Learning to Drive and Simulate Autonomous Mobile Robots

Alexander Gloye, Cüneyt Göktekin, Anna Egorova, 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.
We show that optimal control parameters for several PID controllers can be learned adaptively by driving an omni directional robot on a field while evaluating its behavior, using an reinforcement learning algorithm. After training, the robots can follow the desired path faster and more elegantly than with manually adjusted parameters.
Secondly, we show how to learn the physical behavior of a robot. Our system learns to predict the position of the robots in the future according to their reactions to sent commands. We use the learned behavior in the simulation of the robots instead of adjusting the physical simulation model whenever the mechanics of the robot changes. 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's mechanics is not available, so that analytical optimization or a perfect 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 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 velocity. 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 an adequate 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 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 accurately 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" constant-friction mass model, which does not correspond to the real robots being used.

Our approach to solve this modelling problem is to learn the reaction of the robots to commands. We transmit driving commands to a mobile robot: The desired 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 frames, from our knowledge of the last several ones [4]. The data includes also 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.

2 Related Work

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

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 [18]. First an initial value for the P term is found, from which new heuristic values for the P, I, and D terms are derived. Most of the published methods have been tested 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 [13] 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 [17].

The 4-legged team of the University of Texas at Austin presented recently a technique for learning motion parameters for Sony Aibo robots [12]. 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. 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. Another problem is that we learn to drive in all directions but not just forward. 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]. 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 than 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 [15]; 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 and one for the angle of rotation (all of them in the coordinate system of the robot). The required Euclidean 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 Microcontroller

The control loop on the robot's microcontroller consists of the following sequence of tasks: 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 microcontroller, which is constantly collecting 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 is computed by:

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

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.[1]

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. 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), one for the sideward velocity (v_y), and 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

$$\Delta v_x(t) = P e_x(t) + I\left(\sum_{k=0}^{\ell} e_x(t-k)\right) + D(e_x(t) - e_x(t-1)) \tag{2}$$

There are several constants here: P, I, and D are the proportionality, integration, and difference constants, respectively. The value of $\Delta v_x(t)$ is incremented (with respect to the leading sign) by an offset and then cut into the needed range. 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 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.

---

[1] `http://www-2.cs.cmu.edu/~reshko/PILOT/`

3.4 Learning the PID parameters

We solve the parameter optimization problem using a policy gradient reinforcement learning method as described in [12]. 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 $6 \times 3 = 18$ 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 suite of n parameter sets

$$\begin{aligned} \mathcal{P}^1 &= (p_1 + \pi_1^1, \ldots, p_{18} + \pi_{18}^1) \\ \mathcal{P}^2 &= (p_1 + \pi_1^2, \ldots, p_{18} + \pi_{18}^2) \\ &\vdots \\ \mathcal{P}^n &= (p_1 + \pi_1^n, \ldots, p_{18} + \pi_{18}^n). \end{aligned} \tag{3}$$

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 relative 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 (see Fig. 1(b)). During this test phase, the robot does not receive 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 weighted function of the 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:

$$\begin{aligned} \mathcal{C}_i^+ &= \{\mathcal{P}^j | \pi_i^j = +\epsilon_i, j = 1, \ldots, n\}, \\ \mathcal{C}_i^- &= \{\mathcal{P}^j | \pi_i^j = -\epsilon_i, j = 1, \ldots, n\}, \\ \mathcal{C}_i^0 &= \{\mathcal{P}^j | \pi_i^j = 0, j = 1, \ldots, n\} \end{aligned} \tag{4}$$

For every class of sets, the average quality is computed:

$$\mathcal{A}_i^+ = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^+} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^+\|}, \; \mathcal{A}_i^- = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^-} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^-\|}, \; \mathcal{A}_i^0 = \frac{\sum_{\mathcal{P} \in \mathcal{C}_i^0} \mathcal{Q}(\mathcal{P})^x}{\|\mathcal{C}_i^0\|} \tag{5}$$

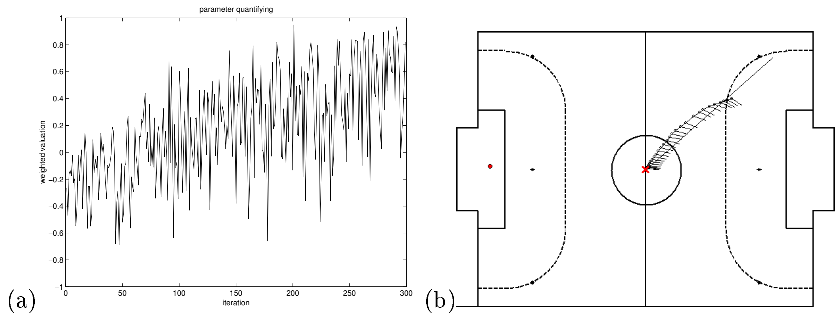

Fig. 1. (a) The quality of the tested parameter sets. The graph shows only the evaluated parameter sets which are varying up to ϵ_i from the learned parameter i. Therefore, the values are noisy. (b) Example of a test run. The straight line shows the desired direction and the dotted line shows the real trajectory of the robot on the field.

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. 6:

$$p_i' = \begin{cases} p_i + \eta_i & \text{if } \mathcal{A}_i^+ > \mathcal{A}_i^0 > \mathcal{A}_i^- \text{ or } \mathcal{A}_i^+ > \mathcal{A}_i^- > \mathcal{A}_i^0 \\ p_i - \eta_i & \text{if } \mathcal{A}_i^- > \mathcal{A}_i^0 > \mathcal{A}_i^+ \text{ or } \mathcal{A}_i^- > \mathcal{A}_i^+ > \mathcal{A}_i^0 \\ p_i & \text{otherwise} \end{cases} \tag{6}$$

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

We made two experiments: In the first, we learned only the P, I, and D values. These values were initialized to 0. The evolution of the learning process is shown in Fig. 1(a) and in Fig. 2. Fig. 1(a) shows the quality of the test runs during the whole learning process and Fig. 1(b) a sample run of the robot. The total quality of a test run can be broken down into the four quality magnitudes. These are shown in Fig. 2. At the beginning, the robot does not move. After 10 iterations of parameter evaluation and adaptation, the robot moves at an average of one meter per second, with acceleration and braking. The average deviation from the desired direction is 10 percent at the end point.

In a second experiment, the PID controller was initialized with the hand-optimized values. The result is shown in Fig. 3. The parameters are not independent, which leads to some oscillations at the beginning.

In the experiment shown in Fig. 3 the rotation controller is also learned, although the task of the robot is only to drive in one direction without rotating. However, rotation control is important for straight movement, because an error

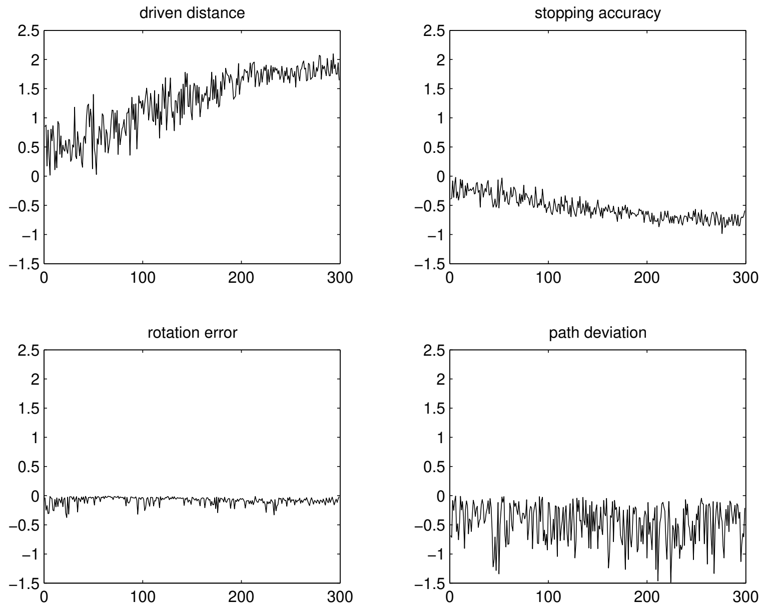

Fig. 2. The four componets of the quality function. The total quality is a weighted average of these four magnitudes.

caused by wheel slippage at the beginning of the movement — in the acceleration phase — can be compensated later.

The PID parameters learned with our experiments are used now for control of our robots. They can be relearned, if the robot is modified, in a few minutes.

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 unavoidable 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 the robot will 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.

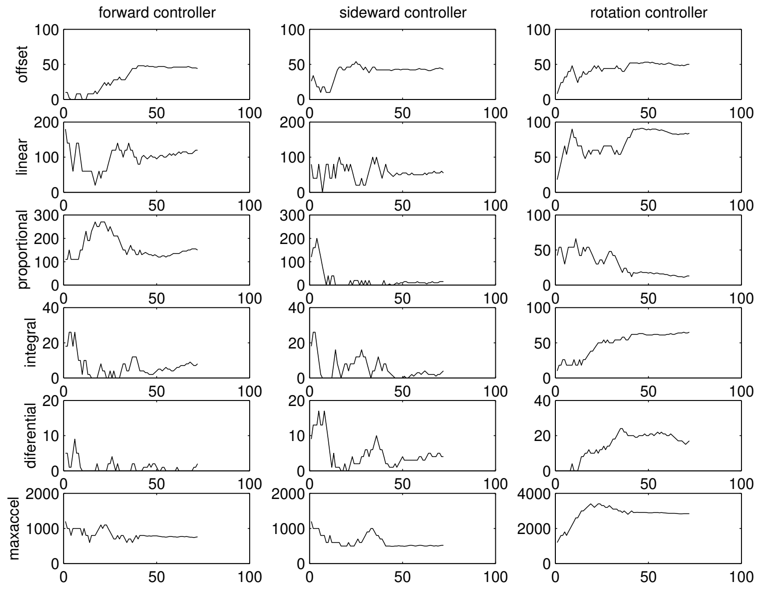

Fig. 3. The learned parameters, started from the hand-optimized set. The parameters are not independent.

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 homogeneous floor) we encode the data in the robot's local coordinate system. Obstacles and walls must be handled seperately. We use six vectors for position, the difference vectors between the last frame which has arrived and the other frames in the past, given as (x, y) coordinates. The orientation data consist of the difference between the last registered and the six previous orientations. Each angle θ is encoded as a pair $(\sin\theta, \cos\theta)$ to avoid discontinuity when the angle crosses from 2π to 0. The desired driving direction, velocity and rotation angle transmitted as commands to the robot of the last six frames are given as one vector with (v_x, v_y, ω)-coordinates. 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 don't use the current frame directly, but indirectly, because we actually use the differences between each last frame and the current frame. The current motor values do not influence the robot motion in the next four frames (because of the delay), so they are irrelevant.

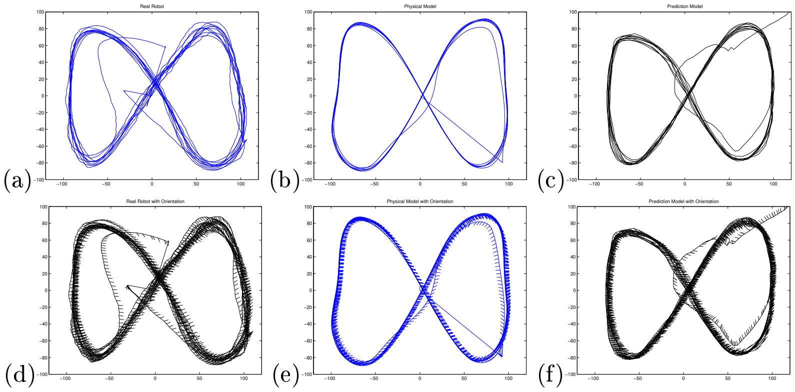

Fig. 4. A comparison of driving paths: (a) real path driven by a real robot, (b) simulation of the same path with a physical model and (c) simulation with the prediction model. (d),(e),(f) show the same paths including the orientation of the robot, drawn as small segments.

We use neural networks and linear regression models to pretict the future positions and orientations of the robot, one or four frames in advance. For details see [8].

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 provided to the behavior software. The behavior software sends commands to the virtual robots assuming that they will receive 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. 4 shows a comparison of the driving paths obtained with different models and reality.

As can be seen in Fig. 4, the "Newtonian" model is too smooth. The real driving behavior is more irregular, because when the robot drives it overshoots or slips on the floor. Our simple "Newtonian" model could be extended by a more realistic one, which includes for examle nonlinear friction terms. The learned behavior (c) reflects more accurately this more problematic and realistic driving behavior, so we don't need any model. This is important for high-level control, because the higher control structures must also learn to cope with unexpected driving noise.

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 model the physical reaction of the robot to external commands.

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 intelligent controllers, specialized in different problems. 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 high-level behaviors using reinforcement learning, so that they compensate disadvantages of the robot on-board control.

We have seen in Fig. 4, that the predicted robot motion (c) is more accurate than the real one (a), because the prediction (any prediction) smooths the observed real movements. The driving noise could be also analyzed and its inclusion in the simulator would make the model even more realistic.

This paper is part of our ongoing work on making robots easier to adapt to an unknown and variable environment. We report elsewhere our results about automatic color and distortion calibration of our computer vision system [9]. The interplay of vision and control software will make posible in the future to build a robot, put it immediately on the field, and observe how it gradually learns to drive.

References

1. Karl J. Aeström, Tore Hägglund, C. Hang, and W. Ho, "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. Karl J. Aeström, and Tore Hägglund, *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. Rodney A. Brooks, and Maja J. Mataric, “Real robots, real learning problems, in Jonathan H. Connell and Sridhar Mahadevan (Eds.), *Robot Learning*, Kluwer Academic Publishers, 1993.
6. Albert Callender, and Allan Brown Stevenson, “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. Alexander Gloye, Mark Simon, Anna Egorova, Fabian Wiesel, Oliver Tenchio, Michael Schreiber, Sven Behnke, and Raùl Rojas, “Predicting away robot control latency”, Technical Report B08-03, Free University Berlin, 2003.
9. Alexander Gloye, Mark Simon, Anna Egorova, Fabian Wiesel, and Raùl Rojas, “Plug & Play: Fast Automatic Geometry and Color Calibration for a Camera Tracking Mobile Robots”, Institut für Informatik, Freie Universität Berlin, February 2004, paper under review.
10. Barbara Janusz, and Martin Riedmiller, “Self-Learning neural control of a mobile robot”, *Proceedings of the IEEE ICNN'95*, Perth, Australia, 1995.
11. 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.
12. Nate Kohl, and Peter Stone, “Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion,” Department of Computer Science, The University of Texas at Austin, November 2003, paper under review.
13. John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.
14. Martin Riedmiller and Ralf Schoknecht, “Einsatzmöglichkeiten selbständig lernender neuronaler Regler im Automobilbereich”, *Proceedings of the VDI-GMA Aussprachetag*, Berlin, March 1998.
15. Mark Simon, Sven Behnke, Raúl Rojas, “Robust Real Time Color Tracking”, 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.
16. Peter Stone, Richard Sutton, and Satinder Singh, “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.
17. K.K. Tan, Q.G. Wang, C.C. Hang and T. Hägglund, *Advances in PID Control*, Advances in Industrial Control Series, Springer Verlag, London, 1999.
18. J. G. Ziegler, and N. B. Nichols, “Optimum settings for automatic controllers, *Transactions of ASME*, Vol. 64, 1942, pp. 759-768.