} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^0\|} \tag{6}$$

This calculation provides us a gradient for each parameter, which shows us, whether some specific variance π makes the results better or not. If this gradient is unambiguous, we compute the new parameter value according to Eq. 7:

$$p_i' = \begin{cases} p_i + \eta_i & \text{if } \mathcal{A}_i^+ > \mathcal{A}_i^0 > \mathcal{A}_i^- \text{ or } \mathcal{A}_i^+ > \mathcal{A}_i^- > \mathcal{A}_i^0 \\ p_i - \eta_i & \text{if } \mathcal{A}_i^- > \mathcal{A}_i^0 > \mathcal{A}_i^+ \text{ or } \mathcal{A}_i^- > \mathcal{A}_i^+ > \mathcal{A}_i^0 \\ p_i & \text{otherwise} \end{cases} \tag{7}$$

Where the learning constant for each parameter is η_i. The old parameter set is replaced by the new one, and the process is iterated until no further progress is detected.

3.5 Results

Bild vorher, nachher. Auf den ersten blick sind die erzielten erfolge erstaunlich. schon nach kurzer zeit fdhrt der robiter auf der gedachten ideallinie. um den objektiven erfolg auch bei normalem spiel zu messen, wurde die qualitdt der vorhersage mit den handoptimierten parametereinstellung verglichen.

von 0 starten geht auch.

drehen wird auch gelernt, obwohl es nicht die aufgabe war. wahrscheinlich deshalb gelernt, weit dadurch drehfehler ausgeglichen werden. wir koennen davon ausgehen, dass dies vorhersage ein gutes messkriterium fuer die kontrolle des roboters ist.

4 Learning the behavior of the robot

We reported in a previous paper how we predict the position of our small-size robots in order to cope with the immanent system delay of the vision and control system [4]. When tracking mobile robots, the image delivered by the video camera is an image of the past. Before sending the new commands to the robot we have to take into account when will the robot receive them, because it

takes some time to send and receive commands. This means that not even the current real position of the robots is enough: we need to know the future position and future orientation of the robots. The temporal gap between the last frame we receive and the time our robots will receive new commands is the *system delay*. It can be longer or shorter, but is always present and must be handled when driving robots at high speed (up to 2 m/s in the small size league). Our system delay is around 100 ms, which corresponds to about 3 to 4 frames of a video camera running at 30 fps.

The task for our control system is therefore, from the knowledge of the last six frames we have received, and from the knowledge of the control commands we sent in each of those six frames, to predict the future orientation and position of the robots, four frames ahead from the past.

The information available for this prediction is preprocessed: since the reaction of the robot does not depend on its coordinates (for a homogenoeus floor) we encode the data in the robot's local coordinate system. We use six vectors for position, the difference vectors between the last frame which has arrived and the other frames in the past, give as (x, y) coordinates. The orientation data consist of the difference between the last registered and the previous orientations. Each angle θ is encoded as a pair $(\sin\theta, \cos\theta)$ to avoid a discontinuity when the angle crosses from a little less than 2π to a little more than 0. The driving direction and velocity transmitted as commands to the robot are given as one vector with (x, y)-coordinates, normalized by the velocity. They are given in the robots coordinate system. We use seven float values per frame, for six frames, so that we have 42 numbers to make the prediction.

We train actually two predictors: one for the position of the robot and another for the orientation. We have used neural networks and linear regression models, in both cases with good results.

Figure 1 shows the result of training the linear model and predicting from one to four frames after the last captured frame. The thin lines extending from the path are the result of predicting the next four frames at each point. The orientation of the robot is shown with a small line segment, and the desired velocity vector for the robot is shown with a larger segment. At sharp curves, the desired velocity vector is almost perpendicular to the trajectory. As can be seen, the predictions are very good for the trained model.

4.1 The simulator

Once we have trained a neural network to simulate the physical response of the robot to past states and commands, we can plug-in this neural network in our behavior simulation. We play with virtual robots: they have an initial position and their movement after receiving commands is dictated by the prediction of the behavior of the real robots in the next frame. We have here an interesting interplay between the learned physical behavior and the commands. In each frame we use the trained predictors to "move" the robots one more frame . This information, however, is not given to the behavior software. The behavior software sends commands to the virtual robots assuming that they will receive

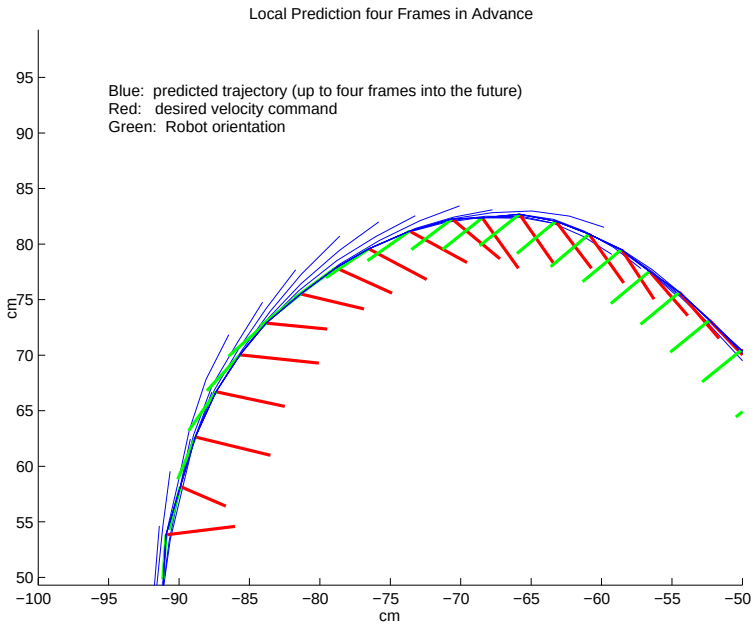


Fig. 1. A trajectory showing the predictions for four frames (thin lines) after each data point. The orientation of the robot is shown in green, and the desired velocity (the command sent) in red.

them with a delay (and the simulator enforces this delay). The behavior software can only ask the neural network for a prediction of the position of the robot in the fourth frame (as we do in real games). But the difference between what the high-level behavior control "knows" (the past, with four frames delay) and how the simulator moves the robot (a prediction, only one frame in advance) helps us to reproduce the effect of delays. Our simulator reproduces playing conditions as nearly as possible.

Fig. 2 shows a screenshot of the simulator running.The right field show the virtual robots playing. The left image usually shows the real robots, but during a simulation is a clone of the right window. The lines in the lower part show the activation values of the different robot behaviors. Using this information, we can adjust and modify the control software to test whole new strategies or simple changes.

5 Conclusions and Future Work

Our results show that it is possible to apply learning methods in order to optimize the driving behavior of a wheeled robot. They also show that learning can even be used to learn the physical reaction of the robot to external commands.

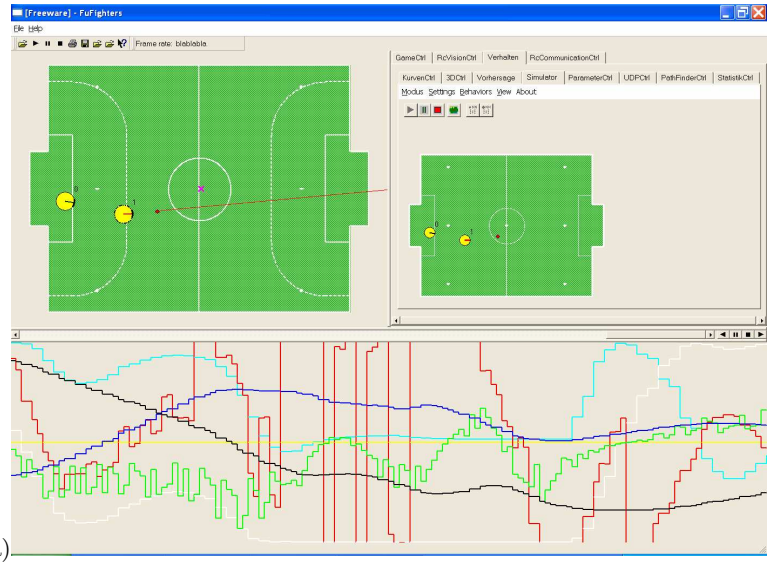
(a) 

Fig. 2. The screenshot shows the simulator running. The curves represent the changing activaton of behavior modes. From this information the "behavior engineer" gains valuable insights.

Optimizing the driving behavior means that we need to weight the options available. It is possible to let robots move faster, but they will collide more frequently due to lack of precision. If they drive more precisely, they will tend to slow down. Ideally, in future work, we would like to derive different PID controllers, for different scenarios. The high-level behavior could decide which one to apply, the more aggressive or the more precise. Another piece of future work would be trying to optimize the PID controller using high-level quality functions related to overall playing behavior.

We have shown in this paper also how to bootstrap a simulation of autonomous mobile robots when no physical model of the robots driving behavior is available. A planned upgrade for our simulator is to make it possible to replay stored games and ask "what-if" questions, by stopping the recording and letting the simulator run from the current game situation, with other strategy parameters entered by hand, in order to modify and optimize the control software. It would be also possible to make this an automatic process, in which a learning module replays game situations again and again in order to tune strategy parameters.

This paper is part of our ongoing work about making robots easier to adapt to an unknown and variable environment. We report elsewhere our results about automatic calibration of our computer vision system.

References

1. Aeström, Karl J., and Hgglund, Tore, Hang, C., and Ho, W., "Automatic tuning and adaptation for PID controllers—A survey," in L. Dugard, M. M'Saad, and I. D. Landau (Eds.), *Adaptive Systems in Control and Signal Processing*, Pergamon Press, Oxford, 1992, pp. 371–376.
2. Aeström, Karl J., and Hgglund, Tore, *PID Controllers: Theory, Design, and Tuning*, Second Edition, Research Triangle Park, NC, Instrument Society of America, 1995.
3. Sven Behnke, Bernhard Frötschl, Raúl Rojas, Peter Ackers, Wolf Lindstrot, Manuel de Melo, Andreas Schebesch, Mark Simon, Martin Spengel, Oliver Tenchio, "Using Hierarchical Dynamical Systems to Control Reactive Behavior", *RoboCup-1999: Robot Soccer World Cup III* Lecture Notes in Artificial Intelligence 1856, Springer-Verlag, 2000, pp. 186–195.
4. Sven Behnke, Anna Egorova, Alexander Gloye, Raúl Rojas, and Mark Simon," Predicting away the Delay", in D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), *RoboCup-2003: Robot Soccer World Cup VII*, Springer-Verlag, 2004, in print.
5. Brooks, Rodney A. and Mataric, Maja J., "Real robots, real learning problems, in Jonathan H. Connell and Sridhar Mahadevan (Eds.), *Robot Learning*, Kluwer Academic Publishers, 1993.
6. Callender, Albert, and Stevenson, Allan Brown, "Automatic Control of Variable Physical Characteristics U.S. patent 2,175,985. Issued October 10, 1939 in the United States.
7. Anna Egorova, Alexander Gloye, Achim Liers, Raúl Rojas, Michael Schreiber, Mark Simon, Oliver Tenchio, and Fabian Wiesel, "FU-Fighters 2003 (Global Vision)", in D. Polani, B. Browning, A. Bonarini, K. Yoshida (Eds.), *RoboCup-2003: Robot Soccer World Cup VII*, Springer-Verlag, 2004, in print.
8. Janusz, Barbara, and Riedmiller, Martin, "Self-Learning neural control of a mobile robot", *Proceedings of the IEEE ICNN'95*, Perth, Australia, 1995.
9. Alexander Kleiner, Markus Dietl, Bernhard Nebel, "Towards a Life-Long Learning Soccer Agent", in D. Polani, G. Kaminka, P. Lima, R. Rojas (Eds.), *RoboCup-2002: Robot Soccer World Cup VI*, Springer-Verlag, Lecture Notes in Computer Science 2752, pp. 119-127.
10. Kohl, Nate, and Stone, Peter, "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion," Department of Computer Science, The University of Texas at Austin, November 2003, paper under review.
11. Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.*, Kluwer Academic Publishers, 2003.
12. Martin Riedmiller and Ralf Schoknecht, "Einsatzmöglichkeiten selbständig lernender neuronaler Regler im Automobilbereich", *Proceedings of the VDI-GMA Aussprachetag*, Berlin, March 1998.
13. , Mark Simon, Sven Behnke, Raúl Rojas, "Robust Real Time Color Tracking. Lecture Notes in Artificial Intelligence", in P. Stone, T. Balch, G. Kraetzschmar (Eds.), *RoboCup 2000: Robot Soccer World Cup IV*, Lecture Notes in Computer Science 2019, Springer-Verlag, 2001, pp. 239–248.
14. Stone, Peter, Sutton, Richard, and Singh, Satinder, "Reinforcement Learning for 3 vs. 2 Keepaway", in P. Stone, T. Balch, G. Kraetzschmar (Eds.), *RoboCup 2000: Robot Soccer World Cup IV*, Lecture Notes in Computer Science 2019, Springer-Verlag, 2001, pp. 249-258.

15. K.K. Tan, Q.G. Wang, C.C. Hang and T. Hagglund, *Advances in PID Control*, Advances in Industrial Control Series, Springer Verlag, London, 1999.
16. Ziegler, J. G. and Nichols, N. B., "Optimum settings for automatic controllers, *Transactions of ASME*, Vol. 64, 1942, pp. 759-768.