A Behavior-Based Mobile Robot Architecture for Learning from Demonstration

Michael Kasper, Gernot Fricke, Katja Steuernagel, Ewald von Puttkamer

Robotics Research Group, Department of Computer Science, University of Kaiserslautern,
Postbox 3049, 67663 Kaiserslautern, Germany

Tel: +49-631-205-2886; fax: +49-631-205-4409
e-mail: kasper@informatik.uni-kl.de

Abstract

Autonomous Mobile Robots (AMR), to be truly flexible, should be equipped with learning capabilities, which allow them to adapt effectively to a dynamic and changing environment. This paper proposes a modular, behavior-based control architecture, which is particularly suited for "Learning from Demonstration"-experiments in the spatial domain. The robot learns sensory-motor-behaviors online by observing the actions of a person, another robot or another behavior. Offline learning phases are not necessary but might be used to trim the attained representation. First results applying RBF-approximation, growing neural cell structures and probabilistic models for progress estimation are presented.

Keywords: Online Learning from Demonstration; Behavior-based architecture; Behavior classification; Growing neural net; Mobile Service Robot

1. Introduction

Behavior-based approaches have been established as a main alternative to conventional robot control in recent years [1]. Due to their modular architecture, these approaches provide high scalability, while limiting the complexity of the individual modules. These can be implemented (or taught) and tested independently, and they directly support software re-use. Furthermore, they meet real-time requirements in a dynamic environment by creating a tight coupling between sensing and acting.

Autonomous Mobile Robots (AMR) need to be equipped with learning capabilities as an essential prerequisite in order to adapt effectively to dynamic and varying environments. This is especially true for the growing field of service robotics, where non-professionals are intended to operate complex mobile robot systems. In this context, *Programming by Demonstration* (or from the viewpoint of the robot: *Learning from Demonstration*) is an interesting alternative to conventional robot programming for learning new skills (behaviors), improving already existing ones or generating new combinations of them.

While the field of robot learning has been an intensively studied research topic over the last decade [9] within the behavior-based robotics community, research on *Learning from Demonstration (LFD)* concentrated mainly on learning *reactive behaviors*, i.e. simple stimulus-response connections. Approaches going beyond reactive behaviors (e.g. [6] describing reinforcement learning of planning rules) are rarely known. Hence, learning from complete temporal sequences of perceptions (rather than from single perceptions) is still an open question [11].

This is only one problem, which is addressed within the *MOBOCOB*-project (***mo**bile ro**bo**t control by **co**ncurrent **b**ehaviors*). In the context of this project the authors developed a framework for investigating learning techniques for behavior-based architectures. The main focus of our ongoing studies lies on *Learning from Demonstration* for temporal sequences in the spatial domain. After a number of good examples have been taught e.g. by a human teacher, the robot is able to imitate the teacher, and moreover to generalize from the given examples.

Fig. 1. Robot *PHOENIX* following a person

Fig. 1 shows robot *PHOENIX* performing the reactively learned task to follow a person by using its onboard laser range finder (LRF) Sick LMS 200.

After a short teaching phase of approx. 2 minutes, during which the robot was shown a couple of examples, the machine was able to reproduce the demonstrated behavior immediately. If for more complex tasks the initial training is not sufficient, specific situations can be taught *additionally*. The robot will subsequently be able to master these situations, too.

The presented paper is organized as follows. The next section gives a formal motivation for behavior-based control. The *MOBOCOB*-architecture is introduced in Section 3. A classification of behaviors based on the formal motivation is defined in Section 4, followed by a brief survey of different aspects of learning in Sections 5. Section 6 describes the learning techniques used within *MOBOCOB*. Finally, first experimental results are presented together with some concluding remarks.

2. Motivating behavior based systems

While behavior-based approaches in robotics are mostly known to be motivated from ethology and (behavioral) psychology [2], we will introduce a more formal motivation for them.

Technically, an AMR consists besides its auxiliary components (batteries, wheels, etc.) of a set of sensors S to perceive the environment, some actuators A to modify the environment (or the robot's configuration in the environment), and a digital control system, which is equipped with some memory Z.

From a mathematical point of view, mobile robot control appears to be a simple problem, since all we need is a function f, which maps the sensor input s to some actuator output a with respect to the internal memory state z:

$$f\colon (s,z) \rightarrow (a,z) \qquad \text{or} \qquad (a,z') = f(s,z)$$

Unfortunately, the desired transformation is quite complex. While the dimension of a is typically small (e.g. a tupel (v,ω) for controlling the robot's movement by specifying its linear and angular velocity), the dimension of the sensor input s can be very high and, even worse, the dimension of the internal state space, which is needed to perform a specific task, may not be known.

In general, we will not be able to find a closed term representation for f. However, we can try to reduce complexity by splitting the domain and dividing the problem into piecewise defined sub-tasks. Thus, we get for disjoint domains D_i:

$$(a,z') = \begin{cases} f_1(s,z) & \text{if } (s,z) \text{ is in } D_1 \\ f_2(s,z) & \text{if } (s,z) \text{ is in } D_2 \\ \dots & \\ f_n(s,z) & \text{if } (s,z) \text{ is in } D_n \end{cases}$$

Alternatively, if we transfer the decision of domain membership into each function f_i (which is reasonable, since the functions „know“ their domain best), we can write:

$$(a,z') = f_1(s,z) \cup f_2(s,z) \cup \dots \cup f_n(s,z)$$

Since sensor input as well as actuator output, and the amount of internal memory does not need to be the same for each function f_i, we yield:

$$(a,z') = f_1(s_1,z_1) \cup f_2(s_2,z_2) \cup \dots \cup f_n(s_n,z_n)$$

This equation already describes a behavior-based architecture. Each f_i denotes an individual behavior and the arbiter corresponds to the union-operator. Since behaviors are usually implemented as individual processes and we do not demand the z_i to be disjoint, behaviors can share memory, which is helpful for inter-process communication.

Before we use this formal motivation to establish a classification of behaviors in Section 4, we introduce the *MOBOCOB*-architecture, to give an example of an actual behavior-based system.

3. The MOBOCOB-architecture

MOBOCOB is implemented on the experimental mobile robot *PHOENIX* (see Fig. 1) which was developed within the *CAROL*-project [5]. *PHOENIX*, a differential drive mobile robot running under the commercial real-time-operating-system QNX, is equipped with a laser range finder (Sick LMS-200), a pan&tilt-video system and both, ultrasonic and infrared proximity sensors. Calculations are performed using two onboard Pentium PC's, which are connected via a wireless Ethernet to the research group's LAN. A laptop, optionally mounted on top of the robot supplies an additional user-interface.

The *MOBOCOB*-architecture is an experimental platform, which utilizes a flexible, modular concept for implementing behaviors, an arbitration unit, and components for sensor and actuator control. All modules share a common software-interface, which easily allows to add new modules such as behaviors or virtual sensors. Fig. 2 depicts the architecture's overall structure. Its basic components are discussed in the following sections.

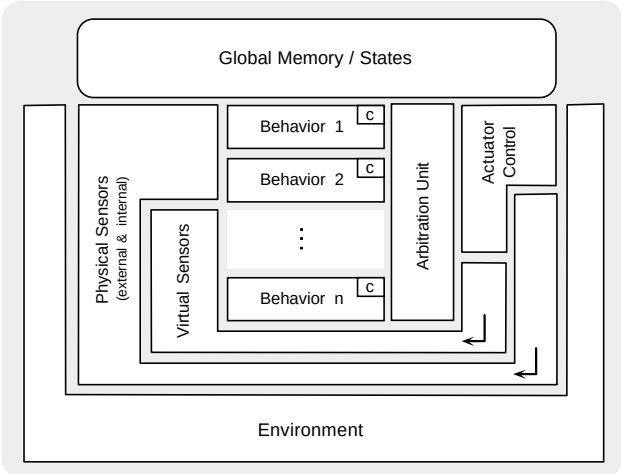

Fig. 2. MOBOCOB-Architecture (data-flow from left to right)

3.1. Sensor and Actuator Modules

Physical sensors either observe the external environment or provide data about the internal state of the robot. In order to reduce the dimensionality of sensor input, it is reasonable to perform some kind of sensor data preprocessing such as filtering or feature detection within video- or range-images. This is done by *virtual sensors* which process the output of physical or other virtual sensors. In this way, a higher abstraction of perceptions can be achieved. Virtual sensors are typically used for sensor fusion, feature detection, object tracking or adding history to physical sensors. They may also be used to observe the output of other modules, such as behaviors or the arbitration unit.

In the person-following example given previously, a virtual sensor "human detector" was used, to detect persons in a horizontal LRF-scan. This virtual sensor utilizes a simple model of leg movement to generate human position hypotheses. Fig. 3 depicts some of these hypotheses in the two-dimensional laser scan.

Actuator modules are much like sensor modules, with the difference, that they consume data rather than producing it. To provide access to both, sensors and actuators an object-oriented generic interface has been implemented. It defines standard access-, data-manipulation and -evaluation functions, such as instantiating new sensors or actuators (*short: SA's*), reading and writing data or calculating the weighted average and the similarity of SA-data-sets.

The same library functions are used to access all SA's, regardless of the underlying data types. A parameter string of the form "<sa-name>: <arg 1>; ... <arg n>", describing the sensor-type and the required arguments is all a module needs, to access any SA. For example, initializing a new entity of a laser range finder can be done by using the following string: "LaserScanner: sectors=48; min=1; max=8000; blur=0", which means that a sensor module for the laser scanner with an angular resolution of 12 sectors and a maximum range of 8 m is initialized.

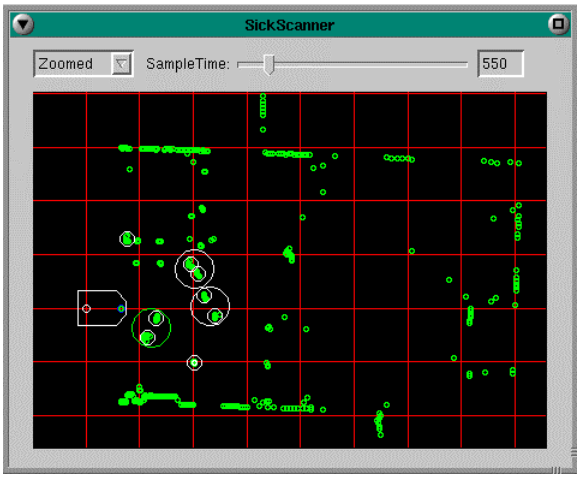

Fig. 3. Possible locations of persons found by the virtual sensor (indicated by large circles)

It is noteworthy, that even for a single physical sensor, different parameter configurations are allowed at the same time, as long as they are compatible. This implies that one sensor's configuration comprises the others. If for example, two distinct behaviors require a different angular resolution of the laser scanner the sensor module initializes the hardware to deliver the higher resolution, while the lower one is automatically calculated from the raw data.

Parameter strings can also be used to group sensors to sensor sets, which are called *multi-sensors*. The standard functions of the interface are recursively applied to each individual sensor of the set. For any combination of sensors and their data types, the library functions are sufficient to access them.

As will be shown in chapter 6, the learning algorithms for behavior modules do not need any a priori knowledge of the actual data structures delivered by a sensor or a set of sensors. Instead, the algorithms rely exclusively on the library functions of the generic interface which encapsulates the SA's raw data with standardized data headers. This makes the learning module completely independent from the given sensors and actuators. Only the parameter string describing the SA's, as well as the learning parameters may vary.

Fig. 4 depicts the communication scheme for the sensor/behavior communication.

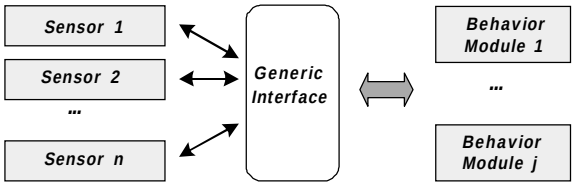

Fig. 4. Data flow using the generic sensor/actuator interface

3.2. Behavior Modules

Within *MOBOCOB*, *behaviors* are realized as parallel QNX-processes, which exchange data via common communication channels. Please recall, that in Section 2, the decision of domain membership was transferred to the individual behaviors. Thus each f_i uses a competence-value c to convey its competence concerning a given situation to the *arbitration unit*. For our further formal considerations, we will omit this value, since it can be seen either as part of the actuator- or state-output of a behavior.

3.3. Arbitration Unit

The *arbiter* observes the output commands produced by individual behaviors and uses the competence-values to generate an overall command set, which is passed to the actuator control unit. Since we do not want to restrict the arbiter to a specific arbitration scheme, competitive behavior selection is possible as well as cooperative behavior fusion, or any combination between them. The arbiter is hierarchically organized and also responsible for some kind of behavior sequencing (compare assemblies or engagement-modules in [2]). In this context, the arbiter can trigger behaviors or can halt and restart the associated processes.

4. Classification of Behaviors

Please recall our motivation from Section 2. Depending on the domain and co-domain of the describing functions, we distinguish four main types of behaviors[1]:

Hidden behaviors. Hidden, most likely deliberative behaviors do not control actuators directly. They can be characterized as f: $(s,z) \rightarrow z$. Usually, processing at this level does not only occur in a sub-symbolic manner, but mainly at a symbolic level for planning and reasoning about the environment. Hidden behaviors typically modify the robot's set of targets or its „motivational" state [2]. Obviously hidden behaviors are poor candidates for *LFD*, since they cannot be observed from a teacher. However, learning can be applied using unsupervised methods like reinforcement learning (see [6]).

Reactive motor behaviors. Purely reactive (reflexive) behaviors, do not depend on state information at all. They directly map sensor input to actuator output as denoted by the formula f: $s \rightarrow (a,z)$. The z component may be used for data transfer to other modules.

Blind motor behaviors. Although reactive behaviors are easy to handle, they solve a simple class of problems only. For example, when no sensor information is available, reactive behaviors are not able to initiate a sequence of actions. This leads to the class of blind motor behaviors, which do not rely on any (external) sensor information at all and which are described by f: $z \rightarrow (a,z)$.

State-dependent motor behaviors. Combining the classes of reactive and blind motor behaviors, we get state-dependent motor behaviors, which require some memory to accomplish a task, and which were the basis for the motivational introduction from above: f: $(s,z) \rightarrow (a,z)$.

Since hidden behaviors are not suited for *LFD* and blind behaviors are straight forward to realize, we focus on both, reactive and *history-dependent behaviors*: a subset of state-dependent behaviors, which are based on temporal sequences and can hence be represented by means of cycle-free state graphs.

5. Aspects of learning

Concerning a behavior-based system, there are many opportunities for applying machine learning techniques advantageously. They can be integral parts of various system components such as behavior modules, the arbitration unit, sensor data preprocessors, or actuator controllers. For some tasks, unsupervised learning is promising, for others supervised learning is adequate.

In this paper, we concentrate on *LFD* as a special type of supervised learning for behaviors. Especially for (motor) behaviors this technique seems to be straight forward and very promising. Instead of explicitly programming a behavior, a (human) teacher simply demonstrates a task to the robot by specifying, which sensors are relevant and by controlling the robot's actuators. This easily allows even non-professionals to teach a new behavior or to adopt an already existing one to new tasks, other environments or a different robot hardware (sensors and actuators). Furthermore, *LFD* can be a basis for implementing new behaviors, which can be subsequently improved by the robot itself, using unsupervised learning techniques,.

[1] This classification scheme takes up ideas of the Kalman decomposition. Alternative classifications can be found in [1] or [7].

There is no difference, whether teaching is performed by a person, another robot or simply by another behavior running on the same robot. The latter is called *behavior cloning* and is interesting for several reasons. One is to clone functionality using different sensors as input, which could be cheaper, faster or more reliable. Another reason is to copy a conventionally programmed behavior which can be improved or extended by further supervised or unsupervised learning. In any case, learning an individual behavior should be considered as approximating a single describing function rather than a set of functions.

The introduced classification of behaviors leads directly to a classification of solvable problems within the spatial domain. *Blind behaviors* are able to playback action sequences independently from any sensor data. This is necessary for instance, when sensor feedback is too slow or not available. From the robot's point of view, learning blind behaviors is simple, as long as they depend only on their own internal states (e.g. a time basis), rather than internal states z_i of other behaviors. However, since blind behaviors do not get any external feedback, they are restricted to short, non-critical action sequences.

Teaching *reactive behaviors* to mobile robots is state of the art and has been investigated by many researchers in the last years using various types of sensors [1], [10]. Teachable tasks include *wall following*, *obstacle avoidance*, *box pushing*, *docking*, *phototaxis* and so on [11]. However, since reactive behaviors just learn simple stimuli-response connections, they are not suited for any history- or state-dependent tasks.

For *state-dependent tasks*, a bijective mapping between sensors and actuators is not sufficient to describe the behavior. For instance, passing a door with a longish robot cannot be solved using reactive behaviors, if the used sensor covers only a limited area in front of the robot (Fig. 5.a). Because of the temporal loss of information about the door position, the robot would get stuck using a reactive behavior. Also, driving a robot with an Ackerman-steering into a parking-box is more than a reactive task, since there exist identical external sensor perceptions, which correspond to completely different actions (reverse direction, see Fig. 5.b).

Originally reactive tasks which 1) do not effect the environment and 2) have to be repeated a fixed number of times are also state-dependent, because the robot cannot derive from the sensor data, how many repetitions have already been accomplished. Figure 5.c exemplifies this by driving the robot around a totem pole for three times.

Some tasks which normally do not have a reactive solution may, however, be solved reactively, if the required memory is „hidden“ in some other components. For example, the door-passage problem could be solved reactively, if instead of the limited physical sensor a virtual 360°-sensor, based on a grid map, is used. The totem pole problem could be solved using an accumulative angular sensor, which takes the place of a counter.

However, such sensors would be task specific. For different problems, different sets of „history sensors“ would have to be implemented. Since they do not provide a general history model, one can easily think of problems, where their history representation is not sufficient.

A more universal concept is to use sequences of reactive behaviors [2]. They correspond to *history-dependent behaviors* introduced in Section 4. These are basically sufficient to solve a large set of robot navigation tasks, like the ones mentioned previously.

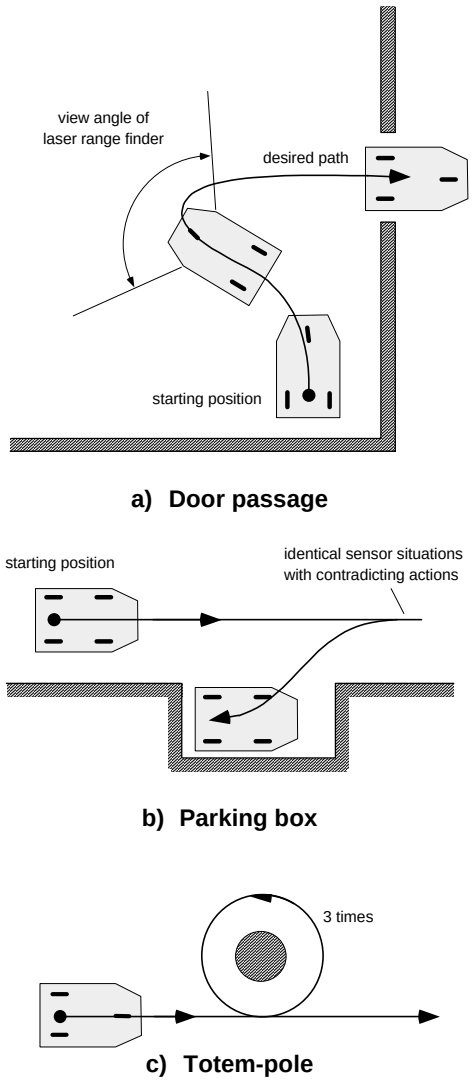

Fig. 5. Non-reactive robot tasks

Since the authors are not aware for any approaches applying *LFD* to history-dependent behaviors in mobile robotics, *MOBOCOB* tries to contribute to this topic. From the viewpoint of *LFD*, it is desirable that individual

behaviors already have sequencing capabilities. Thus teaching involves a single behavior only, and not the entire behavior-arbiter complex.

6. Learning within MOBOCOB

Within the *MOBOCOB*-project, a generic concept for learning-experiments has been developed. The learning module can observe any two communication ports, e.g. (virtual) sensor and actuator ports, and tries to find a mapping between them. Since most modules within *MOBOCOB* share the same communication concept, learning is not only restricted to behaviors, but can also be applied to virtual sensors or actuators.

Observing an existing behavior, the learner can be used for *behavior cloning* or *off-line cloning* (cloning an existing behavior with simulated sensor input, so different learning-algorithms and parameters can be compared under the same conditions).

For *LFD*, the learning module observes the output of a behavior, controlled by the teacher. In most cases, this output is provided by a joystick-module controling the motors of the robot via a (v,ω)-interface. If needed, also the competence-value for a behavior can be taught using a force-sensor, associated with a button of the joystick. The teacher specifies the behavior which should be taught, as well as the learning parameters (reactive/state-dependent, learning-rate, sensors to be used, etc.). Any set of (virtual) sensors can be selected using one parameter string. As mentioned before, the learning algorithm does not need to know any internals of the data-structures it learns from. It abstracts from the data, using the similarity- and average-functions, encapsulated within the generic sensor/actuator-library.

6.1. Learning reactive behaviors

What makes a task reactive is the fact that it does not use any state information to generate the output. Hence, it can be described as $a = f(s)$. We are now looking for an adequate approximation for f. A common mathematical technique is to represent an unknown function by a set of support points. The function then can be regarded as an inter- or extrapolation between these points.

We decided to use *Radial Base Functions (RBF)* with growing neural cell structures [8] to approximate f (and thus the behavior). This technique can cope with non-linearities and is well suited for our extension to history-dependent behaviors. While learning, the robot collects a set of stimuli-response pairs (s,a) describing the taught examples. We use these pairs to derive support points represented by neurons. Each neuron marks the center of a Radial Base Function rbf_i, which is used to interpolate between the support points.[2]

6.1.1. RBF-approximation

RBFs are radial symmetric, i.e. the value of a function depends on the distance of the input to its center, only. The value of $rbf_i(s)$ can be interpreted as how strong the support point $sp_i = (s_i,a_i)$ located in the center of rbf_i influences the output value of $f(s)$. In the center, it will be highly representative and in greater distance, it will not be representative at all.

We have chosen to use a Gaussian base function with its co-domain scaled to [0,1] together with the RBF-approximation defined as:

$$approx(s) = w\_average(rbf_i(s), a_i) = \sum_i \frac{rbf_i(s)}{\sum_j rbf_j(s)} \cdot a_i$$

It is noteworthy, that all rbf_i are different functions of the same type: They have different centers and may also have a different half-width σ, which is defined as the distance (from the center) where rbf_i falls under 0.5. In areas in which the approximated function varies with a high frequency, more support points will be needed than in other "smooth" areas. Appropriately choosing σ ensures that the ranges of influence of adjacent rbf_i do not overlap too much.

Despite the fact that RBF-approximation is defined on distance measurements, the implemented sensor/actuator-library defines similarity-measurements exclusively. As the libraries' interface may be used for any data type, it can not be guaranteed that there always will be an Euclidean metric defined, which is needed for distance measurements. Hence, instead of distance measurements an estimation based on similarity measurements is used:

$$distance(x, y) = \frac{1}{similarity(x, y)} - 1$$

Similarity functions return 1 for identical data and 0 for absolutely different data (however this may be defined). An interesting challenge would be to learn similarity functions as well.

6.1.2. Growing cell structures

Support points should be well chosen. The aim is to retrieve an approximation of high accuracy with a compact representation. The implemented algorithm is inspired by

[2] The current implementation does not use extrapolation methods yet, since extrapolation is calculation intensive and well established for Euclidean domains, only. Until suited extrapolation techniques are developed, the postulated „good“ teacher has to take care, that important „extreme situations“ will be taught to the robot.

connectionist approaches known as *growing neural gas algorithms* [8].

Adding new support points (neurons) is necessary only when the actual teach-in $t = (s,a)$ was „unexpected“, i.e. the output of the current approximation differs significantly from a. This can be expressed as $sim(approx(s),a) > ActDiffBound$ where *ActDiffBound* is a parameter close to one (e.g. 0.95 as used in our experiments). Furthermore, to keep the number of support points small, a new neuron is inserted only, if its sensor reading differs significantly from all known sensor readings, i.e. if $sim(s_i,s) > SensDiffBound$ for all neurons sp_i.

Otherwise, the closest neuron is just modified to better represent the current action a. Modifying a neuron can be seen as adapting the position of the support point to the new teach-in. This is done by setting the (new) values $sp_i' = (s_i',a_i')$ to the weighted average of the old sp_i (weighted w) and the actual teach-in t (weighted $1-w$)[3]:

$$sp_i' = \left((s_i \cdot (1-w) + w \cdot s), (a_i \cdot (1-w) + w \cdot a)\right)$$

Fig. 6 explains what this means for the one-dimensional case. For continuos functions, sensible chosen learning parameters and statistically distributed measurement errors, it can be shown that the approximation-error converges towards zero.

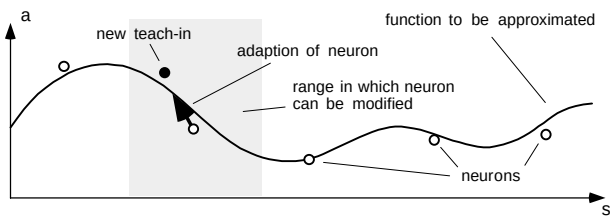

Fig. 6. Adapting neurons to new teach-in

6.1.3. Results with learning reactive tasks

Although a quantitative comparison to other learning approaches is still to be evaluated, it has revealed that learning reactive tasks with the proposed algorithm is very efficient.

Fig. 7 shows a reactively learned door passage within a real environment. After teaching three examples only, *PHOENIX* was able to reproduce the shown trajectories (dotted lines) immediately. The resulting neural net consisted of 19 neurons, representing the task.

[3] The learning rate w indicates, how strong the modification of the previous representation can be. With a constant learning rate, early teach-ins lose weight through each later modification, allowing actions to be overwritten or retrained. On the other hand, the same relevance for every teach-in can be achieved by setting the weight of the k-th teach-in w_k to the reciprocal of k.

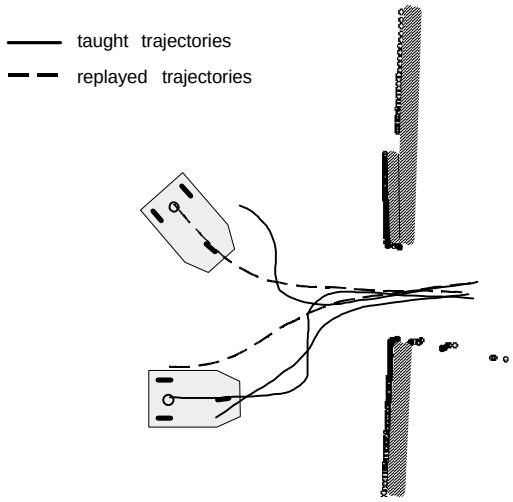

Fig. 7. Reactively learned door passage

Learning was based on the front-mounted LRF with its 180 degree scan grouped into 12 sectors with a limited range of two meters. Similar trajectories can be achieved at other doors (e.g. with a different width) within comparable environments. In cases, where the induction capabilities of the framework are not sufficient, e.g. in order to cope with a door having an orthogonal wall as one door post, these specific cases can simply be taught additionally and the robot will master them. Applying a specific virtual *door-sensor* similar to the *human detector* from Section 3.1 would have revealed even better results.

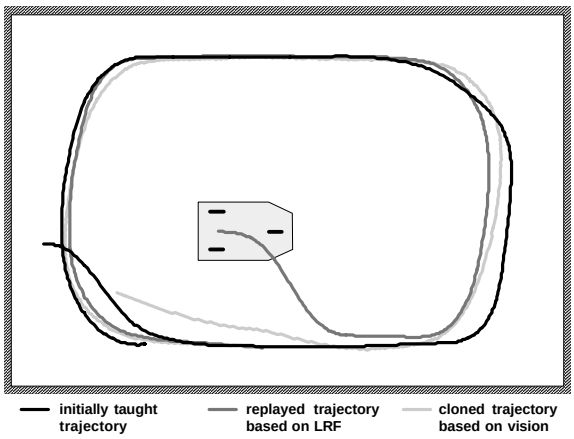

Fig. 8. Taught and cloned wall-following task

A second example of a simple reactive task is given in Fig. 8. It shows the trajectories for a wall following task, which was taught using the laser range finder with the same settings as before. Fig. 8 shows an output trajectory of the resulting neural gas net which was composed of 79 neurons.

This behavior was subsequently cloned by replaying it, while the robot at the same time was learning the stimulus-response pairs by exclusively using a video camera as sensor for the new behavior. The camera observed the environment, consisting of a dark floor and white walls, by pointing 1.5 m in front and slightly to the right-hand side of the robot. Fig. 9 shows a typical perceived video image.

Fig. 9. Video image when following a wall

The 12-dimensional feature vector, the network was trained with, consisted of a single row (512 pixels, taken from the middle of the image), which was compacted to 12 gray-scale values. The resulting neural net had 162 neurons and performed as shown by the light gray line in Fig. 8.

Obviously, the new behavior performs similar to the first one. However, the new trajectory is slightly smother. This results from the input vector's smaller bandwidth and a lower reactivity, resulting from a lack of "extreme" situations while building up the network. This effect is comparable to making analog copies of audio tapes. So in general, the sequence of clones should be kept short, as long as teaching is not followed by special optimization procedures.

6.2. Learning history-dependent behaviors

One of *MOBOCOB's* aims is to learn *history-dependent behaviors*. This can be achieved by using the same representation, which was already used for learning reactive tasks. However, instead of merging all teach-ins (s,a) into a single representation, the learner collects temporal sequences (chains) of (s,a)-pairs. This may lead to several chains describing a single task (Fig. 10). Each node implicitly represents the complete history of perceptions and actions obtained so far.

When applying a learned history-dependent behavior, the robot has to determine which state of *progress* within each chain represents the current situation best. Each node of a chain contains an actuator command a. Thus, depending on the estimated progress we have to interpolate between the commands of a chain when executing a taught behavior. Furthermore, for multiple chains, even interpolation between these chains is necessary.

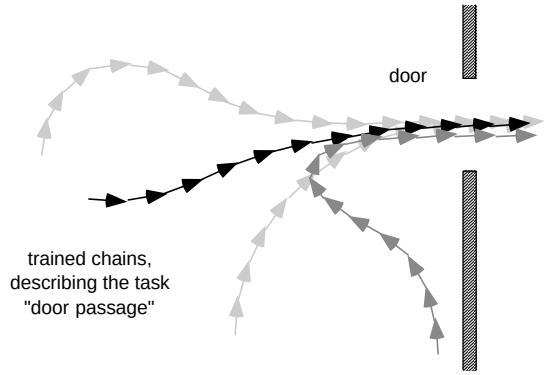

Fig. 10. (s,a)-sequences for passing a door

For behaviors defined by a single chain, progress is totally temporally ordered. When having multiple chains, each chain represents an individual training example which forms a distinct solution of the task. In this context, it might be necessary to decide between *alternatives*, e.g. when teaching „obstacle avoidance" by alternately evading a few times to the left and to the right hand side of an obstacle, respectively (Fig. 11). Since the progress of the individual alternatives is hardly comparable, progress is not totally ordered. Multiple chains, for which progress is comparable (and hence totally ordered) are called *variants* of each other.

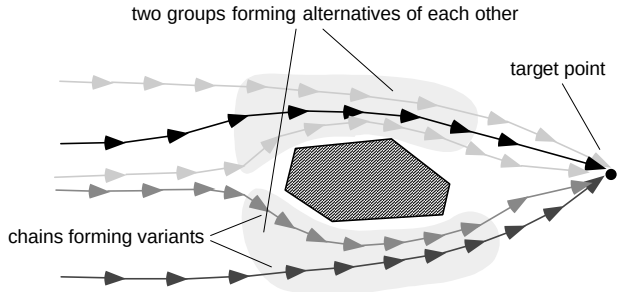

Fig. 11. Taught sequences, defining an obstacle avoidance task

Against this background, we are faced with two major problems: a) Identifying sets of (sub-)sequences which form variants (so called *groups*) and b) evaluating the totally ordered progress within these variants. There are several possible ways for identifying variants or alternatives within a given set of chains, e.g. by the user

manually defining situation based “checkpoints”. We will not go into detail here, but will focus on the second matter, instead.

The most simple case of totally ordered progress is found in single-chain behaviors. This is described in the following section. Thereafter, the method is generalized to groups and multiple chains containing alternatives, which are the most general cases of history-dependent behaviors.

6.2.1. Single-chain behaviors

Due to their ability to cope with dynamic environments and uncertainty, probabilistic approaches have been established for solving various problems in mobile robotics, e.g. for self-localization [4] and speech recognition [3]. They are also well suited for progress estimation within behavior chains.

In general, progress can be regarded as a probability distribution over a chain. This continuos distribution can be approximated by a discrete distribution over all nodes. A node’s value represents the robot’s belief of being in (or close to) a specific state or node. In this way, the nodes not only represent support points for the actuator function but also for the probability distribution. Peaks of the distribution characterize *hypotheses* of possible positions. At the very beginning, the probabilities are either evenly distributed or the ones at the beginning of a chain are slightly emphasized. While executing a behavior, the distribution is periodically updated by applying two rules: a) *shifting* the probabilities along the chain and b) *synchronizing* to the environment. In this context, weighting between synchronizing and shifting is an important parameter. If there is no synchronizing at all, we are restricted to blind behaviors.

Synchronizing is performed by emphasizing the probability of those nodes, for which the current sensor readings are similar to the expected ones (Fig. 13.a).

Defining a general model for shifting is more difficult. Within the set of fixed support points, two subsequent ones may be located far apart from each other, while the maximum of a hypothesis is located somewhere in between. In this case, the hypothesis’ probability is distributed to the adjacent support points. Uncertainty about the exact position of the maximum has got the same representation, hence both cases are indistinguishable. To guarantee a sufficient approximation of the probability distribution, we have to limit the maximum distance between the nodes (comparable to Shannon’s Theorem in signal-theory). One has to trade off between accuracy and compactness of a representation to meet real-time demands. Loss of accuracy is acceptable as long as synchronizing performs sufficiently well in order to compensate the accumulating error.

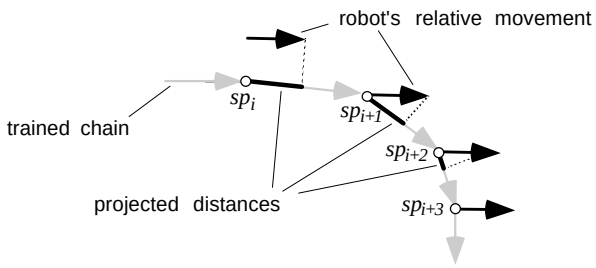

Fig. 12. Projecting the robot’s relative movement to the trajectory between two support points

For the spatial domain, we propose the use of a geometrical heuristic for shifting. This heuristic is based on dead-reckoning and the relative positions of the nodes. The robot’s relative movement within an update interval can be projected onto the trajectory between two subsequent support points as shown in Fig. 12.

The length of the projected distance results in the relative advancement $adv_{rel,i,k}$ between each pair of support points sp_i and sp_{i+k}. By properly selecting k for each sp_i, $adv_{rel,i,k}$ can be limited to the range of [0..1]. For $k = 1$ the probability value p_i of a node sp_i can be updated by the simple rule

$$p_i' = adv_{rel,i-1} \cdot p_{i-1} + (1-adv_{rel,i}) \cdot p_i$$

This corresponds to a linear interpolation of the probability distribution between two adjacent support points. Therefore shifting leads to a loss of accuracy resulting in a blurring of distribution’s peaks (hypotheses). Fig. 13.b shows this effect, which is acceptable as long it can be compensated by synchronizing.

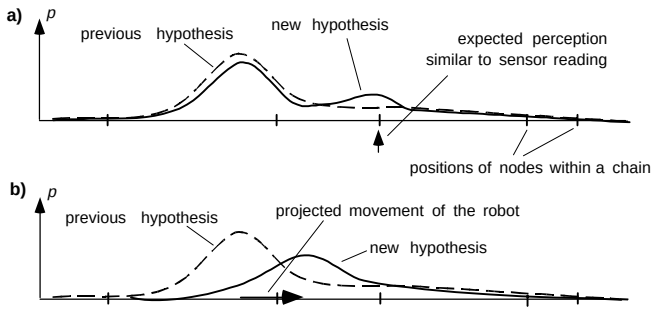

Fig. 13. Updating hypotheses by a) synchronizing and b) shifting

6.2.2. Multiple-chain behaviors

Please recall the example „obstacle avoidance“ from Fig. 11. When interpolating between different variants forming a group, it is reasonable to interpolate between those nodes

only, which represent (almost) equivalent states of progress. Once the nodes representing the actual progress have been selected, calculating the actuator commands by interpolating between these nodes is a purely reactive task. Advancing within the group will select different sets of nodes and therefore different reactive control functions. Hence, a sequence of reactive behaviors can be seen as an instance of a group. A new behavior within this sequence is selected whenever the succeeding nodes represent the state of progress better than the previous ones. Fig. 14 sketches a possible partitioning of a group into individual reactive behaviors while executing a history-dependent obstacle-avoidance task.

Please note, that progress within a group is also totally ordered. The methods for calculating progress within single-chain behaviors can therefore be directly transferred to progress evaluation for groups.

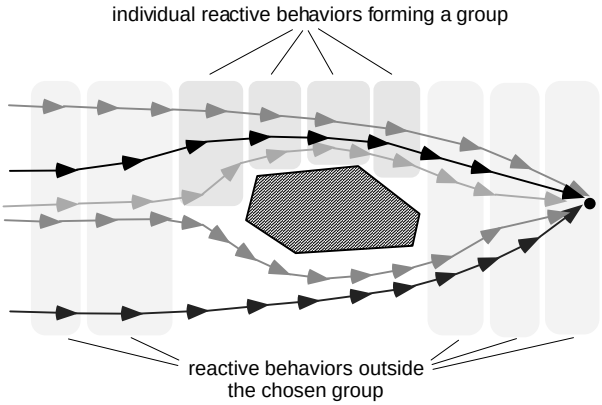

Fig. 14. Sequence of reactive behaviors while executing a history-dependent task.

Conclusion

A common, modularized behavior-based architecture, which is particularly suited for learning-experiments has been presented. The formal classification of behaviors offers opportunities for learning blind, reactive and history-dependent tasks. In any case, teaching can be conducted by a human teacher, another robot or simply by another behavior running on the same robot. Since relevant functions are encapsulated within a generic sensor/actuator-interface, the learning algorithm is independent of both, physical and virtual sensors.

First experiments for learning reactive and history-dependent tasks have performed within the *MOBOCOB*-Project. While learning reactive behaviors using RBF-approximation with growing neural cell structures is very satisfactory, there are still open questions concerning history-dependent tasks. To cope with small relative progress, unevenly distributed nodes, or unknown starting positions, the approach has to be further optimized.

Chances to overcome these problems might be given by using improved approximations for probability distributions (not just linear ones) or by using different sets of support points for representing actuator commands and probability distributions. The *support points* of a distribution could be shifted rather than shifting the distribution itself. However, using a probabilistic approach seems to be an appropriate way to cope with uncertainty of progress estimation. Further work will improve the obtained results and compare them with traditional approaches.

References

[1] R. C. Arkin, Behaviour-based Robotics, MIT Press, Cambridge Massachusetts; 1998

[2] C. Balkenius, Natural Intelligence in Artificial Creatures, Cognitive Studies 37 PhD-Thesis, Lund University, 1995.

[3] A. Bonafonte, X. Ros, J. B. Marino, Explicit Modeling of Duration in HMM, in R. Ayuso, Speech Recognition: New Advances and Trends, in: Proceedings of the NATO Advanced Study Institute on New Advances and Trends in Speech Recognition and Coding, Bubión, Granada, Spain, 1993.

[4] W. Burgard, D. Fox, S Thrun, Active Mobile Robot Localization by Entropy Minimization, in: Proceedings of the 2nd Euromicro Workshop on Advanced Mobile Robots (EUROBOT '97), Brescia, Italy, 1997.

[5] Documentation of the CAROL Project, Department of Computer Science, University of Kaiserslautern, http://ag-vp-www.informatik.uni-kl.de.

[6] J. Donnart, J. Meyer, Learning Reactive and Planning Rules in a Motivationally Autonomous Animat, in: IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Learning Autonomous Robots, 1996.

[7] G. Fricke, Mobile Robot Teaching within a Behavioural Architecture, Diploma Thesis, Department of Computer Science, University of Kaiserslautern, 1999.

[8] B. Fritzke, Wachsende Zellstrukturen - Ein selbstorganisierendes Netzwerkmodell - Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik), Friedrich Alexander Universität Erlangen, Nürnberg 1992.

[9] S. Mahadevan, Machine Learning for Robots: A Comparison of Different Paradigms, in: Proceedings of the IROS Workshop Towards Real Autonomy, Osaka, Japan, 1996.

[10] P. Martin, U. Nehmzow, 'Programming' By Teaching: Neural Network Control in the Manchester Mobile Robot, published in: International Conference on Intelligent Autonomous Vehicles, Helsinki, 1995.

[11] U. Nehmzow: Applications of Robot Training: Clearing, Cleaning, Surveillance, International Workshop on Advanced Robotics and Intelligent Machines, Salford, UK, 1995.

Michael Kasper was born in Würzburg, Germany in 1968. He received his Dipl.-Inform. degree from the University of Kaiserslautern's Department of Computer Science in 1994. Since 1995 he is a research assistant at the Robotics and Process Control Research Group at the University of Kaiserslautern. His research interests include behavior-based control, sensor data processing and robot architectures.

Gernot Fricke was born in Bad Kreuznach, Germany in 1971. He studied computer science at the University of Kaiserslautern with a focus on AI and robotics. 1999 he received his Dipl.-Inform. degree from the Department of Computer Science. Currently he is employee of Realmedia Deutschland GmbH in Munich.

Katja Steuernagel was born in Zeitz, Germany in 1978. She received her intermediate diploma in technical computer science from the University of Mannheim. Since 1998 she is studying Techno-Informatics at the University of Kaiserslautern with a focus on robotics, computer networks and data base systems.

Ewald von Puttkamer, born 1936, received his diploma degree in physics from the University of Freiburg, Germany in 1965 and got his PhD in physics from the same university in 1969 with a thesis on photoion-photoelectron coincidence measurements. 1969 he went to the University of Mainz and 1970 to the then newly founded University of Kaiserslautern. In 1975 he became university professor at the Computer Science Department. Since the early 80s his research interests are in the field of autonomous mobile robots.