***Biologically inspired robot behavior engineering*, pp. 173 - 197**
Physica-Verlag GmbH, Heidelberg, Germany, 2003

Self-Adapting Neural Networks for Mobile Robots

Ralf Salomon

Department of Electrical Engineering and Information Technology, Institute of Applied Microelectronics and Computer Science
University of Rostock, 18051 Rostock, Germany
`ralf.salomon@etechnik.uni-rostock.de`

Abstract. In the context of research on intelligence, autonomous agents and in particular mobile robots are to behave on their own without any human control. Unfortunately, the real world exhibits plenty of noise, uncertainties, sudden changes, etc, which all imposes significant challenges on the design of appropriate control architectures. This chapter starts off with an existing controller, known as the distributed adaptive control architecture and shows how significant improvements can be achieved by incorporating biological mechanisms, such as proprioception. The resulting controller requires much less preprogrammed design knowledge, exhibits more flexible adaptation capabilities, and is more fault tolerant with respect to environmental changes and sensor failures as its predecessors.

1 Introduction

For many years, artifi cial intelligence (AI) has investigated the principles and mechanisms of intelligence. By so doing, (classical) AI has focused on “formalizable” intelligence, such as expert systems and chess programs, with the latter chalanging the intellectual abilities of world-class players. Despite these and other successes, many problems remain unsolved in classical AI. Since most approaches *naturally* involve a human intermediary to interface between the program and its environment, (software) agents that directly interact with their real-world world have been widely neglected. In order to provide some new alternatives to classical AI, Rodney Brooks [4, 5] has laid down a new research direction that has been named nouvelle or new AI. By focusing on agent-environment interactions, however, researchers had to learn to think in different ways, since several problems naturally disappear whereas other arise “out of the blue”. In order to better appreciate this approach, section 2 briefly describes the autonomous agents perspective.

One goal of research on autonomous agents is to build robots that are able to behave appropriately without human control in dynamically changing, partially unknown environments. Research on evolutionary robotics [7, 9, 10, 12, 21, 28], for example, employs different evolutionary algorithms [1], such as genetic algorithms [11] or evolution strategies [26, 29], to evolve complete control architectures. These systems, however, are mostly limited to adaptation on an evolutionary time scale, since most of the evolved controllers do not feature any learning or self-organizing process. Exceptions are controllers presented by Floreano and Mondada [10] as well as Nolfi and Parisi [21], which feature learning to a limited extend. The interested reader may find several chapters on the adaptation on an evolutionary time scale throughout this book.

In many mobile robots, the control structure is implemented by neural networks. For several reasons, however, network models, such as backpropagation training or Kohonen feature maps [15], cannot be used in autonomous systems (see also [27]). These training algorithms require a separation into a training and a performance phase, and such a separation seems inappropriate for autonomous agents, since an autonomous agent is exposed to a continuous, unending stream of sensory patterns. Not only the composition of these patterns but also the relevance of each pattern might change over time. Even if it were possible to train a neural network controller prior to a performance phase, the resulting controller would be limited to the training environment; a trained network would not be able to adapt to unconsidered dynamical changes.

For the goal of building systems that exhibit robust behavior in dynamically changing environments, the employment of self-organizing processes seems one of the most promising options. The distributed adaptive control (DAC) architecture [25, 31] is a good example of such efforts. DAC and other approaches [16], feature self-organization and learning in only some minor components; learning is mostly limited to some visual components, and the robot more or less operates with an open-loop control architecture, i.e., internal feedback is not utilized. From an engineering point of view, those forms of learning would be called self-calibration. Section 3 reviews the DAC architecture and discusses some real-world experiments that indicate several limitations and options for improvement, especially for mobile robots that behave in the real world rather than in simulation. These limitations are intrinsic to these approaches, since the designer induces too much “world modeling” for particular environments with at least some constant properties. In other words, during the development, the designer decides which visual patterns have which meaning, and therefore, limit the controller to specific, static environments.

Surprisingly little attention has been devoted to feeding back proprioceptive activity patterns. Proprioceptive patterns refer to stimuli arising within an organism. Proprioception is omnipresent in living creatures [13, pp. 324ff]. For instance, if an animal moves a ligament, muscle spindles feed information about the muscular tension and the ligament/joint position back to the brain via the spinal cord. In case of exceeding a certain threshold, the spinal cord triggers reflexes in order to prevent damaging the joints. Section 4 proposes a new self-organizing control architecture, called self-organization-through-proprioception (STP) architecture, that is inspired by these biological observations, and can be seen as an extension of the DAC architecture. The STP architecture uses proprioceptive data in order to better evaluate chosen actions with respect to visual data mediated by the environment, i.e., the controller uses different modalities. Section 5 summarizes the test environments as well as all parameter settings, and Section 6 presents some results obtained with the proposed control architecture. Finally, Section 7 concludes with a detailed discussion.

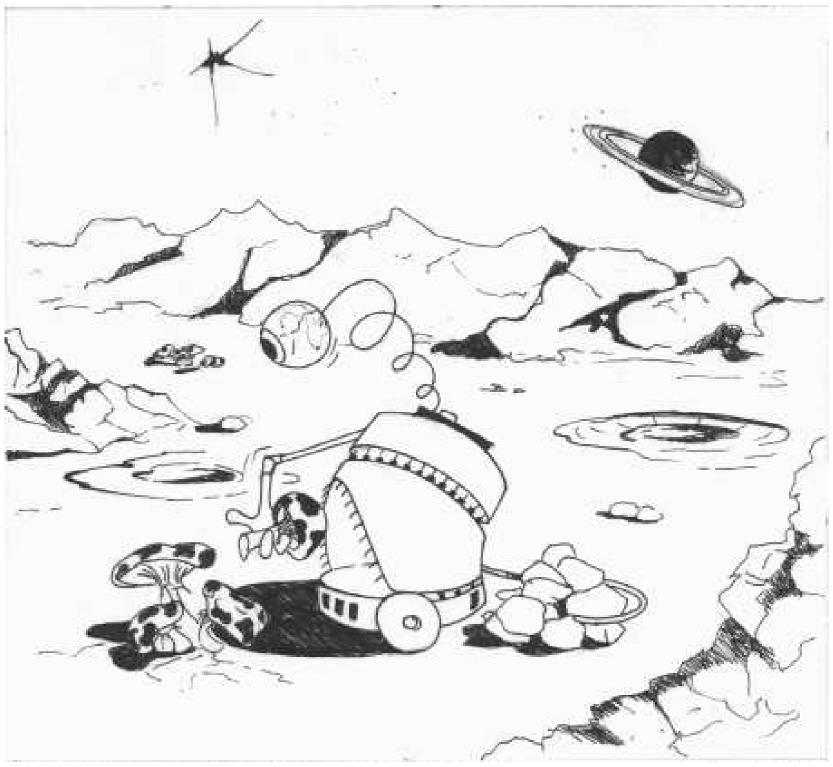

Fig. 1. The Autonomous Agent Perspective

2 Motivation: The Autonomous Agent Perspective

Autonomous agents can be used not only for research on intelligence but more generally as a framework for research on soft computing. This section first gives an overview of autonomous agents, it then discusses their particular perspective, and finally, it explains why autonomous agents provide a valuable framework for research on soft computing.

Figure 1 illustrates a typical example of an autonomous agent, which is a physical robot autonomously operating (i.e., without any human control) in a remote environment. In this example, the robot is equipped with two wheels, two grippers, one for food and one for ore, an eye (to symbolize a camera), and several other (infrared) sensors. Inside its body (not shown), the robot hosts two motors, an energy source, a control architecture that links the sensors with the actuators, and possibly other components. It is important to note that autonomous agents are to interact with their environments *on their own*, that they have to extract all information from their environment by themselves, and that no human intermediary is involved in the perception-action cycle, which also excludes any remote control or any other human supervision. The ultimate goal of research on autonomous agents is to understand intelligence by building autonomous robots [24].

The particular view on the study of intelligence pursued by research on autonomous agents has been laid down by Brooks [4, 5]. The main assumption of nouvelle or new AI [4, 5, 24] is that intelligence cannot emerge from an abstract computational level on which arbitrary physical symbols are combined (see also the Physical Symbol System Hypothesis [20]); the symbol processing system cannot assign a suitable meaning to the symbols, since the symbols are given by the outside world, e.g., the designer. Brooks [3, p. 5] argues:

> "Nouvelle AI is based on the physical grounding hypothesis. This hypothesis states that to build a system that is intelligent it is necessary to have its representations grounded in the physical world. Our experience with this approach is that once this commitment is made, the need for traditional symbolic representations soon fades entirely. The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough."

For the goal of understanding intelligence as a result of an agent-environment interaction, several design principles of autonomous agents have been developed [22, 24]. These design principles concern, among other things, the agent's body, its control architecture, the sensory-motor coordination, and the demand of being a "complete" agent, and therefore, they impose several constraints on how to build such agents. For the purpose of this chapter, the following two design principles are of major interest. According to the complete-agent principle (the first design principle) [22], an autonomous agent must be physically realized (embodied), be able to sustain itself over an extended period of time (self-sufficient), interact with its environment without any human control (autonomous), and must be controlled from its own perspective (situated). The loosely-coupled-processes principle (the third design principle) [22] argues that the control architecture should consist of several, parallel operating, loosely coupled processes, rather than consisting of a central decision unit, since no *homunculus* has yet been found in natural brains; a significant amount of evidence [3, 17, 19] indicates that a particular complex behavior emerges from the cooperation of a set of simple behaviors. A thorough discussion of these design principles as well as a collection of case studies can be found in Pfeifer [22, 23] and Pfeifer and Scheier [24], and detailed discussions on several other case studies are presented in Brooks [3–5], Lambrinos [16], Lambrinos and Scheier (1996), and Pfeifer and Scheier [23].

Before proceeding, a few remarks should be given on software programs versus embodied agents, for which the term "mobile robots" would be more appropriate. Even though software simulations are considered extremely helpful for designing agents [22], it seems that due to the strong focus on embodiment, software programs are not considered autonomous agents. In the present context, two different types of software programs, simulations and software agents, should be distinguished. Throughout this chapter, simulation software refers to programs that run and investigate a simplified mobile robot in a simplified environment, and software agents refer to programs that "live," for instance, in cyberspace – the world inside computers and networks. These software agents can be considered autonomous agents as long as real-world aspects, such as uncertainty and dynamically changing environmental conditions, hold for these programs. An overview on research on software agents can be found in Etzioni [8], Maes [18],

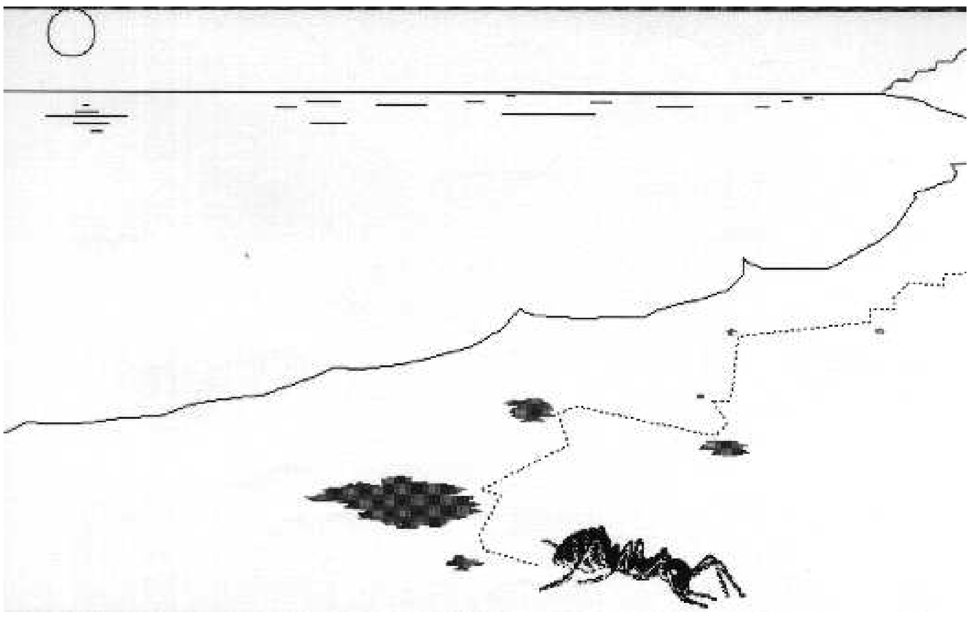

Fig. 2. Simon's ant [30].

and Wooldridge and Jennings [32]. Even though the similarity between the agents and animals/humans is considered of high importance for the study of natural intelligence [23], it is of less importance for the purpose of this chapter. Therefore, autonomous agents refer to mobile robots as well as software agents throughout this chapter, and both forms are distinguished only if required by the context.

For research in the context of autonomous agents, the frame-of-reference problem [6] is of special importance. The frame-of-reference problem states that things might appear different from the agent's and observer's perspective. In other words, the designer must be careful not to mix up observable behaviors with the agent's mechanisms causing these behaviors. The frame-of-reference problem can be nicely illustrated with Simon's ant presented in Fig. 2. Given the ant's path on the beach, an observer might come to the conclusion that the ant features a path planing module. However, this conclusion is not necessarily true. It might be that the ant just wanders around and avoids obstacles. Furthermore, even the last consideration does not give rise to the conclusion that the ant features an object-avoidance module; the object-avoidance behavior is a result of the observation, which takes place in the observer's mind, and yet, no object-avoidance modules have been found in ants. Therefore, no conclusion about the internal mechanisms should be directly drawn from observations.

3 Background

The background of this chapter is mainly given by the DAC architecture, which is described in detail in Subsection 3.1. It features self-organization and life-long learning without distinguishing between a learning and a performance phase. Due to its design, however, the architecture exhibits problems that are discussed in Subsection 3.2.

3.1 The DAC Architecture

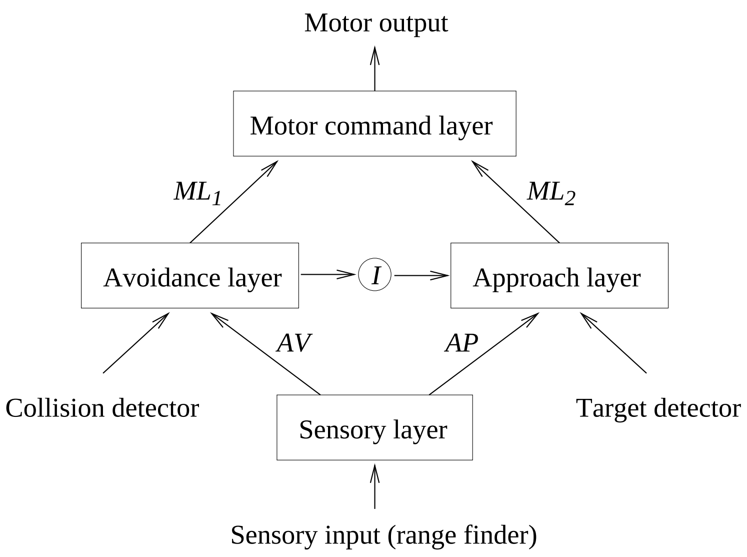

Fig. 3. In the distributed adaptive control (DAC) architecture, the avoidance units, which are triggered by a simple collision detector, trigger the motor command units via fixed ML_1 connections that are prewired by the designer. Learning takes place in the *AV* layer, which establishes existing correlations between the collision detector and the range finder. The approach layer is similarly embedded and can be inhibited by the avoidance layer. For further details, see text.

The distributed adaptive control (DAC) architecture has already been proposed in Pfeifer and Verschure [25] and Verschure *et al*. [31]. It consists of four unit layers and the four connection layers ML_1, ML_2, *AV*, and *AP* (Fig. 3). The motor layer consists of several units that execute certain reflexes, such as move forward, turn right, back up, etc. The command units are activated in a winner-take-all manner, where the move-forward-unit is activated by default. The robot's action space is completely defi ned by the motor command layer, since no other output unit is present in the architecture.

Each command unit can be triggered by a unit of the avoidance or approach layer. These two layers are connected to the motor units via hard-wired ML_1 and ML_2 connections. The approach layer receives its input data from a target detector (a nose), and tries to steer the robot toward a target. If, for example, the robot senses a food source on the right-hand-side, the approach layer triggers a turn-right reflex.

Units in the avoidance layer are triggered by a collision detector. If, for example, a collision occurs on the left-hand-side, the avoidance layer might trigger a command unit that performs a back-up-and-turn-right reflex. In addition, the avoidance layer inhibits the approach layer in case of a collision. This form of a dynamic suppression of another module has been proposed in Brooks [2].

The combination consisting of the the collision and target detectors, the avoidance and approach layers, as well as the motor command layer provides the basic control system. This subsystem is sufficient to move the robot around and find food while recognizing obstacles. The collision detector and the target detector together provide a *value system*, since these detectors provide feedback about the quality of the robot's behavior.

Both the avoidance layer and the approach layer are connected to a sensory layer via fully connected, modifiable connections located in the *AV* and *AP* layer, respectively. The sensory layer receives its input from a range finder, which can be implemented by infrared proximity or ambient light sensors. The activation of a unit in the avoidance or approach layer not only triggers a specific command unit but also triggers the execution of a learning step.

In mathematical terms, the system is defined as follows. The activation of an approach or avoidance unit s_i^λ is given by

$$s_i^\lambda = f\left(c_i^\lambda + \sum_{j=1}^{N} s_j K_{ij}^\lambda\right) , \tag{1}$$

where λ denotes either the approach or the avoidance layer, c_i^λ is the input from the corresponding target or collision detector, s_j is the activation of the range-finder unit j, K_{ij}^λ represent the weights of the *AV* and *AP* connections, and $f(\cdot)$ is a threshold function. In a learning step, the weights K_{ij}^λ are updated as follows

$$K_{ij}^\lambda \leftarrow K_{ij}^\lambda + \frac{1}{N}\left(\eta^\lambda s_i^\lambda s_j - \epsilon \bar{s}^\lambda K_{ij}^\lambda\right) , \tag{2}$$

where N is the number of the range-finder units, η^λ is the learning rate, ϵ is the decay rate, and $\bar{s}^\lambda$ is the average activation of layer λ. The learning rule (2) consists of a Hebbian term (η^λ) and an active forgetting term (ϵ).

With the preprogrammed part of the DAC architecture, the robot can move around. In case it experiences a collision or locates a food source, the corresponding detectors initiate the execution of an appropriate reflex and a learning step. Over time, the DAC architecture learns to associate range finder data with the collision and target detector. Once the connections K_{ij}^λ have sufficient strength, the range finder can trigger units in the avoidance layer initiating certain reflexes executed by the corresponding command units before any unit in the collision or approach layer is activated. An extensive analysis of many successful experiments done in simulation can be found in [25, 31].

3.2 Limitations of the DAC Architecture

Despite many successful DAC simulation experiments [25, 31], further simulations reveal, however, that a fair amount of the DAC architecture has to be determined by the

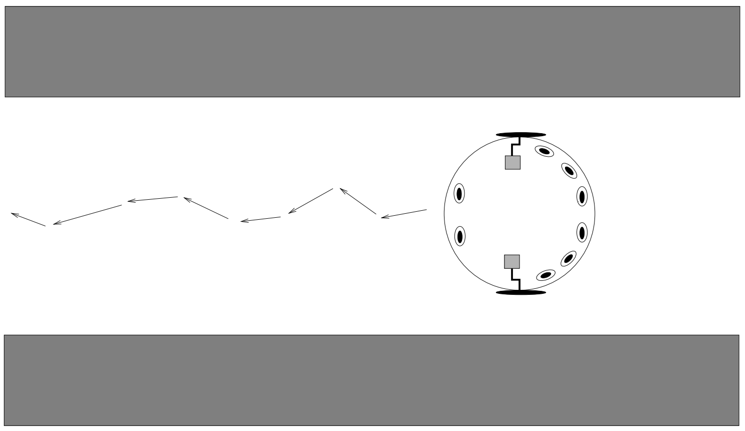

Fig. 4. In a narrow aisle, the DAC architecture constantly backs up a robot, since the corresponding motor command units are always triggered due to high infrared sensor readings.

designer. As is evident from equations (1) and (2), learning takes place only in layers *AV* and *AP*, and thus, self-organization is limited to these two layers. In addition, several parameters, such as ϵ, the two learning rates η^λ, and various thresholds for $f(\cdot)$ have to be specifi ed. In the simulation experiments presented in Verschure *et al.* [31], these parameters have been varied considerably, indicating that they have a signifi cant influence on the robot's behavior.

In addition to simulation, the DAC architecture has also been implemented[1] on the Khepera robot, which is explained in detail in the Appendix. As expected from simulation, the control architecture works adequately for environments with a low density of obstacles. As in the simulation [25, 31], Khepera does not have any collision sensor, and therefore, the collision sensors are derived from the infrared proximity sensors. After normalization, each proximity sensor provides values in the range $[0..1]$, and a value of 1 is used to indicate a collision. More formally, the collision detector is derived as follows:

$$c_i^\lambda = g(s_j, \theta_c) \; , \tag{3}$$

where $g(\cdot)$ denotes a threshold function with threshold θ_c. Since the implementation uses Khepera's six frontal sensors, the collision detector consists of six units, which are all hard-wired to appropriate command units.

As already explained, the collision sensors are derived from the infrared sensors (Khepera) or the range fi nder (simulation), respectively. Hence, both detectors are highly correlated, and actually represent only one modality. Therefore, learning is very easy. It has been reported [31] that the robot did not experience any further collision after 600 learning steps have been performed. For a collision-free behavior, it is suffi cient to set all K_{ii}^λ connections above function threshold, since both detectors are highly correlated.

[1] The DAC implementation for the Khepera robot can be obtained by sending an email to lambri@ifi.unizh.ch.

In other words, setting $K_{ii} > \theta_c$ in equation (3) leads to an avoidance behavior without activating any collision unit.

The high correlation between both detectors can also be observed in the *AV* layer after a sufficiently large number of learning steps. Each learning step predominantly increases the connections K^{λ}_{ii} that connect corresponding collision and infrared sensors. Due to real-world correlations, however, each learning step also reinforces connections to neighboring sensors. Essentially, when setting all parameters appropriately, the robot learns to move around without hitting any obstacle by establishing a matrix $\boldsymbol{K}$ with strong diagonal elements and some significant connections around the diagonal.

The collision detector is part of the value system, and thus plays an important role in the entire system. The problem is that the infrared sensors send very imprecise data. For a constant agent-object distance, the actual sensor reading depends on the ambient light, the material the sensed object is made of, the sensor's technical characteristic, and additional noise. Hence the value system is very sensitive with respect to these sensors.

In addition, the derivation of the collision sensors from the proximity sensors has the effect that the DAC architecture *assumes* a collision if a proximity sensor exceeds a certain threshold. The control architecture cannot *determine* whether or not a collision has occurred; the decision can be false in both ways, i.e., a collision and a non-collision can be mistakenly assumed.

By deriving the collision sensors from another sensory input device, the DAC architecture implements a significant amount of system-environment assumptions, which are not sufficiently grounded in the sensory-motor couplings; the environment *indirectly* feeds back the motor activities to the control architecture. It can happen that the control architecture assumes a collision even though the robot has sufficient distance to any obstacle, or it might happen that the range finder provides sensor readings below collision threshold, even though the robot is blocked by an obstacle. The first problem can be observed in the following experiment.

In Fig. 4, the robot is located in a small aisle. Due to its technical characteristics, the DAC architecture (constantly) receives sensor readings indicating a collision on either lateral side, even though the robot does not touch a wall. This effect does not occur in simulation, since the range finder provides precise data, and thus reliable information about collisions. Since the DAC architecture constantly assumes a collision in this narrow aisle, it constantly triggers certain reflexes, which constantly moves the robot backwards.

The reason for the robot behavior presented in Fig. 4 is that too many assumptions were used in the design process of the control architecture. If a robot is to exhibit robust behavior in a dynamically changing, partially unknown environment, the control architecture should not assume specific sensor values for particular situations. On the contrary, the control architecture must be self-adaptive and should tune as many parameters as possible by appropriate mechanisms. Furthermore, it can happen that a real-world sensor fails. A collision sensor, for example, can always send a value of 0 or 1. A robust control architecture should be capable of dynamically dealing with such a situation.

Besides the limitations discussed in this subsection, the DAC architecture offers a very valuable approach to self-organization. The self-organizing process utilizes a value system in order to correlate different sensors for choosing different actions. The following section uses some of the ideas introduced in the DAC architecture to propose a new self-organizing control architecture that can be seen as an extension of the DAC architecture.

4 The STP Architecture

This section describes the self-organization-through-proprioception (STP) architecture in full detail. It is helpful to fi rst describe the STP architecture on a functional level and to then consider implementation details.

As can be seen from Fig. 5, the STP architecture consists of a controller, a value system, and several sensors. The controller is inspired by the subsumption architecture [2] and consists of two modules, which are often called processes. Each module determines its individual contribution to the motor activation by using the activity of the infrared proximity sensors. The contributions are then combined and sent to the motors. The "turn" module can inhibit the activity of the "forward" module.

The value system uses *proprioceptive* data originating from the motor side, which is a different sensory modality than is used by the actual controller. At the motor side, speed difference *SD* sensors calculate the difference between the desired and the actual speed, which is measured by attached wheel encoders *WE*. Depending on the *SD*

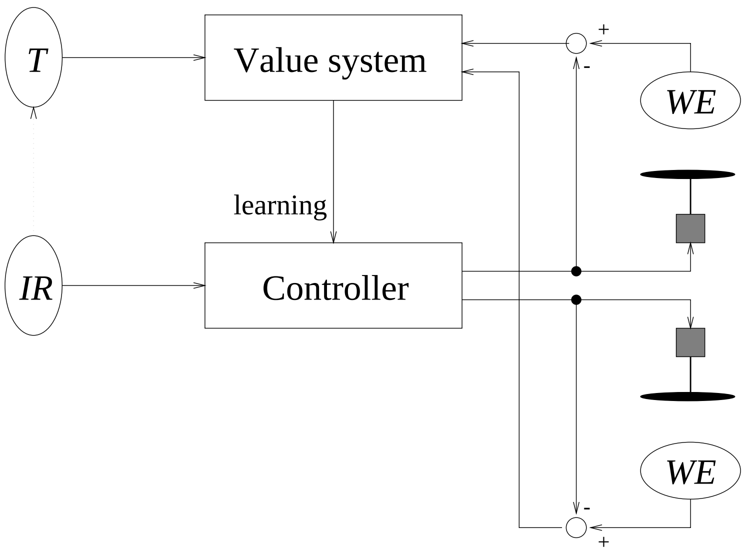

Fig. 5. The self-organization-through-proprioception (STP) architecture consists of a value system and a controller that features a turn and a forward moving module. Both modules use only the infrared sensors for determining the motor activations. The value system uses the difference (*SD*) between the motor activation and the attached wheel encoders (*WE*) (proprioception) to recognize a collision. For further details, see text.

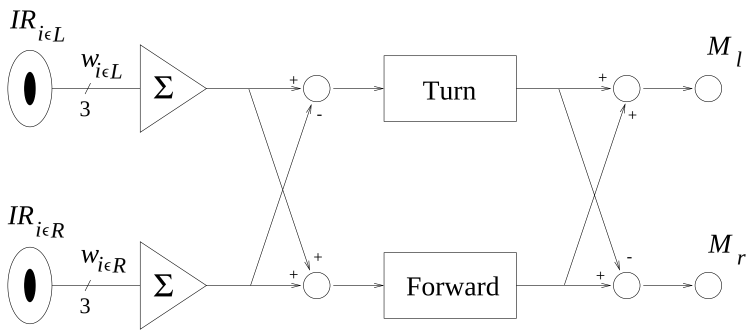

Fig. 6. The implementation of the STP controller module exploits the symmetric morphology of the robot. The turn module uses the difference between the activation on left-hand and right-hand side, whereas the forward module uses the sum of both activations. The sum and difference of both modules is fed to left and right motor, respectively.

sensor readings, the value system decides whether or not a collision has occurred. The idea is that in case of a collision, the surface-to-wheel friction slows down the wheels signifi cantly. The effect, however, depends on the surface, the wheels, and the electrical characteristics of the motors. When using Khepera on a wooden surface, for example, the wheels are mostly stopped by the environment, but sometimes, they are spinning. To deal with this problem, the value system uses an additional non-moving sensor *NM*, which is described below.

Let us now discuss the STP architecture on the implementation level, which is shown in Fig. 6. The implementation seems different from the functional description as presented in Fig. 5, since both modules use the same connections $\boldsymbol{w}$. The number of parameters is kept small so as to avoid tuning by too many degrees of freedom. As can be seen from Fig. 6, the weighted sum $IR_i w_i$ is separately calculated for the left-hand-side $L = \{1, 2, 3\}$ and the right-hand-side $R = \{4, 5, 6\}$. The sum of both values is fed into the forward unit, whereas the difference is used by the turn unit. Each unit determines its contribution, and the sum and difference are fed to the left and right motor, respectively. The activation of the forward unit s_f is calculated as

$$s_f = 1 - 1.2 \tanh\left(\sum_{i=1}^{6} IR_i w_i\right) , \qquad (4)$$

which yields values in the range $s_f \in [-0.2, 2.2]$. That is, the forward unit yields any speed between fast forward and slow backwards. The activation of the turn unit s_t is given by

$$s_t = \tanh\left(\sum_{i=1}^{3} IR_i w_i - \sum_{i=4}^{6} IR_i w_i\right) . \qquad (5)$$

The left and right motor activations M_l, M_r are fi nally

$$M_l = s_f + s_t,\ M_r = s_f - s_t\ . \qquad (6)$$

Both motor activations are in the range $M_l, M_r \in [-1.2, 3.2]$.

As already mentioned, the value system uses an additional non-moving sensor *NM*. This sensor is derived from the infrared sensors in the following way. For each frontal sensor, the difference between the current and previous time step is maintained $\Delta IR_i^t = IR_i^t - IR_i^{t-1}$, where the superscript indicates time. The vector length

$$nm = \sqrt{\sum_i (\Delta IR_i^t)^2} \tag{7}$$

is then used to update the sensor in a set-to-peak decay fashion:

$$NM \leftarrow \begin{cases} nm & \text{if } nm \geq NM \\ \tau_{nm} NM & \text{if } nm < NM \end{cases} \tag{8}$$

Essentially, the *NM* sensor is a *flow* sensor as is omnipresent in nature. This sensor monitors the amount by which the activation of the infrared sensors are changing, since high changes indicate movements. The *NM* sensor value decays with the time constant τ_{nm}, and is set to the current change value, if the current change value exceeds the current sensor value. If the *NM* sensor value falls below a certain threshold, the value system assumes that the robot is not moving. As is shown in Section 6, the use of the *NM* sensor improves the robustness in cases where the *SD* sensors fail. The implementation uses the *NM* sensor, since Khepera does not feature other motion sensors, such as a speed measure device/wheel.

Prior to each calculation of the motor activations, the connection weights w_i are updated depending on whether or not the value system indicates a collision. In case of a collision, the weights are increased according to

$$w_i \leftarrow w_i + \eta IR_i \ , \tag{9}$$

where η is the learning rate. In case of a non-collision, the connection strength is reduced by a decay factor ϵ

$$w_i \leftarrow \epsilon w_i + w_i(1-\epsilon)(1 - IR_i) \ . \tag{10}$$

The motivation of the learning rules is as follows. In case of a collision, learning rule (9) increases each weight w_i in proportion to the activity of its corresponding sensor IR_i. This increase leads to a stronger inhibition of the forward unit s_f and a stronger excitation of the turn unit s_t. Depending on the obstacle, the ambient light, the robot's morphology, etc., the robot is eventually able to move away from the obstacle. In the non-collision case, learning rule (10) decreases each connection strength in order to avoid overgeneralization. This decrease depends on the connection strength w_i *and* the sensor reading IR_i. If sensor i does not sense any obstacle, i.e., $IR_i \approx 0$, the weight w_i remains unchanged. If sensing an object $IR_i \gg 0$, however, the weight is decreased, since the high sensor reading does not lead to a collision. It should be noted that by applying learning rule (10), the connection strength w_i cannot increase, since $0 \leq IR_i \leq 1$. The intention of both learning rules (9) and (10) is to project the collision-sensor correlation onto the weights $\boldsymbol{w}$.

Even though the STP architecture is inspired by the DAC architecture, significant differences can be identified. First, the STP architecture does not feature an output layer, in which each unit represents a fixed, predefined reflex. Rather, different STP modules/processes generate continuous activations, which are combined at the motor side. Second, both the STP controller and the STP value system use different sensory modalities; the value system makes its decision upon proprioceptive activity patterns. This design decision is justified by the assumption that such proprioceptive sensors are *more reliable* than visual sensors, such as a CCD camera or infrared proximity sensors. Third, since one of the learning rules is always applied in each step, the weights continuously change, which increases the chance of choosing appropriate actions in erroneous situations, such as sensor faults. The main goal of the STP architecture is to yield adaptive and robust behavior, and Section 6 shows the degree to which extend this goal has been achieved.

5 Methods

The proposed STP architecture was implemented on the Khepera robot and was tested in different environments. The first environment is an arena that is shown in Fig. 7. The arena is of size 60x45 cm and the walls are made from wood with a height of 3 cm. This arena has narrow corridors at many locations. As already explained in Fig. 4, the DAC architecture does not exhibit any useful behavior in this environment, since these corridors trigger backup-and-turn reflexs most of the time.

The second environment consists of an arena of size 100x150 cm with white walls of height 30 cm. Here, 10 obstacles of size 10x10 cm were randomly distributed.

Unless otherwise stated, the following parameter settings were used. The learning rate and decay factor were set to $\eta = 0.1$ and $\epsilon = 0.95$. All weights were initially set to $w_{1 \le i \le 6} = 0.5$, and in all experiments, the speed scale factor was set to $s_s = 3.0$ (see also the Appendix).

The value system assumes a collision, if one of the speed difference sensors *SD* reports a value greater than 0.8. The non-moving sensor uses a decay factor $\tau_{nm} = 0.95$.

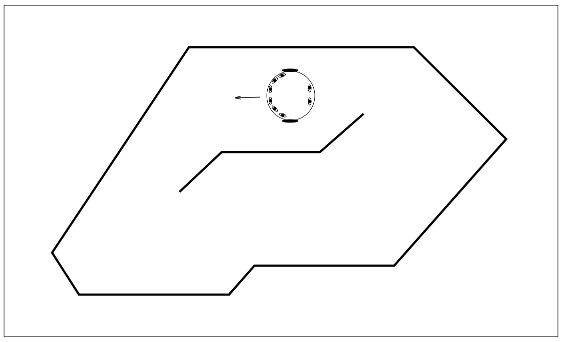

Fig. 7. The first test environment.

The value system also assumes a collision, if the *NM* sensor yields a value smaller than 0.15 *and* if the averaged infrared sensor activity is above 0.2.

In the presence of noise, it can be very useful to low pass input signals. A low-pass filter can be implemented by $x \leftarrow x(1 - t_c) + t_c I$, where x is the low-passed signal, t_c is a time constant[2], and I is the input signal.

In the experiments reported in this chapter, only the speed difference sensors *SD* were low-pass filtered with a time constant $t_{sd} = 0.7$. In the presence of ambient light produced by electrical bulbs, low-pass filtering of the infrared sensors would be of particular interest. However, the 100 Hz noise[3] cannot be appropriately filtered, since sensors can be read only once within a 100 ms time interval. If low-pass filtering of the infrared sensors were applied, the controller would become too slow to yield reasonable behavior. Therefore, low-pass filtering of the infrared signals was not considered.

6 Results

The STP architecture was tested in different environments under various environmental conditions, and exhibited the expected adaptive behavior. This section discusses some of these experiments in full detail and explains some of its adaptive behavior properties.

In the first arena in which the DAC architecture does not exhibit reasonable behavior, the STP architecture, in contrast, had no problem moving around. Normally, it moved straight ahead and avoided obstacles. It sometimes collided with a wall, and after some steps, moved away continuing with moving around. The robot never became stuck in any run. Collisions occurred infrequently, with each collision taking approximately 8-10 steps.

Figure 8 shows how the left and right weights $w_{1 \le i \le 3}$, $w_{4 \le i \le 6}$ developed over 1000 time steps, which were arbitrarily chosen from a long run. Figure 8 clearly shows that the value system recognized a collision at around time steps 120, 260, 390, 770, and 890. A collision caused a significant increase of those weights w_i that receive high activity from their associated infrared sensors IR_i (see also learning rule (9)). Between two subsequent collisions, each weight was slowly reduced in proportion to the activity of its associated infrared sensor.

Further observations can be made from Fig. 8. First, the decrease resembles a $\exp(-cx)$ function, which can be expected from learning rule (10). Second, horizontal line segments indicate that in these time steps, the associated infrared sensor had no activity. Third, most of the time, the lateral connections w_1 and w_6 had the smallest values. Mostly, these two weights were quickly decreased, since the high activity of a lateral sensor was not correlated with a collision. For example, while following a wall, the lateral sensor had a high activity close to the value 1, and the weight was decreased, since the value system triggered learning rule (10). Conversely, the frontal weights were slowly decreased, since the frontal sensors sent much less activity. Fourth, Fig. 8 shows that the robot experienced more collisions on the left-hand-side, which indicates that in

[2] Actually, t_c represents the reciprocal of a time constant, i.e., t_c is the first derivative of $dx(t)/dt$ of a continuous signal.

[3] The light intensity of electrical bulbs is oscillating with a 100 Hz (120 Hz in the USA) due to the alternate current of the power supply.

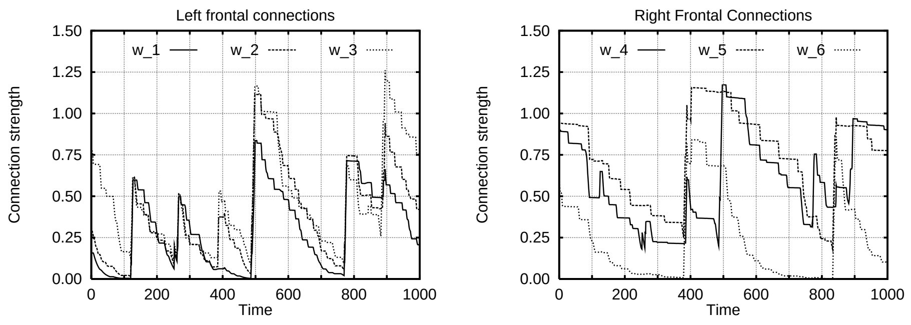

Fig. 8. The evolution of the weights $w_{1 \le i \le 6}$ in the first test environment *without* any ambient light. It can be seen that in case of a collision, those weights are increased that have a high activation of their corresponding infrared sensors. During normal operation, the all weights are slowly decreased in proportion to the infrared sensor activation, which is normally low for the two frontal sensors w_3 and w_4.

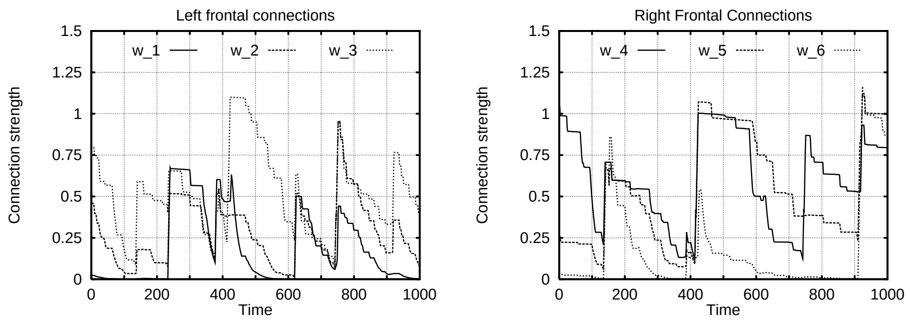

Fig. 9. The evolution of the weights $w_{1 \le i \le 6}$ in the first test environment *with* ambient light.

this particular time series, just by chance, the robot was mainly circling around clockwise.

The next example consists of the same experimental setup, including all parameter settings, but with signifi cant ambient light produced by electrical light bulbs. As explained in the Appendix, ambient light signifi cantly modifi es the sensor readings for an unchanged agent-object distance. Different sensor readings obviously lead to a modifi ed control characteristic given by equations (4) and (5). Figure 9 shows that consequently, the STP architecture evolves different connection strengths.

The way in which the STP architecture adapts to changed sensor readings can be seen from Fig. 10, which shows the connection strength averaged over the shown 1000 time steps of all six weights in a dark environment (left bars) and under ambient illumination (right bars). From Fig. 10, it can be seen that ambient illumination leads to a higher average connection strength for the frontal sensors and a lower average connection strength for the lateral sensors. In a dark environment, the infrared sensor readings are generally lower. A lower sensor reading would lead to a smaller distance to obstacles, which in turn, would increase the probability of a collision. Consequently, the connections remain at higher values, since learning is triggered earlier. The frontal connections are only slightly affected, since obstacles do not appear in front of the agent that often. It should be noted that the observer can hardly distinguish the behavior under different illumination.

In the second environment, the STP architecture easily controls the robot. Most of the time, all infrared sensors yield very low activation that is on the noise level. Consequently, the weights w_i remain constant over longer periods of time. In other words, the much smaller density of obstacles is represented in higher average values of the connections.

The last experiment studied the behavior of the STP architecture when the surface-wheel friction was not high enough to sufficiently slow down the wheels in case of a collision. This situation is depicted in Fig. 11, which plots the activity of the non-moving sensor *NM*, and the two speed contributions s_f and s_t. At time step 8, the robot collided with an obstacle. In the presented situation, the *SD* sensors did not have significant activity, since the wheels were freely spinning. The robot, however, was not moving, and consequently, the infrared sensors sent constant values. Therefore, the activity of the *NM* sensor fell below the threshold, and at time step 44, the value system recognized the collision. The STP architecture then behaved as already discussed.

A significant amount of experimentation was devoted to sensor changes and failures. To this end, the sensor readings of selected sensors were artificially multiplied by a constant $c \in [0.1, 10]$ or were even set to the constant values 0 or 1. In all cases, the STP architecture exhibited reasonable behavior. When setting a sensor to the constant value 1, the STP architecture had the most problems. In this case, the robot was spinning after a collision for a few steps until “normal” behavior could be observed. In all other cases, the observable behavior was hardly changed.

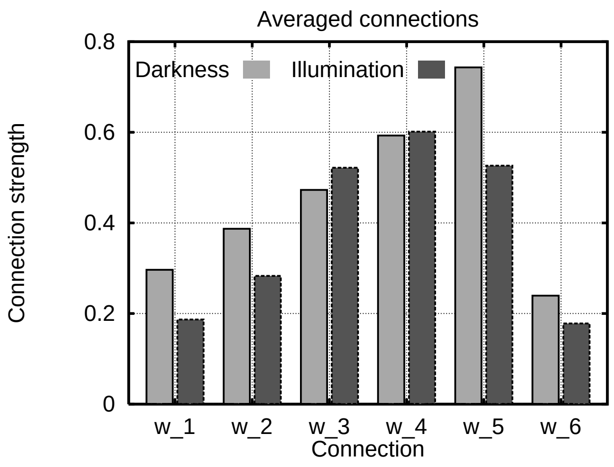

Fig. 10. Connection weights averaged over 1000 time steps in a dark (light grey bars) and an illuminated (dark grey bars) environments.

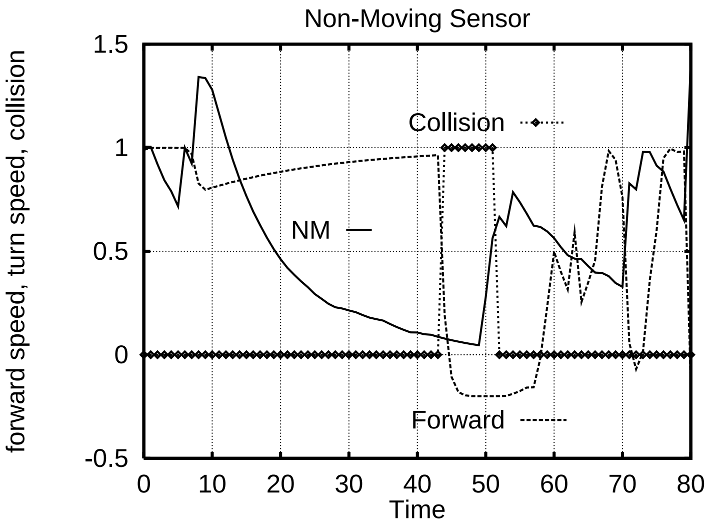

Fig. 11. Recognizing a collision by the non-moving sensor *NM*.

Finally, this section discusses some parameter settings mentioned in Section 5. The value system recognizes a collision if the *SD* sensors values exceed a threshold normally set to the value 0.8. This parameter setting is not critical. When using a value greater than 0.8, the value system requires some more steps to recognize a collision. This extra amount of time is required to further weaken the forward unit's inhibition (equation (4)), which is due to the weight decay caused by learning rule (10). Similarly, changing the values for the learning rate η and decay factor ϵ leads to slight changes in the observable behavior. A decay factor ϵ closer to 1 would stretch the time between two subsequent collisions. However, a smaller decay factor allows for faster adaptation to new environments.

7 Discussion

This chapter has proposed a new self-organizing-through-proprioception (STP) architecture that is signifi cantly inspired by the DAC architecture already presented in Pfeifer and Verschure [25] and Verschure *et al.* [31]. From the DAC architecture, STP adopts the general idea of using a value system for training the actual controller. The other components have been added or modifi ed as follows. The controller of the STP architecture employs different modules that send continuous values to the motors, rather than employ a set of (action) reflexes that are triggered in an on/off mode. Furthermore, the value system has been decoupled from the controller by using different sensory modalities; the value system uses *proprioceptive* activity patterns originating from the motor side. By employing *additional* proprioceptive feedback, the STP architecture is able to monitor and evaluate its own controller. In other architectures, such as DAC, the effect of the control (motor) actions can be observed only indirectly through the environment and visual sensors. The experimental results presented in this chapter have shown that the STP architecture adapts to different environmental conditions, and that it exhibits very robust behavior with respect to sensor changes and failures.

As already mentioned, the use of proprioceptive information is omnipresent in nature. Proprioceptive sensors allow an organism to monitor and evaluate its own behavior. Furthermore, the number of proprioceptive sensors is much smaller than the number of audio or visual sensors, such as receptors in the retina. Since these sensors are protected by the body and just monitor internal information sources, they are much less sensitive to external disturbances, such as additional light source or temperature changes. The remaining task is to build a mapping between external sensory information and the expected behavior.

In order to maintain focus on the general ideas, the implementation of the STP architecture was not tailored for the Khepera robot. The fi rst example that concerns this point is given by Khepera's on-board PID controllers, which help the designer operate the robot in a forward mode, since the PID controllers (try to) regulate the motors such that their speeds are as close as possible to the desired settings. In other words, the designer can decide, which motor speed might be appropriate for a given situation, and the desired speed is probably maintained by the on-board PID controllers. This *forward* strategy, however, hides a large amount of information, which makes it harder to detect a collision and impossible to detect whether the robot moves uphill or downhill. Further control architectures to be developed by future research might implement these PID controllers themselves in order to make additional sources of feedback signals available. These additional feedback signals would allow for tuning further system characteristics that depend on the agent-environment interaction.

As a second example, the speed difference sensors *SD* could have been tailored for Khepera. It is theoretically conceivable that during a collision, the weights might be increased such that the robot turns and tries to move backwards into the obstacle. In this situation, the *SD* sensors would still indicate a collision, and the robot would be trapped. This problem can be overcome by taking the *SD* sensors into account only if the desired speed is forward directed. In the fi rst place, this problem arises from the robot, and was therefore not considered in the previous sections, since it can be easily fi xed, as already outlined.

The stability of the integrated agent-control system is of major concern. For this goal, some important time constraints have to be considered. For stability of such feedback control systems, the time constants of outer loops should be higher than those of inner loops. For example, the PID controllers should be slower than the actual motors, and the *SD* sensor readings should be low-passed such that they change slower than the motor speeds; otherwise a temporary speed lag could be considered a collision. When adding further control loops to self-adapt further parameters, such as the learning and decay rates η and ϵ, the possible changes should be much smaller than the average weight modifi cations. Otherwise instability could emerge due to oscillations. A possible option for self-adaptation of η and ϵ could be the defi nition of an additional need or an additional performance measure, such as the overall energy consumption of the robot.

Further research will be dedicated to improve the value system. Besides adding some of the features discussed above, the main focus will be on the learning rules. The weakest point of the value system is that it triggers *either* learning rule (9) *or* (10). It seems desirable to develop a value system that activates each learning rule to a certain degree. As a result, all learning rules would establish a dynamic equilibrium, rather than "jumping" between discrete stages. It seems that in this way, many more learning components could be simultaneously considered, since the value system would not need to select a particular learning rule in a winner-take-all style. Another important topic for future research will be the integration of a CCD camera. Since the value system is (besides the non-moving sensor *NM*) entirely based on motor feedback, it can be expected that this integration does not cause further diffi culties.

Acknowledgements

The author gratefully thanks Christopher Lueg and Peter Eggenberger for helpful discussion. Many thanks are due to David Fogel for encouragement and special thanks are due to Dimitrios Lambrinos for providing the DAC implementation on Khepera.

References

1. Bäck, T., Schwefel, H.-P.: An Overview of Evolutionary Algorithms for Parameter Optimization. Evolutionary Computation **1** (1993) 1-23
2. Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation **2** (1986) 14-23
3. Brooks, R.: Elephants Don't Play Chess. Robotics and Autonomous Systems **6** (1990) 3-15.
4. Brooks, R.: Intelligence Without Reason. In: Mylopoulos, J., Reiter, R. (eds.): Proceedings of the 12th International Conference on Artificial Intelligence (IJCAI-91). Morgan Kaufmann, San Mateo, CA (1991) 569-595
5. Brooks, R.: Intelligence without representation. Artificial Intelligence **47** (1991) 139-159
6. Clancey, W.: The Frame of Reference Problem in the Design of Intelligent Machines. In: Lehn (ed.), Architectures for Intelligence. The 22nd Carnegie Mellon Symposium on Cognition. Erlbaum, Hillsdale, NJ (1991) 357-423,
7. Cliff, D., Husbands, P., Harvey, I.: Evolving Visually Guided Robots. In: Meyer, J.-A., Roitblat, H.L., Wilson, S.W. (eds.) From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA (1992) 374-383
8. Etzioni, O., Weld, D.: A Softbot-Based Interface to the Internet. Communications of the ACM **37**:7 (1994) 72-76
9. Floreano, D., Mondada, F.: Evolution of Homing Navigation in a Real Mobile Robot. IEEE Transactions on Systems, Man, and Cybernetics - Part B, **26**:3 (1996) 396-407
10. Floreano, D., Mondada, F.: Evolution of Plastic Neurocontrollers for Situated Agents. In: Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., Wilson, S.W. (eds.): From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA (1996) 402-410
11. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company (1989)

12. Harvey, I., Husbands, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary Robotics: the Sussex Approach. Robotics and Autonomous Systems, Special Issues on "Practice and Future of Autonomous Agents" **20**:2-4 (1997) 205-224
13. Kandel, E., Schwartz, J., Jessel, T.: Essentials of Neural Science and Behavior. Appleton and Lange, Norwalk, CU (1995)
14. Khepera Users Manual Laboratoire de microinformatique, Swiss Federal Institute of Technology (EPFL), 1015 Lausanne, Switzerland
15. Kohonen, T.: Self-Organization and Associative Memor. Springer-Verlag, Berlin, Germany (1989)
16. Lambrinos, D.: Navigating with an Adaptive Light Compass. In: Moran, F., Moreno, A., Merelo J.J., Chacon, P. (eds.): Proceedings of the Third European Conference in Artificial Life ECAL'95. Springer-Verlag, Berlin, Germany (1995) 602-613
17. Lambrinos, D., Scheier, C.: Building Complete Autonomous Agents: A Case Study on Categorization. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems IROS'96. IEEE Press (1996) 170-177
18. Maes, P.: Agents that Reduce Work and Information Overload. Communications of the ACM **37**:7 (1994) 31-40
19. Maris, M. te. Boekhorst, R.: Exploiting Physical Constraints: Heap Formation through Behavioral Error in a Group of Robots. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems IROS'96. IEEE Press (1996) pp. 1655-1660,
20. Newell, A., Simon, H.A.: Physical symbol systems. Cognitive Science **4** (1976) 135-183
21. Nolfi, S., Parisi, D.: Learning to Adapt to Changing Environments in Evolving Neural Networks. Adaptive Behavior. **5**:3 (1997) 343-363
22. Pfeifer, R.: Building "Fungus Eaters": design principles of autonomous agents. In: Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., Wilson, S.W.: (eds): From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA (1996) 3-12
23. Pfeifer, R., Scheier, C.: Sensory-Motor Coordination: the Metapher and beyond. Robotics and Autonomous Systems, Special Issue on "Practice and Future of Autonomous Agents." **20**:2-4. (1997)
24. Pfeifer, R., Scheier, C.: Understanding Intelligence. MIT Press, Cambridge, MA (1999)
25. Pfeifer, R., Verschure, P.: Distributed Adaptive Control: A Paradigm for Designing Autonomous Agents. In: Varela, F., Bourgine, P. (eds.): Proceedings of the First European Conference on Artificial Life. MIT Press, Cambridge, MA (1992) 21-30
26. Rechenberg, I.: Evolutionsstrategie. Frommann-Holzboog, Stuttgart, Germany (1973)
27. Salomon, R.: Neural Networks in the Context of Autonomous Agents: Important Concepts Revisited. In: Dagli, C.H., Akay, M., Chen, C.L.P., Fernández, B.R., Ghosh, J. (eds.): Proceedings of the Artificial Neural Networks in Engineering (ANNIE'96). ASME Press, New York, NY (1996) 109-116
28. Salomon, R.: Increasing Adaptivity through Evolution Strategies. In: Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., Wilson, S.W. (eds.): From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA (1996) 411-420
29. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley and Sons, Inc, New York, NY (1995)
30. Simon, H.A.: The Science of the Artificial, 2nd edn. The MIT Press, Cambridge, MA (1976)
31. Verschure, P., Kröse, B., and Pfeifer, R.: Distributed Adaptive Control: The Self-Organization of Behavior. Robotics and Autonomous Systems **9** (1992) 181-196
32. Wooldridge, M.J., Jennings, N.R.,: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review **10**:2 (1995) 115-152

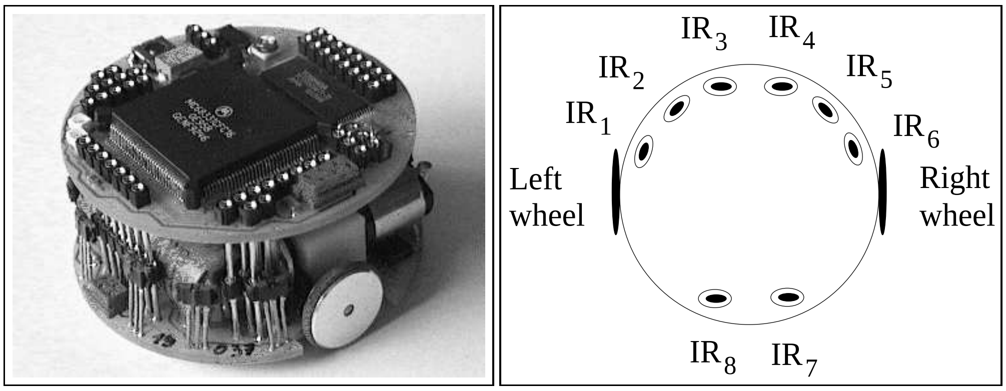

Fig. 12. The Khepera robot and the approximate location of the infrared sensors $IR_1 \ldots IR_8$.

Appendix: The Khepera Robot

Figure 12 shows the Khepera robot, which has been used in all the experiments presented in this chapter. Its body is 32 mm high and 55 mm in diameter. The robot is equipped with eight infrared and eight ambient light sensors, as well as two motors. Khepera's sensors and motors are controlled by a Motorola 68331 microcontroller. The robot can operate in two modes. In the first mode, a program is downloaded into the onboard memory, which allows Khepera to operate without any further hardware. In the second mode, which was used in all experiments presented here, Khepera is connected to a workstation via a serial link. A detailed description of the robot and its electrical parts can be found in the *Khepera Users Manual*. Due to A/D conversions and data transmissions to the host, reading the sensors and setting the motor speeds require approximately 100 ms. During the 100 ms time interval, the robot moves with constant speed.

The sensor readings of Khepera's eight infrared sensors are integer values in the range $[0, 1023]$. The interface normalizes the sensor readings such that the values are in the range $[0, 1]$. The sensors give reasonable input values for object distances between 10 mm and 60 mm. A sensor value of 1023 indicates that the robot is very close to an object, and a sensor value of 0 indicates that the robot does not receive any reflection of the infrared signal. It should be noted that the actual sensor reading for a constant agent-object distance depends on the ambient light, the material the sensed object is made of, and the sensor itself. Furthermore, each sensor reading is disturbed by additional noise as is illustrated in Fig. 13, which shows the normalized sensor reading for a constant agent-object distance under different illuminations.

Khepera can control each motor independent of the other by sending commands to the robot. Valid speed values are integer values in the range $[-40, 40]$. Khepera's onboard PID controllers compensate the robot's dynamics and control the motors such that the actual speed is as close as possible to the desired speed. By means of attached wheel encoders, the robot measures the motor's real speed, which differ from the command

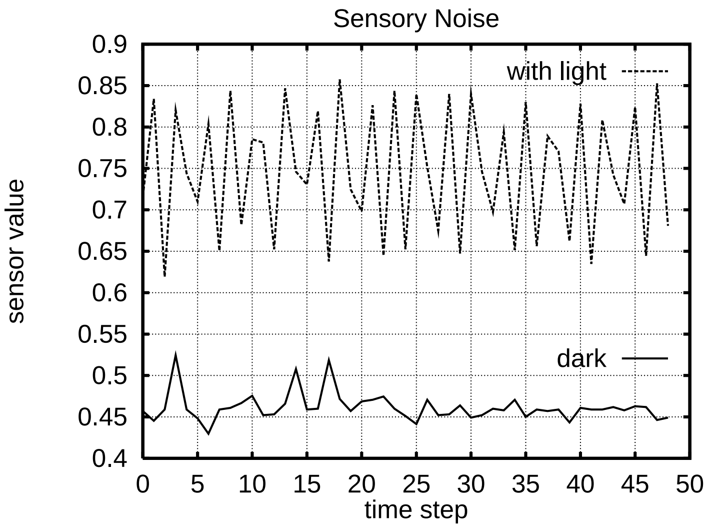

Fig. 13. Examples of the noisy sensor readings with ambient light and in darkness.

setting in situations, where the robot cannot move. The real speeds can be obtained by sending special commands to the robot.

The interface maintains a speed scale factor s_s, by which the motor activities M_l and M_r are multiplied prior to setting the motor speeds. Similarly, the incoming wheel encoder values are divided by the same factor.

Khepera's limitation to integer speed values requires a floating-point-to-integer conversion. In order to avoid this artifact, the interface adds [0,1]-equal-distributed random numbers to the motor speeds at each time step. By this means, a controller output of, for example, 0.5 is converted into a "0-1-. . ." sequence. Without this special conversion, a continuous control output would be mapped onto a step function, which can cause further problems.