A Behavior-Based Architecture for Teaching More than Reactive Behaviors to Mobile Robots

Michael Kasper, Gernot Fricke, Ewald von Puttkamer
University of Kaiserslautern
Department of Computer Science
Postfach 3049, D-67663 Kaiserslautern, Germany
kasper@informatik.uni-kl.de

Abstract

This contribution gives an overview of learning aspects regarding behavior-based control architectures for autonomous mobile robots. We propose a common, modularized architecture, which is particularly suited for learning-experiments. The main focus lies on "Learning from Demonstration" in a spatial domain, which means teaching motor-behaviors by humans or other robots. First results applying RBF-approximation, growing neural cell structures and probabilistic models for progress estimation are presented.

1. Introduction

Behavior-based approaches have been established as a main alternative to conventional robot control in recent years [Arkin98]. Due to their modular architecture, these approaches provide high scalability, while limiting 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), to be truly flexible, should be equipped with learning capabilities, so they can adapt effectively to a dynamic and changing environment. This is especially true for the growing field of service robotics, where non-professionals are intended to operate robots. In this context *Programming by Demonstration,* or from the viewpoint of the robot: *Learning from Demonstration (LFD)*, is an interesting alternative to explicit robot programming for learning new skills (behaviors), improving 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 [Mahadevan96], within behavior-based robotics, research on *LFD* concentrated mainly on learning simple stimulus-response connections, so called *reactive behaviors*. There are only few approaches known which go beyond reactive behaviors (e.g. [Donnart96] describing reinforcement learning of planning rules). Hence, learning from complete temporal sequences of perceptions (rather than from single perceptions) is still an open question [Nehmzow95].

Within the *MOBOCOB*-project (***mo**bile **ro**bo**t** control by **co**ncurrent **b**ehaviors*) we addressed this problem and developed a framework for investigating behavior-based architectures with special respect to learning techniques. The main focus of the still ongoing studies lies on *Learning from Demonstration* of temporal sequences in the spatial domain. After teaching a few good examples by a human teacher, the robot is able to imitate the teacher and to generalize from the given examples.

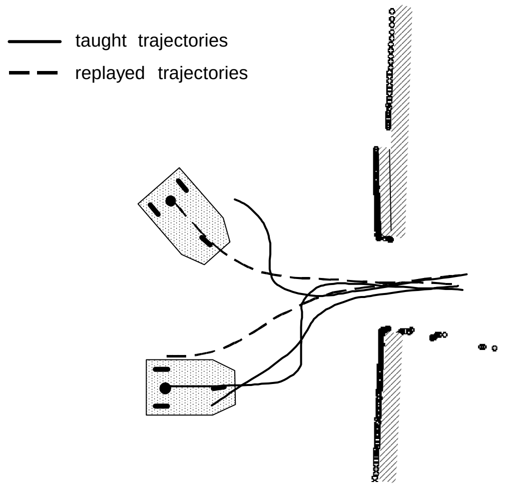


Figure 1. Reactively learned door passage

Figure 1 shows a reactively learned door passage with the robot *PHOENIX* within a real environment. After teaching only three examples, the robot was able to reproduce the shown trajectories (dotted lines). If for more complex tasks the training was not sufficient, i.e. the robot performed not in the way it was expected to, then these cases can simply be taught *additionally*, so the robot will master them.

In the next section we give a formal motivation for behavior-based approaches. The *MOBOCOB*-architecture as a flexible, behavior-based system is introduced in section three. A classification of behaviors based on the formal motivation is defined in section four, followed by a brief survey of different aspects of learning in sections five. 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 [Balkenius95], we will give a more formal motivation for them.

Technically, an AMR consists beside its energy supply of a set of sensors S to perceive the environment, some actuators A to modify the environment (or the robot's position), and a digital control system, which is equipped with some memory Z. From a mathematical point of view, mobile robot control appears to be simple, since all we need is a function f, mapping the sensor input s to some actuator output a with regard to the internal memory state z, as the following equation denotes:

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

Unfortunately, the desired transformation is quite complex. While the dimension of a is typically small (e.g. a tuple (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 - worse than that - the dimension of the internal state space, which is needed to perform a specific task, is not even 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 deviding the problem into piecewise defined sub-tasks. So we get for disjoint domains D_i:

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

Transfering the decision of domain membership into each function f_i 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 is already a behavior-based control architecture. Each f_i denotes an individual behavior and the arbiter corresponds to the union-operator. Since behaviors are commonly 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 four, 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* which was developed within the *CAROL*-project [CAROL99]. *PHOENIX* is a differential drive robot running under the commercial real-time-operating-system QNX, equipped with a laser range finder (Sick LMS-200), ultrasonic sensors and a pan&tilt-video system as its main sensors. Calculations are currently performed with two onboard Pentium PC's, connected through a wireless Ethernet to the research group's LAN.

The *MOBOCOB*-Architecture is intended to be an experimental platform, utilizing a flexible, modular concept for implementing behaviors, sensors, actors and the arbitration unit. All modules share a common interface for access, hence adding new modules such as behaviors or sensors is simple. Figure 2 depicts the structure.

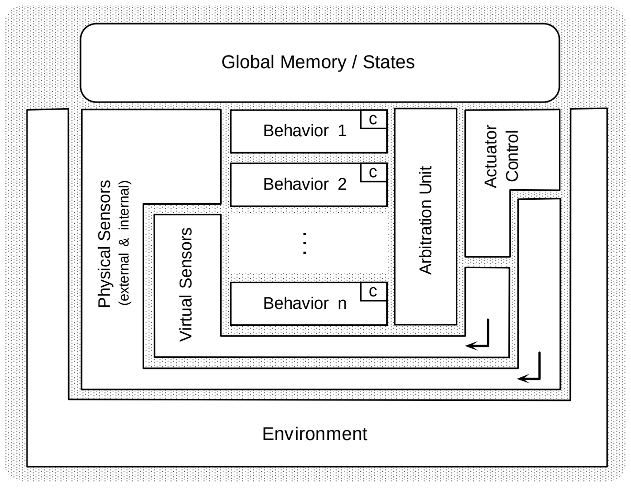


Figure 2. MOBOCOB-Architecture (primary data-flow from left to right)

Physical sensors observe the environment (using external sensors) and read internal sensors, such as odometers. In order to reduce the dimension of sensor input, it is reasonable to do some 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 one or more physical or virtual sensors (compare [Henderson84]). In this way, a higher level of abstraction of perceptions can be achieved, while at the same time reducing the amount of data transferred to the behaviors. Virtual sensors are typically used for sensor fusion, feature detection, object tracking or adding history to physical sensors. They may also be used to watch the output of other modules, such as behaviors or the arbitration unit.

In *MOBOCOB*, a library has been realized that provides access to sensors and actuators, defining standard data-manipulation and -evaluation functions, such as weighted average and similarity [Riemann99]. This way, (behavior) modules can be implemented independently from the actual sensor or actuator type, i.e. different sensor or actuator sets may be used without recoding. This is especially true for learning modules and taught behaviors, since these normally need no direct access to data structures of sensors or actuators. Instead, the learning algorithms rely exclusively on these library functions.

Behaviors are realized as parallel QNX-processes, exchanging data through the same communication channels like the other modules. The reader should recall that in section two, we have transferred the decision of domain membership to the individual behaviors, thus each f_i denotes its competence for a specific situation through a competence-value c 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.

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, there can be competitive behavior selection as well as cooperative behavior fusion, or any combination between. The arbiter is hierarchically organized and also responsible for some kind of behavior sequencing (compare assemblies or engagement-modules in [Balkenius95]). In this context, the arbiter can trigger behaviors or can halt and restart the associated processes.

4. Classification of behaviors

Recall the motivation from section two. Depending on the domain and co-domain of the describing functions, we distinguish several types of behaviors:

- purely reactive motor behaviors,
- blind motor behaviors,
- state dependent motor behaviors and
- hidden, most likely deliberative, behaviors.

Hidden behaviors do not control actuators directly, hence they can be characterized as f: $(s,z) \rightarrow z$. Usually, processing at this level does not only take place in a sub-symbolic manner, but mainly on a symbolic level for planning and reasoning about the environment. Hidden behaviors typically modify the robot's set of targets or its „motivational“ state [Balkenius95]. Obviously they are no good candidates for *Learning from Demonstration*, since there is no way to observe these behaviors from a teacher. However, learning can be applied using unsupervised methods like reinforcement learning (see [Donnart96]).

Purely *reactive (reflexive) behaviors* on the other hand, 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. 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 behaviors*. They do not rely on any (external) sensor information[1] and are described by f: $z \rightarrow (a,z)$.

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 basis for the motivational introduction from above: f: $(s,z) \rightarrow (a,z)$.

In the following we will focus on reactive as well as *history-dependent behaviors*: a subset of state-dependent behaviors, which base on temporal sequences and hence can be represented through cycle-free state graphs.

5. Aspects of learning

Before discussing the general impacts of the above classification on *LFD*, we give a short introduction to the topic. Considering a behavior-based system there are many opportunities for machine learning. Learning techniques can be applied to any components such as behaviors, arbitration, sensor data preprocessing or actuator control. For some tasks, unsupervised learning is promising, for others supervised learning is adequate.

We concentrate on *LFD* as a special kind of supervised learning for behaviors. This technique could also be used for other modules as well, but especially for (motor) behaviors it seems to be straight forward and very promising.

[1] Reading internal sensors can be interpreted as accessing a part of the internal state z. On the other hand, one might see the observation of internal states z as „sensing“ as well, but states which are directly influenced by the observing behavior itself, should be excluded. We will retain the differentiation between (external) sensor data and internal states to achieve a problem-oriented classification of behaviors.

The basic idea is, instead of explicitly programming a behavior, a (human) teacher simply demonstrates a task to the robot by specifying, which sensors are relevant and by driving the robot's actuators a few times. This way, it is easy to teach or adopt a behavior not only for professionals, but also for laymen to new tasks, other environments or different robot hardware (sensors and actuators). Thus *LFD* can be a basis for implementing new behaviors which could be improved later, using unsupervised learning techniques on the robot.

We will see that there is no difference, whether teaching is conducted by a human, another robot or simply by another behavior 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 is to copy a conventionally programmed behavior which can be extended (re-trained) by further supervised learning.

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 at all. 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 [Arkin98], [Martin95]. Teachable tasks include *wall following*, *obstacle avoidance*, *box pushing*, *docking*, *phototaxis* and so on [Nehmzow95]. However, since reactive behaviors just learn simple stimuli-response connections, they are not suited for any history- or state-dependent tasks.

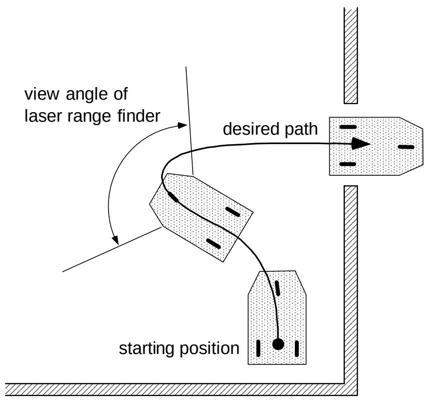
A behavior is called *state-dependent*, whenever a bijective mapping between sensors and actuators is not sufficient to describe the task. For instance, passing a door with a longish robot can not be solved using purely reactive behaviors, if the used sensor covers only a limited area in front of the robot (Figure 3.a). Because of the temporal loss of information about the door position, a reactive behavior would get stuck. Also driving into a parking-box with a vehicle using an Ackerman-Steering, is more than a reactive task, since there exist identical external sensor perceptions, corresponding to completely different actions (reverse direction, see Figure 3.b).

Originally reactive tasks that do not change the environment and have to be repeated a fixed number of times are also state-dependent, because the robot can not conclude from the sensor data, how many repetitions have already been completed. Figure 3.c illustrates this by the example of driving around a totem pole for three times.

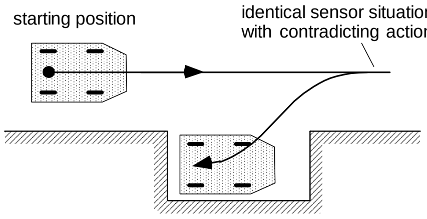
Some tasks that normally do not have a reactive solution may, however, be solved reactively, if the needed memory is „hidden“ in some other components. For example, the door-passage problem could be solved reactively, if instead of the limited physical sensor a virtual 360°-sensor, based on a grid map, 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. And as they do not provide a general history model, one can easily think of problems, where their history representation is not sufficient.

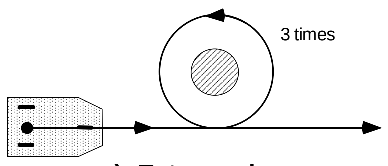
A more universal concept is to use sequences of reactive behaviors [Balkenius95]. They correspond to *history-dependent behaviors* introduced in section four, which are sufficient to solve a large set of robot navigation tasks, such as the ones mentioned above.



a) Door passage



b) Parking box



c) Totem-pole

Figure 3. Non-reactive robot tasks

There are no approaches known for applying *LFD* to history-dependent behaviors in mobile robotics, hence *MOBOCOB* will focus this topic. From the viewpoint of *LFD*, it is desirable that already individual behaviors have some sequencing capabilities, so that teaching involves a single behavior only, rather than the 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 our case, this is a joystick-module driving the motors of the robot via a (v,ω)-interface. The competence-values for a behavior can be taught through a force-sensor applied to 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 grouped, instantiating a new virtual sensor. 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 sensor/actuator-libraries. After teaching a couple of good examples, the robot is able to imitate the teacher and to generalize from the given examples.

6.1 Learning reactive behaviors

What makes a task reactive, is 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, so the function can be regarded as inter- or extrapolation between these points.

We decided to use *Radial Base Function (RBF) approximation* with growing neural cell structures to represent f (and thus the behavior), since this technique can cope with non-linearities and is 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. These pairs are used to derive support points represented by neurons. Each neuron marks the center of a Radial Base Function rbf_i, used to interpolate between the support points.[2]

6.1.1 RBF-approximation

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

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

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

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, defined as the distance (from the center) where rbf_i falls under 0.5. In areas in which the approximated function changes frequently, more support points will be needed than in other „smooth“ areas. By appropriately setting σ, one can assure that the areas 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-libraries define similarity-measurements exclusively, since it can not be guaranteed that there will always be an euclidian metric defined. Hence instead of distance measurements an estimation based on similarity is used:

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

Similarity functions return a value of one for identical data and zero for absolutely different data (however this may be defined).

6.1.2 Growing cell structures

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

[2] The current implementation does not use extrapolation methods yet, since extrapolation is calculation intensive and well established for euclidian 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.

inspired by connectionist approaches known as *growing neural gas algorithms* [Fritzke92], [Zimmer95].

Adding new support points (neurons) is only necessary when the actual teach-in $t = (s,a)$ was „unexpected", i.e. the output o 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 by adapting its position to the new tech-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]:

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

Figure 4 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.

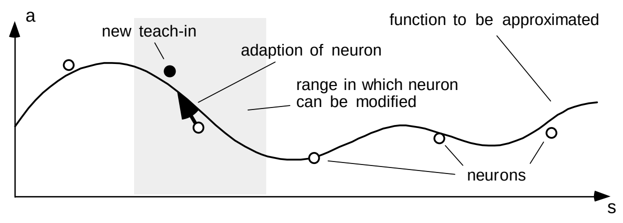

a
new teach-in
adaption of neuron
function to be approximated
range in which neuron can be modified
neurons
s

Figure 4. Adapting neurons to new teach-in

An example of a reactively learned door passage, was already given in the introduction (Figure 1), which lead to 19 neurons, representing the task. Learning was based on the front-mounted laser range finder with distance readings grouped to 12 sectors.

6.2 Learning history-dependent behaviors

MOBOCOB is able to learn *history-dependent behaviors*. Basically we use the same representation as for reactive tasks, but instead of merging all teach-ins (s,a) into one single representation, the learner collects temporal sequences (chains) of (s,a)-pairs. This may lead to several chains describing a single task (Figure 5). Each node represents implicitly the complete history of perceptions and actions up to this position.

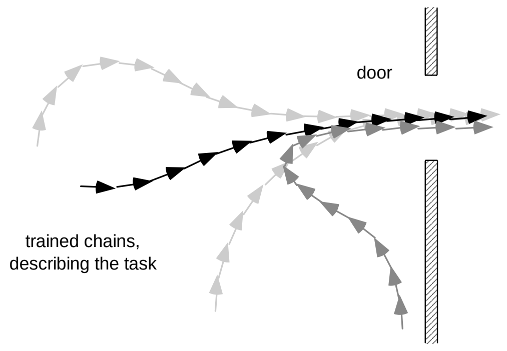


Figure 5. (s,a)-sequences for passing a door

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

For behaviors, defined by a single chain, progress is totally (temporally) ordered. Regarding multiple chains, each chain represents an individual training example. These may form distinct solutions for the same task. It might be necessary to decide between alternatives, e.g. when teaching „obstacle avoidance" and first evading to the left, later to the right side a few times (Figure 6). In this case, progress is not comparable between the alternatives (the upper and lower trail), hence progress is not totally ordered. Multiple chains, for which progress is comparable (and totally ordered) are called *variants* of each other.

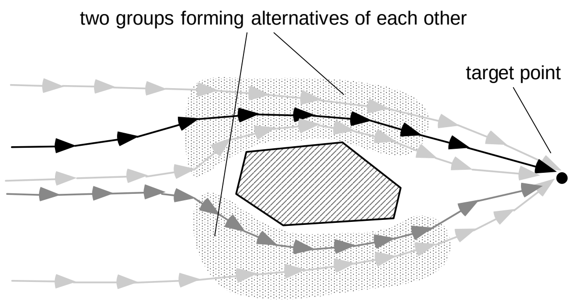


Figure 6. Taught sequences, defining an obstacle avoidance task

[3] The learning rate w indicates how much a previous representation can be modified. With a constant w, 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.

According to the above, we have to face two major problems: a) Identifying sets of (sub-)sequences forming variants (so called *groups*) and b) evaluating the totally ordered progress within these variants. There are several ways to identify variants or alternatives within a set of chains, we will not discuss here. Instead we focus on the second matter and present possible solutions.

The most simple case of a totally ordered progress is found in single-chain behaviors, the following sub-section deals with. Thereafter the subject is generalized to groups and multiple chains containing alternatives, which are the most general case 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 concerning mobile robotics such as self-localization [Burgard97] or speech recognition [Bonafonte93]. They also seem to be 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 one over all nodes. It's values represent the robot's belief of being in (or close to) a specific state or node. This way, the nodes represent not only support points for the actuator function but also for the probability distribution. Peaks of the distribution characterize *hypotheses* of possible positions. At the beginning, the probabilities will be evenly distributed or the ones at the beginning of a chain may be slightly emphasized. While executing a behavior, the distribution is periodically updated by two rules: a) *shifting* (*transposing*) the probabilities along the chain and b) *synchronizing* to the environment. Weighting between synchronizing and transposing 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 reading is similar to the expected one (Figure 8.a). Defining a general model for transposing is more difficult. Two subsequent support points may be located far apart from each other, while the maximum of a hypothesis lies in the middle between them. The hypothesis' probability is distributed to the adjacent support points. Uncertainty about the exact position of the maximum uses 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 (compare Shannon's Theorem). A trade-off between accuracy and compactness of representation has to be made to meet real-time demands. Loss of accuracy is acceptable as long as synchronizing is sufficient to compensate the cumulating error.

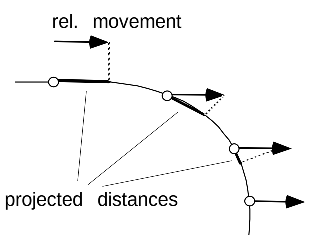


Figure 7. Projecting the relative movement to the trajectory between two support points

For the spatial domain, we propose the use of a geometric heuristic for transposing, based on dead-reckoning and the relative positions of the nodes. We can project the robots relative movement to the trajectory between two subsequent support points as shown in Figure 7.

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

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

This corresponds to only linear interpolation of the probability distribution between the support points, and therefore transposing leads to a loss of accuracy resulting in blurring the peaks (hypotheses) of a distribution. Figure 8.b shows this effect, which is acceptable as long as it can be compensated by synchronizing (see above).

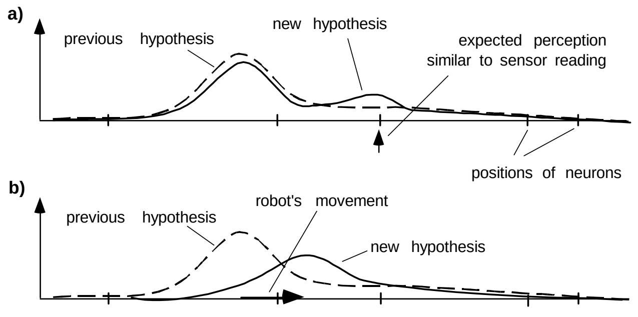


Figure 8. Updating hypotheses by a) synchronizing and b) shifting

6.2.2 Multiple-chain behaviors

Recall the example „obstacle avoidance“ (Figure 6). When interpolating between different variants which form a group, it is reasonable to interpolate only between those nodes that represent (almost) equivalent progress. Once the nodes representing the progress have been selected, action evaluation is reactive (by interpolating between these nodes), since the state of progress is represented by the selection. Advancing within the group will select different sets of nodes and therefore different reactive control functions. Hence, a group can be seen as defining a sequence of reactive behaviors. The behavior changes whenever other nodes represent the state of progress better.

The methods of evaluating progress for single-chain behaviors can therefore be transferred to progress evaluation for groups, as progress within a group is also totally ordered. Figure 9 sketches the combination of several alternative groups to a single history-dependent behavior.

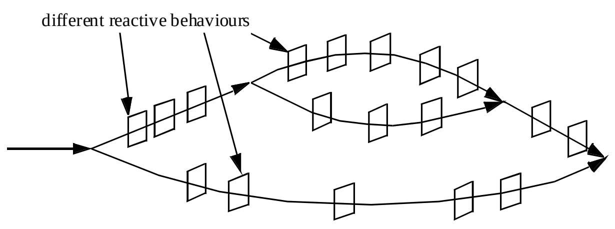


Figure 9. Sequences of reactive behaviors forming a history-dependent behavior.

Conclusion

We have motivated a common, modularized behavior-based architecture, which is particularly suited for learning-experiments. The formal classification of behaviors showed opportunities for learning blind, reactive and history-dependent tasks. In each case, teaching can be conducted by a human, another robot or simply by another behavior on the same robot (online or offline cloning). The learning algorithm itself is independent of the employed sensors by encapsulating relevant functions within sensor/actuator-libraries.

First experiments for learning reactive and history-dependent tasks have been carried out 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 the use of better approximations for probability distributions (not just linear) or the use of different sets of support points for representing actuator commands and probability distributions, so that the support points of a distribution could be transposed rather than the distribution itself. However, using a probabilistic approach seems to be an appropriate way to cope with uncertainty of progress estimation. Further works will have to improve the obtained results and compare them to traditional approaches.

References

[Arkin98] R. C. Arkin: *Behaviour-based Robotics*, MIT Press, Cambridge Massachusetts; 1998

[Balkenius95] C. Balkenius: *Natural Intelligence in Artificial Creatures*, Lund University Cognitive Studies 37, 1995

[Bonafonte93] A. Bonafonte, X. Ros, J.B.Marino: *Explicit Modeling of Duration in HMM* in R. Ayuso: *Speech Recognition: New Advances and Trends*, proceedings of the NATO Advanced Study Institute on New Advances and Trends in Speech Recognition and Coding, held in Bubión, Granada, Spain, June 28 - July 10, 1993

[Burgard97] W. Burgard, D. Fox, S Thrun: *Active Mobile Robot Localization by Entropy Minimization*, 2nd Euromicro Workshop on Advanced Mobile Robots (EUROBOT '97) Brescia, Italy; October 22-24, 1997

[CAROL99] Documentation of the CAROL Project, Department of Computer Science, University of Kaiserslautern, 1999; http://ag-vp-www.informatik.uni-kl.de

[Donnart96] 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

[Fricke99] G. Fricke: *Mobile Robot Teaching within a Behavioural Architecture*, Diploma Thesis, Department of Computer Science, University of Kaiserslautern, 1999

[Fritzke92] B. Fritzke: *Wachsende Zellstrukturen - Ein selbstorganisierendes Netzwerkmodell* - Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik), Friedrich Alexander Universität Erlangen, Nürnberg 1992

[Henderson84] T. Henderson, E. Shilcrat, *Logical Sensor Systems*, Journal of Robotic Systems 1(2), pp. 169-193, 1984

[Mahadevan96] S. Mahadevan: *Machine Learning for Robots: A Comparison of Different Paradigms*, IROS Workshop Towards Real Autonomy, Osaka, Japan, 1996

[Martin95] P. Martin and U. Nehmzow: *'Programming' By Teaching: Neural Network Control In The Manchester Mobile Robot* published at Intern. Conf. Intelligent Autonomous Vehicles 1995, Helsinki.

[Nehmzow95] U. Nehmzow: *Applications of Robot Training: Clearing, Cleaning, Surveillance*, International Workshop on Advanced Robotics and Intelligent Machines, Salford, UK, 5.-6.4.1995

[Riemann99] F. Riemann: *Entwicklung eines generischen Sensorkonzeptes für einen Autonomen Mobilen Roboter*, Diploma Thesis, Department of Computer Science, University of Kaiserslautern, 1999

[Zimmer95] U. R. Zimmer: *Adaptive Approaches to Basic Mobile Robot Tasks*, Ph.D. Thesis, Department of Computer Science, University of Kaiserslautern, 1995