Title: Incremental Robot Shaping.

Authors: Joseba Urzelai, Dario Floreano, Marco Dorigo and Marco Colombetti.

Affiliation: Laboratory of Microcomputing (LAMI), Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland (J.U. and D.F.)
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (M.D.)
Dipartimento Elettronica e Informazione, Politecnico di Milano, Milano, Italy (M.C.)

Corresponding Author: Joseba Urzelai, Laboratory of Microcomputing (LAMI), Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne, Switzerland. Phone: +41 21 693 6696. Fax: +41 21 693 5263. E-mail: Joseba.Urzelai@epfl.ch

Running Title: Incremental Robot Shaping.

Keywords: Robot Shaping, Incremental Evolution, Behavior Based Robotics, Modular Architectures, Evolution and Learning, Reinforcement Learning, Robot Navigation.

Abstract: We propose a modular architecture for autonomous robots which allows for the implementation of basic behavioral modules by both programming and training, and accommodates for an evolutionary development of the interconnections among modules. This architecture can implement highly complex controllers and allows for incremental shaping of the robot behavior. Our proposal is exemplified and evaluated experimentally through a number of mobile robotic tasks involving exploration, battery recharging and object manipulation.

Incremental Robot Shaping*

Joseba Urzelai[1], Dario Floreano[1],

Marco Dorigo[2] and Marco Colombetti[3]

[1]Laboratory of Microcomputing (LAMI),
Swiss Federal Institute of Technology (EPFL),
CH-1015 Lausanne, Switzerland

[2]IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

[3]Dipartimento Elettronica e Informazione,
Politecnico di Milano, Milano, Italy

1 Robot Shaping

Modern approaches to behavior engineering of autonomous robots have stressed the importance of modular and distributed architectures composed of simple and interconnected elements (Brooks, 1990; Dorigo & Schnepf, 1991, 1993; Dorigo & Colombetti, 1994) where each component has full or partial access to sensory data and can affect the actions taken by the robot. Distributed modular control has several potential advantages: it is an open system, it is intrinsically robust to local failures, and it is suitable for gradual "shaping", that is, incremental training of independent behavioral competencies (Dorigo & Colombetti, 1998).

With modular architectures, relatively complex behavioral patterns can be built bottom-up from a set of simple basic behaviors. Two aspects are of key importance for the success of such an approach: (i) the set of basic behaviors,

*A preliminary version of this paper was published in the proceedings of the Genetic Programming Conference, Madison, Wisconsin, 1998.

and (ii) the mutual interactions among them. As regards the first point, the choice of which behavioral modules to assume as basic is typically made by a human designer. Once such a choice has been made, however, it is often feasible to use machine learning techniques as an aid to the implementation of the basic modules. Machine learning methods can also be exploited to develop the interactions among modules.

A problem which is often difficult to solve is finding the optimal balance between human design and the use of machine learning techniques. To find a reasonable solution, one should always have a clear idea of why learning is used in a specific application (Floreano, 1997). In fact, a robot's ability to learn its own behavior can be exploited to: (i) cut development costs by relieving human designers of part of their burden; (ii) bypass the practical impossibility to completely describe the robot's environment and task a priori; (iii) endow the robot with capacities for self-adaptation, which may play an essential role in both optimizing behavior with respect to some performance measure, and in coping with unforeseen changes in the environment.

However, to exploit machine learning techniques at their best, the whole development activity has to be conceived and organized in the appropriate way. In the course of our research, we have developed a methodology to assist an engineer in the process of designing and training an autonomous robot. The *Behavior Analysis and Training* methodology (BAT; Dorigo and Colombetti, 1998) is a first example of a behavior engineering methodology involving the analysis of behavior, the integration of machine learning techniques with other aspects of robot design, and the independent assessment of the learning activity and of the resulting global behavior. The BAT methodology proposes a rational organization of the main phases of robot development based on learning techniques, that is: application description and behavior requirements, behavior analysis, specification, design and implementation of the physical robot, training, and behavior assessment.

The two phases that more sharply distinguish BAT from classical methodologies for the development of software products are behavior analysis and training.

Behavior analysis includes the specification of the basic behaviors and of their relationships. Training involves exploiting the robot's learning capacities in order to develop the required behavior.

Among the possible approaches to machine learning, reinforcement learning seems to be particularly fit for developing robotic behaviors. The reason is that the only information necessary to train a reinforcement learning agent is a scalar evaluation of its behavior, which is relatively easy to produce as compared to supervised learning requiring step-by-step feedback based on an error value generated from a training set. As it has been argued elsewhere (Dorigo and Colombetti, 1998), the Reinforcement Program (i.e., the computer program that produces the evaluation of behavior) can be regarded as a high-level specification of the desired behavior. From this standpoint, learning translates such specifications into a working controller. Such translation is situated, in the sense that it is sensitive to aspects of the actual robot-environment interactions which may not be modelled in any part of the system, and of which the robot designer might even be unaware.

It should be pointed out that robot controllers based on a complex interaction of many basic components may be very difficult to understand, and particularly so if a substantial amount of training has been carried out, due to the sensitivity of learning to unmodelled aspects of the world. A lot of experimental activity may be necessary to grasp their behavior to a satisfactory degree. This fact makes the final assessment of the robot's behavior a more fundamental activity than traditional system testing. Indeed, behavior assessment has to show not only that the system learned what it has been taught, but also that it has been taught the right thing which, in practical applications, might be far from trivial.

As we have already pointed out, a major problem is in deciding what should be explicitly designed and what should be left for the robot to learn from experience. In several practical situations, it is neither necessary nor advisable to train a control system completely from scratch. On the one hand, learning not only takes long time, but it also does not guarantee convergence if the parameter search space is very large. On the other hand, there might be some aspects of the behavior that could be easily pre-programmed exploiting available knowledge about the task constraints. A modular architecture can easily accommodate both adaptive and pre-programmed modules.

Adaptation in modular architectures can take place at two different levels: within a module and between modules. The first level is often concerned with acquisition of a new specific competence and/or with fine tuning of some parameters of the module, such as threshold adjustments, to accommodate minor changes in the environment or robotic platform. This aspect of adaptation is equivalent to local search in a restricted parameter space and is well-suited for learning paradigms such as, for example, reinforcement learning. The second level of adaptation is concerned with coordination of all modules and/or with acquisition and integration of new modules to cope with major environmental changes and task constraints. This aspect of adaptation is analogous to a global search over a coarser space composed of a finite set of behavioral competencies, and seems well-suited for an evolutionary approach (Brooks, 1992). The evolutionary method chosen should allow for the addition and integration at later stages of development of new modules necessary to cope with modifications of the robot shell and/or new behavioral tasks.

Besides giving a contribution to the rapidly growing field of evolutionary robotics, we hope our paper may play a part in the current trend of artificial intelligence and robotics, which takes a deeper understanding of the biology of intelligence to be an important prerequisite for success. While we do not aim at modelling biological mechanisms in any strict sense, our work is biologically inspired under several respects. In general, we think that biologically inspired models can play a part at different levels of system development, namely:

- at the functional level, where the functions of a system are considered with respect to its environment;
- at the architectural level, where the internal structure of the system is designed in relationship with its functions;

- at the implementation level, where is considered the problem of concretely realizing an architecture.

At the functional level, we have been influenced by biology in considering behavior (viewed as the interaction between the system and its environment) as the target of our work, and in taking adaptiveness as a basic feature for any system which has to interact with the physical world. In agreement with the BAT methodology, such a standpoint has direct consequences at the architectural level: first, the centrality of behavior lead us to adopt a behavior-based architecture; and second, the need for adaptiveness gives great importance to the system's learning capabilities and to the methodology to be followed for training. More precisely, the learning methods we have adopted (reinforcement learning and evolutionary computation) are both inspired by biology, and the shaping methodology used in training is loosely derived from experimental psychology. In fact, the way in which a robot is trained through reinforcements is remarkably similar to the procedure that psychologists use to train a laboratory animal to perform a predefined behavioral response (Skinner, 1938): in both cases, the final behavior is gradually approached by shaping the subject's spontaneous behavior through rewards and punishments. In this work, we have not pursued the biological analogy further down to the implementation level. We believe however that the lower is the level of analysis, the stronger become the constraints posed by the hardware used for implementation, which, in the case of artificial systems, is remarkably different from that of biological organisms. Therefore, bringing biological inspiration down to the implementation level raises a whole new set of issues and challenges that go beyond the scope of this work.

Approaches related to shaping methodologies can be found in the work by (Nehmzov & McGonigle, 1994), who implemented a pattern associator using supervised learning to change on-line the weight strengths of a neural network that controlled a mobile robot, and in the work by (Touretzky & Saksida, 1996, 1997), who presented a model of an operant conditioning technique in which behaviors were progressively combined in order to yield more complicated action sequences. Multi-level adaptation in modular architectures has been addressed by (Jordan & Jacobs, 1994) work on mixture of experts and, in the context of autonomous robotics, by (Tani & Nolfi, 1998). Nolfi also indicated that emergent modularity might outperform hand-designed and pre-defined modularity (Nolfi, 1997b, 1997a). Asada worked on emergent behavior acquisition in modular architectures (Asada, Noda, Tawaratsumida & Hosoda, 1996).

This paper is organized as follows. In Section 2 we briefly describe our approach and methodology. In Section 3 we introduce our modular architecture. In Section 4 we present a number of paradigmatic experiments and discuss their results. Finally, in Section 5 we draw some conclusions.

2 The BAT Methodology

In this paper we describe the implementation on a real robot of an open modular architecture that can be incrementally shaped via evolutionary and learning mechanisms while the robot interacts with its environment. The three main objectives of the proposed method are:

- the architecture must be capable of integrating adaptive modules with pre-programmed behaviors;
- the system must allow for incremental and autonomous construction of a suitable architecture, as demanded by the shaping policy or by major changes in the task constraints;
- individual modules must be capable of quickly re-adapting themselves to local changes without requiring a full re-engineering of the whole architecture.

As we have already remarked, the development of a modular robot controller by design and training presupposes a suitable methodology, that is, a suitable ordering of critical decisions. Here we shall briefly sketch the ordering of decisions according to the BAT methodology (Colombetti, Dorigo, & Borghi, 1996; Dorigo & Colombetti, 1998). The first step in the development process is a sufficiently detailed description of the robot, the environment, and the target behavior, that is, the task that the robot is intended to carry out. It should be kept in mind that behavior is not a process produced by the robot alone; rather, by behavior we mean the interaction between the robot's "body" (i.e., its physical shell) and the environment. For example, the very notion of obstacle avoidance will be materialized by different behaviors in an office setting and in an environment containing clusters of small, randomly placed obstacles.

The second step in the process of developing a modular controller is an analysis of the global target behavior, with the goal of breaking it down to a set of simpler behaviors. The idea is that the robot's global behavior should emerge from the interaction of basic behaviors, which in turn are not decomposed into simpler elements but are generated from the interaction between the sensorimotor mapping implemented by a single module and the environment in which the robot is situated. In fact, the very notion of basic behavior is rather fuzzy. In one context, grasping an object may be a basic action, while in another it may be necessary to break the grasping behavior into a complex sequence of simpler behaviors, each deserving a separate implementation. Only experience with robot development can tell a designer where to stop the analysis.

Often, behavior analysis makes it clear that basic behaviors need special sensors to gather the necessary input information from the environment, or special effectors to act on the environment. The third step in the development process therefore includes the design, implementation and testing of necessary additions to the robot's sensorimotor interface.

The fourth step includes the implementation of the behavioral modules and of their interconnections, which may be fully developed by hand or acquired through some machine learning technique. In the experiments described later on, some behavioral modules are directly programmed and some are learned, while the interconnections are always learned.

The use of learning techniques to develop robotic behaviors is far from trivial. It is a main point of the BAT methodology that the ability to learn is best exploited if an appropriate training activity is carried out. From the point of view of the trainer, the details of the learning process are usually irrelevant. However, the trainer must be aware of the kind of information that must be provided during training to guide the learning process toward the target behavior. Moreover, when developing a modular architecture one has to decide in advance how to "shape" the robot's behavior, choosing between a modular and a holistic shaping strategy (i.e., training each basic behavior separately, and then their interconnections, or training the whole target behavior in one shot). In the experiments presented later on, we systematically adopt a holistic shaping strategy. As regards the learning mechanism, we exploit reinforcement learning techniques. This involves designing and implementing a suitable reinforcement program, that is, a software module which generates the reinforcements used by the learning algorithm to produce the desired behavioral modules and interconnections.

The last step in the process of robot development is the final assessment of behavior. When robot training is adopted, there is always the problem of proving that the behavior obtained satisfies the initial requirements. When this is not the case, there may be two different reasons: either the robot has not learned what it has been taught; or the robot has learned correctly, but it has not been taught the right thing. Therefore, behavior assessment must clearly distinguish between the effectiveness of the learning process and the correctness of the reinforcement program.

3 The Modular Architecture

The control architecture employed is composed of a set of fully interconnected modules.

———————————

INSERT FIGURE 1 ABOUT HERE

Each module (figure 1) has an input and an output. The input comes from the sensors of the robot. The output consists of two messages: an activation level that indicates whether the module wants to affect the robot current action and an output vector which consists of a motor command. At each time step, a winner-take-all process occurs among modules. The module with the highest activation level wins the competition and is allowed to access the motor resources and control the robot for a short time slice.

The internal structure of modules is based on two components: an *activation network* and a *behavior generator*. The activation network decides the activation level of the module by combining current sensory information with the weighted sum of activation signals coming from other modules. The resulting activation level, besides being used by the winner-take-all process, is also sent out to the other modules in the architecture. Connections among modules can have excitatory or inhibitory values. In our experiments the activation network is implemented as a feedforward neural network. The behavior generator can be a pre-programmed behavior, a neural network (as in figure 1), a classifier system (Booker, Goldberg, & Holland, 1989), or any other structure suitable for generating motor commands and other behavioral decisions in response to sensory inputs.

Optionally, modules incorporate also a local learning algorithm and a *reinforcement program*. The learning algorithm can adapt both the parameters of the activation network and of the behavior generator using a reinforcement program defined by the engineer. The reinforcement program is also used to generate an indication of the performance level of the module. Learning is automatically enabled whenever the performance of the module falls below a predefined threshold (e.g., 90% positive reinforcement signals). In the implementation described below the reinforcement program used immediate reinforcement signals based on strictly local (to the module) sensory information. The learning algorithm is a modified version of the complementary reinforcement back-propagation algorithm (CRBP) (Ackley & Littman, 1990). For each input pattern μ, it propagates the input vector I^μ through the network to produce a real-valued output vector S^μ ($s_i \in [0, 1]$) on the output layer. Each output value is interpreted as the probability that an associated random bit takes on the value 1. From these probabilities a binary output vector O^μ is stochastically produced. For each input pattern I^μ, if the action corresponding to the current O^μ is rewarded by the reinforcement program, the synaptic weights are modified with the generalized delta rule (Rumelhart, Hinton, & Williams, 1986) by back-propagating the discrepancy between the binary motor commands and the real-valued output vector, $(O^\mu - S^\mu)$. This amounts to increasing the probability of generating the same motor command O^μ for that input pattern I^μ. On the other hand, if the action generates a negative signal, the synaptic weights are changed in the opposite direction on the basis of $((1 - O^\mu) - S^\mu))$. The assumption on which the CRBP algorithm is implicitly based is that when the O^μ output corresponds to a punished action, then $(1 - O^\mu)$ corresponds to the correct action for the same input pattern I^μ. In this way rewarded outputs will be more likely to occur again and punished outputs will tend to produce the complement output vector in similar situations (Meeden, 1996).

The pattern of connectivity among the modules and their individual activation networks are encoded on a binary string and evolved by a genetic algorithm (figure 2). Evolution is incremental and operates on variable-length genotypes. Initially, a set of basic modules are defined by the engineer on the basis of available knowledge about the task requirements and the characteristics of the robot shell. The genetic string encodes the synaptic strengths of the activation networks of all available modules and the values of inter-module connections. Each value is encoded on the genetic string as a 4-bit integer number and is normalized in the continuous range [0,1] before decoding it into the corresponding weight value.

INSERT FIGURE 2 ABOUT HERE

An initial population of such controllers is evolved until an individual is generated that satisfies the task criteria. Individual modules with learning abilities can be separately trained before evolution and/or during evolution, depending on the task constraints. If the task constraints change, or if new hardware modules are added to the robot, it is possible to define new modules and to increment the genotype length by including the new activation networks and all connections to previous modules. However, old parts of the genotype are masked so that they cannot be affected by the crossover and mutation operators. Incremental evolution offers at least two advantages. It allows the engineer to modify parts of the task definition, environment, and/or robot configuration without restarting the whole evolutionary process, and it makes possible to evolve behaviors which otherwise would not be evolvable (as it has been shown by (Floreano, 1993; Nolfi & Floreano, 1998) in experiments in which environment complexity was gradually increased, and by (Harvey, Husbands, & Cliff, 1994) who experimented with a varying fitness function). In this article we focus on the former aspect of incremental evolution.

4 Evolution and Shaping

In this section we apply the BAT methodology (Colombetti et al., 1996; Dorigo & Colombetti, 1998) described in the previous section 2. In particular, we run an experiment aimed at demonstrating two aspects of incremental robot shaping: incremental evolution to accommodate hardware and task modifications, and automatic local adaptation of an individual module to changed environmental conditions.

4.1 Robot, environment, task

The robot employed in this experiment was the miniature mobile robot Khepera (Mondada, Franzi, & Ienne, 1993) which has a circular shape, a diameter of 55 mm, a height of 30 mm, and a weight of 70 g (figure 3). Khepera is supported by two wheels (and two small Teflon balls) whose speeds and rotation directions can be independently controlled. Eight infrared proximity sensors, six positioned on the front side and two on the back side, return continuous values between 0 and 1 proportional to the distance of an object from the sensor. Additionally, each sensor can measure the ambient light. The robot is also equipped with a microcontroller and four rechargeable batteries.

INSERT FIGURE 3 ABOUT HERE

The environment was a rectangular arena (40 cm by 60 cm) whose walls were covered with white paper. A 20 cm long metallic bar for battery charge was attached to one side of the arena and a 20-watt light bulb was positioned above it (figure 3). The metallic bar could be used for recharging the robot batteries by using special contacts plugged on the robot.

The desired task was that of developing a controller capable of moving the robot around the arena as much and as long as possible, avoiding obstacles, and periodically recharging the batteries at the recharging station.

4.2 Behavioral decomposition and module allocation

The global task can be decomposed into four simple behaviors: wander, obstacle avoidance, light following, and battery recharge. Four modules were accordingly allocated, as described below.

Wander Used to move the robot around the environment. The behavior generator was a programmed simple straight motion (whatever the sensor activation is). The input consisted of information coming from the proximity sensors, used by the activation network in order to decide whether the wandering module was activated or not. The output commanded the robot's wheels.

Obstacle-Avoidance The behavior generator was an adaptive neural network mapping sensor activations into one of four possible motor actions[1]. A reinforcement program punished increased sensor activations and rewarded decreased sensor activations. The learning algorithm was CRBP described in section 3. The input consisted of information coming from the proximity sensors and the output commanded the robot's wheels.

Light-Following The behavior generator moved the robot towards light sources using the direction of the vector resulting from the activity of all ambient light sensors. It received ambient-light sensor values and the battery level as inputs, and the output commanded the robot's wheels. No learning mechanism was required because there was only one light source and sufficient gradient information in the environment.

Recharge The behavior generator froze every motor activity until the battery charge indicated full. The module was a programmed wait action that simulated the battery charging operation. It received front proximity sensor values[2] and the battery charge indicator as inputs, and commanded all available motors.

Individual activation networks were genetically evolved and no other local learning mechanism affected them during evolution. All modules were fully interconnected and encoded on binary strings as described in section 3 above.

[1] Discrete actions corresponding to motor speeds are given by two output neurons, each one corresponding to a motor. Each output neuron can have a positive or a negative value, generating in all four possible movements: go forward, turn right, turn left, move backward.

[2] The proximity sensors were used by the activation network in order to detect the battery charging point and activate the module.

4.3 Training

Khepera was attached via a serial port to a Sun SparcStation by means of a lightweight aerial cable and rotating contacts. The genetic algorithm and control architecture were run on the workstation and the serial cable was used to read sensor activations and send motor commands every 100 ms. This solution allowed us to keep track of several data during training for later analysis.

Instead of using the real batteries available on the Khepera (which last approximately 30 minutes and require additional 40 minutes to recharge), a virtual battery lasting 50 seconds and a fast virtual recharger (taking 5 seconds to recharge whenever the robot touched the metallic bar in recharging mode) were simulated during training. Electric power was provided through the aerial cable.[3]

The only module with local learning capabilities was *obstacle avoidance*. Local learning in the *obstacle avoidance* module was always active for all individuals. For each action, the reinforcement program provided a negative reinforcement (R=-1) if the activation of the proximity sensors increased, or a positive reinforcement (R=1) if the action of the proximity sensors decreased.

An initial population of 100 individuals was randomly created and evolved on the physical robot without any human intervention for 40 generations (approximately 2 days). Each individual, starting with a full battery, was tested for a maximum of 300 actions (a full battery allowed approximately 200 actions). Fitness points were calculated and accumulated at every action according to a function that encouraged straight motion and low sensor activation (Floreano & Mondada, 1994).

$$\Phi = \Delta V * (1 - i) \tag{1}$$

where ΔV is a measure of how straight the motion of the robot is, and $(1 - i)$ encourages distance from walls. $\Delta V = \left|(v_{left} + v_{right})\right| / (2{*}v_{max})$, $0 \leq \Delta V \leq 1$,

[3] This strategy did not change the difficulty and/or realism of the training environment, but considerably speeded up our measures.

where v_{left} and v_{right} are the signed speed values of the wheels (positive is one direction, negative the other), and v_{max} is the maximum rotation speed of a wheel (positive value). $0 \leq i \leq 1$ is the activation value of the proximity sensor with the highest activity. Controllers capable of recharging the battery could accumulate more fitness scores than others.

4.4 Analysis of results

Fitness scores were scaled in the range $[0, 1]$. The maximum fitness score attainable without recharging was approximately 0.5. Figure 4 plots the average population fitness and the fitness of the best individual at each generation during evolutionary training. After few generations there are already individuals capable of recharging, but their low score indicates that they still spend much time against walls. In the environment used for the training stage, individuals with a fitness value above 0.70 (achieved after 25 generations) performed the desired task appropriately coordinating all the modules.[4]

———————————

INSERT FIGURE 4 ABOUT HERE

———————————

Figure 5 shows the performance of the obstacle avoidance module for the best individual of the first generation. Initial motor actions[5] generated by the behavior generator network produced a random movement of the robot, resulting in a poor performance. As the robot received reinforcement signals from its interactions with the environment, the network gradually learned the correct synaptic weights for performing the obstacle avoidance behavior. A performance level of 0.6 corresponds to a good obstacle avoidance behavior. An ideal performance of 1.0 could be achieved only by a robot that moves forward in an environment without obstacles.

[4] A maximum score of 1.0 would be attainable only in an environment without walls.

[5] A motor action lasted 100 ms. Before each new motor action the speed of each wheel was set to a new value.

INSERT FIGURE 5 ABOUT HERE

4.5 Task and hardware modifications

In a second stage, we added a number of small objects in the environment and equipped the Khepera with a gripper module (figure 6). The gripper module has two degrees of freedom: it can lift/lower the arm and open/close the gripper. An optical barrier between the two segments of the gripper provides sensory information on the presence of an object.

INSERT FIGURE 6 ABOUT HERE

The desired task was that of collecting the highest number of objects and releasing them outside the arena, while maintaining the already evolved abilities of navigation, obstacle avoidance, and battery recharge.

4.6 Behavioral decomposition and module allocation

The new task can be decomposed in two relatively complex behaviors: object gripping and object releasing. The complexity comes from the fact that each module must learn to discriminate between objects and walls. Two new modules were allocated as follows.

Object-Gripping The behavior generator implemented the following sequence of programmed actions. It moved the robot towards the direction of the vector resulting from the activation of the proximity sensors, it lowered the gripper, it backed until the optical barrier was on (object well-positioned), closed the gripper, and lifted the object. The activation network decided whether the pattern of activity of the infrared sensors corresponded to an object or to a wall. The reinforcement program provided reinforcements using information coming from the optical sensor on the gripper. If the gripper detected a graspable object, the activation network received a positive reinforcement (R=1), otherwise a negative reinforcement (R=-1).[6] Learning was always enabled for all individuals. The learning algorithm was that described in section 3. The input consisted of information coming from the optical barrier and proximity sensors, and the output commanded the robot's wheels and the gripper.

Object-Releasing The behavior generator implemented the following sequence of programmed actions. It moved the robot towards the vector resulting from the activation of the proximity sensors, lowered the gripper, and dropped the object. The activation network learned to discriminate walls from objects as in the Object-Gripping module, but the value of the reinforcement signal was reversed. Learning was always enabled for all individuals. The input consisted of information coming from the optical barrier and proximity sensors, and the output commanded the robot's wheels and the gripper.

———————————

INSERT FIGURE 7 ABOUT HERE

———————————

The new modules were fully connected to all previously existing modules (figure 7) and the genotype representation was augmented by including the new inter-module connections and the synaptic values of the new activation networks.

As mentioned above, the synaptic values of the new activation networks were learned during the lifetime of each decoded individual. Learning started from the evolved synaptic values, but final values were not written back into the genetic string. Alternatively, one could use a Lamarckian mechanism, that is, encode on the genetic representation the changes acquired through life time

[6] If the robot tried to pick a wall up, the optical barrier of the gripper was not activated.

learning. Although this method might speed up both the evolutionary and learning process, it might also drive the controller into local minima (for a discussion of Darwinian versus Lamarckian evolution, see (Parisi & Nolfi, 1996)).

4.7 Training

Evolutionary training resumed from the last evolved population (generation 40). The already evolved parts of the chromosomes were masked in order to exclude them from crossover and mutation. The individuals of the initial generations of the new task pushed the objects towards the walls.[7] In order to make the evolutionary process automatic, virtual objects were employed. A virtual object is the generation of a pattern of sensory activation as if a physical object was encountered by the robot. Sensor activations were recorded before training for an object positioned at various locations around the shell of the robot. For each individual, eight objects were virtually placed in random locations. Without this procedure, somebody should have assisted the robot by manually repositioning objects in the arena during training.[8] Evolutionary training was continued for 40 generations. The fitness function, which maintained the previous components, was extended by adding 0.5 points for every gripped object and 1.0 point if the object was correctly released outside the arena (see also (Nolfi, 1997b) for a similar fitness definition). The Khepera mobile robot had to learn to discriminate between objects and walls. Both the *object-gripping* and the *object-releasing* modules learned autonomously to distinguish between objects and walls by lowering the gripper in front of the detected obstacle, observing whether the optical barrier between arms detected an object (see also (Pfeifer, 1996) for a similar procedure), and associating the sensory pattern of infrared sensor activation with the corresponding category (figure 8).

[7] As two new modules had been added to the architecture, inter-modular coordination was not optimal and obstacle avoidance was not activated every time an object was detected.

[8] As individuals of the initial generations pushed the objects towards the walls, after some generations all the objects would have been placed in contact with the walls and the robot would have not been able to pick them up by lowering the gripper.

———————————

INSERT FIGURE 8 ABOUT HERE

———————————

4.8 Analysis of results

Both the average population fitness and the fitness of the best individuals gradually increased across generations (figure 9). After approximately 15 generations (generation 55 in the figure), the best controllers already displayed the desired abilities. It should be noted that within 300 actions best individuals can pick up at most 5 real objects (corresponding approximately to a fitness value of 8). However, since virtual objects appeared at random without taking into account robot trajectories and gradual depletion of the environment observed, fitness values during evolutionary training could be higher than 8.

———————————

INSERT FIGURE 9 ABOUT HERE

———————————

Figure 10 plots the performance of the *Object-Gripping* module for the best individual after 15 generations while it learns to discriminate small objects from walls using the local learning algorithm.

———————————

INSERT FIGURE 10 ABOUT HERE

———————————

Figure 11 displays the activation of the modules composing the control architecture while the best individual of the last generation was tested with 8 real objects evenly distributed in the arena. The *Wander* behavior is active most of the time, sporadically interrupted by the *Obstacle-Avoidance* behavior. The *Light-Following* behavior becomes active when the battery is almost discharged. While the robot returns to the recharging station, it has an object in the gripper and therefore it temporarily activates the *Obstacle-Avoidance* behavior to avoid an object on its way. Once the recharging bar is detected, the *Recharge* module takes control until the battery is fully charged. Five objects are gripped and released outside the arena.

———————————

INSERT FIGURE 11 ABOUT HERE

———————————

4.9 Environmental change

The objects employed during evolutionary training had a diameter of 10 mm. In order to test the adaptation abilities of the *Object-Gripping* and *Object-Releasing* modules, all objects were replaced by larger objects with a diameter of 25 mm (figure 12) while the best individual analyzed above was tested in the environment. Recall that local learning is automatically enabled whenever the performance indicator of the module drops below a threshold (here the threshold was 100% correct response, therefore learning was on almost all the time).

———————————

INSERT FIGURE 12 ABOUT HERE

———————————

4.10 Analysis of results

Figure 13 shows the re-adaptation performance of the *Object-Gripping* module when the small objects are replaced by larger ones. No additional evolutionary training is required.

———————————

INSERT FIGURE 13 ABOUT HERE

———————————

After object substitutions, walls and objects are sometimes confused resulting in a performance drop. However, the signal coming from the optical sensor on the gripper continuously provides information that punishes or rewards the activation network. Re-adaptation to new objects takes place locally and automatically without requiring modifications in other parts of the control architecture.

5 Conclusions

In this paper we have presented a modular architecture for autonomous robots which allows for the implementation of basic behavioral modules by both hand programming and training, and accommodates for an evolutionary development of the interconnections among modules. This architecture is particularly suitable for developing autonomous robots along the lines of the BAT methodology, which stresses the importance of analyzing the target behavior into basic components and of incremental shaping. The feasibility of the approach is shown by the results of experimental activity carried out on a number of tasks of non-trivial complexity.

The approach to incremental robot shaping described here significantly extends previous results by Dorigo and Colombetti (1998). On one hand, connections among modules are developed more freely than in Dorigo and Colombetti's distributed classifier system (ALECSYS). On the other hand, the experiments reported show the feasibility of holistic shaping (i.e., of the simultaneous training of several modules and of their interconnections). As a whole, the approach advocated in this paper leads to a more natural and effective way of training autonomous robots to perform complex behaviors. At the same time, this approach also extends previous results from (Floreano & Mondada, 1996) in the direction of practical applicability of the evolutionary method. By incorporating some knowledge about the task, the adaptive process is almost one order of magnitude faster than in experiments where evolution started from scratch. Furthermore, here the evolved controllers can be easily re-used and extended to more complex tasks.

Another interesting point (which so far has been investigated only to a very limited extent) is that the proposed architecture allows one to easily mix preprogrammed and learned behaviors. This aspect is going to be of fundamental importance in practical applications, because the use of learning for certain specific behaviors may be uneconomical or even dangerous. Moreover, the fact that any reinforcement learning algorithm can be used within the basic models makes our architecture extremely flexible and potentially suitable for a wide range of possible applications.

In principle, one may consider the possibility of allowing for different learning mechanisms also at the level of connections. At this level, however, we think that restricting to an evolutionary approach is well justified. While it is not possible to constrain a priori the kind of learning to be carried out within a single behavioral module, we know that at the higher level the agent has to learn a network of connections. At this level, the use of an evolutionary algorithm ensures that the space of possible connections will be searched globally, thus minimizing the risk of converging to a highly suboptimal local maximum of the fitness function.

While the results described in this paper appear to be encouraging, we are aware that there is a long way to go before an extensive use of learning becomes feasible in practical robotic applications. In particular, we should increase our understanding of how sensory input can be used best to compute reinforcements. In fact, we think that for any learning agent it will be crucial to extract as much information as possible from its own interactions with the environment. A distributed modular approach to solving this problem seems to us an efficient solution.

Acknowledgments

The authors thank the anonymous referees for detailed and constructive comments. Joseba Urzelai is supported by grant nr. BF197.136-AK from the Basque Government and the EPFL. Dario Floreano acknowledges support by the Swiss National Science Foundation, project nr. 21-49174.26. Marco Dorigo is a Research Associate with the FNRS, the Belgian National Fund for Scientific Research. Marco Colombetti has been supported by a University Research Fund for the years 1995-96.

Reference

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, *23*, 279–303.

Ackley, D. H., & Littman, M. L. (1990). Generalization and scaling in reinforcement learning. In Touretsky, D. S. (Ed.), *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA.

Booker, L. B., Goldberg, D. E., & Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*, 235–282.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, *6*, 3–15.

Brooks, R. A. (1992). Artificial life and real robots. In Varela, F. J., & Bourgine, P. (Eds.), *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life*. The MIT Press/Bradford Books, Cambridge, MA.

Colombetti, M., Dorigo, M., & Borghi, G. (1996). Behavior Analysis and Training: A methodology for behavior engineering. *IEEE Transactions on Systems, Man and Cybernetics-Part B*, *26*, 365–380.

Dorigo, M., & Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, *71*, 321–370.

Dorigo, M., & Colombetti, M. (1998). *Robot shaping: An experiment in behavior engineering*. MIT Press, Cambridge, MA.

Dorigo, M., & Schnepf, U. (1991). Organization of robot behaviour through genetic learning processes. In *Proceedings of the Fifth IEEE International Conference on Advanced Robotics (ICAR'91)*. IEEE Press.

Dorigo, M., & Schnepf, U. (1993). Genetics-based machine learning and behavior-based robotics: A new synthesis. *IEEE Transactions on Systems, Man and Cybernetics*, *23*, 141–154.

Floreano, D. (1993). Emergence of home-based foraging strategies in ecosystems of neural networks. In Meyer, J., Roitblat, H. L., & Wilson, S. W. (Eds.), *From Animals to Animats II: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Floreano, D. (1997). Reducing human design and increasing adaptivity in evolutionary robotics. In Gomi, T. (Ed.), *Evolutionary Robotics*, 187–220. AAI Books, Ontario, Canada.

Floreano, D., & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In Cliff, D., Husbands, P., Meyer, J., & Wilson, S. W. (Eds.), *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Floreano, D., & Mondada, F. (1996). Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, *26*, 396–407.

Harvey, I., Husbands, P., & Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In Cliff, D., Husbands, P., Meyer, J., & Wilson, S. W. (Eds.), *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Meeden, L. (1996). An incremental approach to developing intelligent neural network controllers for robots. *IEEE Transactions on Systems, Man and Cybernetics-Part B*, *26*, 474–484.

Mondada, F., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In Yoshikawa, T., & Miyazaki, F. (Eds.), *Proceedings of the Third International Symposium on Experimental Robotics*, Tokyo. Springer Verlag.

Nehmzov, U., & McGonigle, B. (1994). Achieving rapid adaptation in robots by means of external tuition. In Cliff, D., Husbands, P., Meyer, J., & Wilson, S. W. (Eds.), *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Nolfi, S. (1997a). Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration. In Gomi, T. (Ed.), *Evolutionary Robotics. From Intelligent Robots to Artificial Life*, 21–48. AAI Books, Ontario, Canada.

Nolfi, S. (1997b). Using emergent modularity to develop control system for mobile robots. *Adaptive Behavior*, *5*, 343–364.

Nolfi, S., & Floreano, D. (1998). Co-evolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial Life*. To appear.

Parisi, D., & Nolfi, S. (1996). The influence of learning on evolution. In Belew, R. K., & Mitchell, M. (Eds.), *Adaptive Individuals in Evolving Populations. SFI Studies in the Science of Complexity*, Vol. XXVI, 417–428. Addison-Wesley.

Pfeifer, R. (1996). Building "fungus eaters": Design principles of autonomous agents. In Maes, P., Matarić, M., & Meyer, J. (Eds.), *From Animals* *to Animats IV: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagation of errors. *Nature*, *323*, 533–536.

Skinner, B. F. (1938). *The behavior of organisms: An experimental analysis.* Appleton Century, New York.

Tani, J., & Nolfi, S. (1998). Learn to perceive world as articulated: An approach for hierarchical learning. In Pfeifer, R. (Ed.), *Proceedings of the Fifth International Conference of the Society for Adaptive Behavior*. MIT Press, Cambridge, MA.

Touretzky, D. S., & Saksida, L. M. (1996). Skinnerbots. In Maes, P., Mataric, M., Meyer, J. A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats IV: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA.

Touretzky, D. S., & Saksida, L. M. (1997). Operant conditioning in skinnerbots. *Adaptive Behavior*, *5 (3/4)*, 219–247.

Figure Captions

Figure 1: Structure of a module. See text for explanation. In this module both the activation network and the behavior generator are implemented as neural networks. The module also includes a strictly local learning algorithm L that uses reinforcement signals coming from a reinforcement program P. A module can receive weighted signals from other modules (connections in), which are summed up and combined with the sensory input in order to drive the activation level. In the same way, a module can send its activation state to other modules in the form of weighted signals (connections out).

Figure 2: Genetic encoding of the control architecture. A chromosome encodes the synaptic strengths of the activation networks of all modules and all inter-module connections. For each module of the architecture, synaptic weights of the activation network are coded first, followed by connection weights to the other modules. Each value is encoded on the genetic string as a 4-bit integer number and is normalized in the continuous range [0,1] before decoding it into the corresponding weight value. New modules are added by increasing the genotype length. In this case, previously evolved modules and inter-connections are masked to protect them from cross-over and mutation operators.

Figure 3: The Khepera robot is placed in the environment with a battery charger. The task is to keep moving in the environment avoiding obstacles, and periodically recharge the batteries.

Figure 4: Fitness values during evolution. Dash-dotted line shows average population fitness, continuous line the fitness of the best individual at each generation. Data smoothed using rolling averages (time window = 3).

Figure 5: Performance of the Obstacle-Avoidance module while learning to avoid walls from randomly initialized weights. Reinforcement values are rolling averages (window size = 30). Performance below zero means higher percentage of negative reinforcements, above zero higher number of positive reinforcements.

Figure 6: Small objects of roughly equal size are randomly distributed in the environment and the Khepera robot is equipped with a gripper module. The additional task is to find the objects, pick them up and release them outside the

arena while maintaining the previous navigation and recharging abilities.

Figure 7: The new architecture for the extended task and hardware components. Two new modules have been added and connected to previous modules.

Figure 8: Self-supervised learning to discriminate objects by active exploration. The robot wanders around until it detects an obstacle (1 and 2), then turns towards the detected obstacle (3), lowers the gripper in front of it, and observes whether the optical barrier between arms detected an object in order to associate the sensory pattern of infrared sensor activation with the corresponding category (4).

Figure 9: Fitness values during incremental evolution on the extended task. Dash-dotted line shows the average population fitness, continuous line the fitness of the best individual at each generation. Data smoothed using rolling averages (window size = 3). Temporary dip from generation 60 to 70 took place overnight and it might be due to some physical events, such as the robot getting stuck with the gripper, or noisy transmission over the serial line.

Figure 10: Performance of the Object-Gripping module while learning to discriminate small objects (diameter is 10 mm) from walls. Reinforcement values are rolling averages (window size = 30). Performance below zero means higher percentage of negative reinforcements, above zero higher number of positive reinforcements. Performance around zero means random discrimination.

Figure 11: Module activations for 225 actions of the best individual after 80 generations. Five objects are picked up and released outside the arena. While the third object is in the gripper, the robot goes to recharge. Legend: BtLv: battery level; Wand: Wander behavior; ObAv: Obstacle-Avoidance behavior; LtFl: Light-Following behavior; BtCh: Recharge behavior; ObGr: Object-Gripping behavior; ObSr: object in the gripper (sensor); ObRl: Object-Releasing behavior.

Figure 12: A small object (10 mm of diameter) seen during evolutionary training and a new larger object (25 mm).

Figure 13: Performance of the Object-Gripping module when small objects are substituted by larger objects. Reinforcement values $-1, 1$ are rolling averages (window size = 30). Performance below zero means higher percentage of negative reinforcements, above zero higher number of positive reinforcements. Performance around zero means random discrimination.

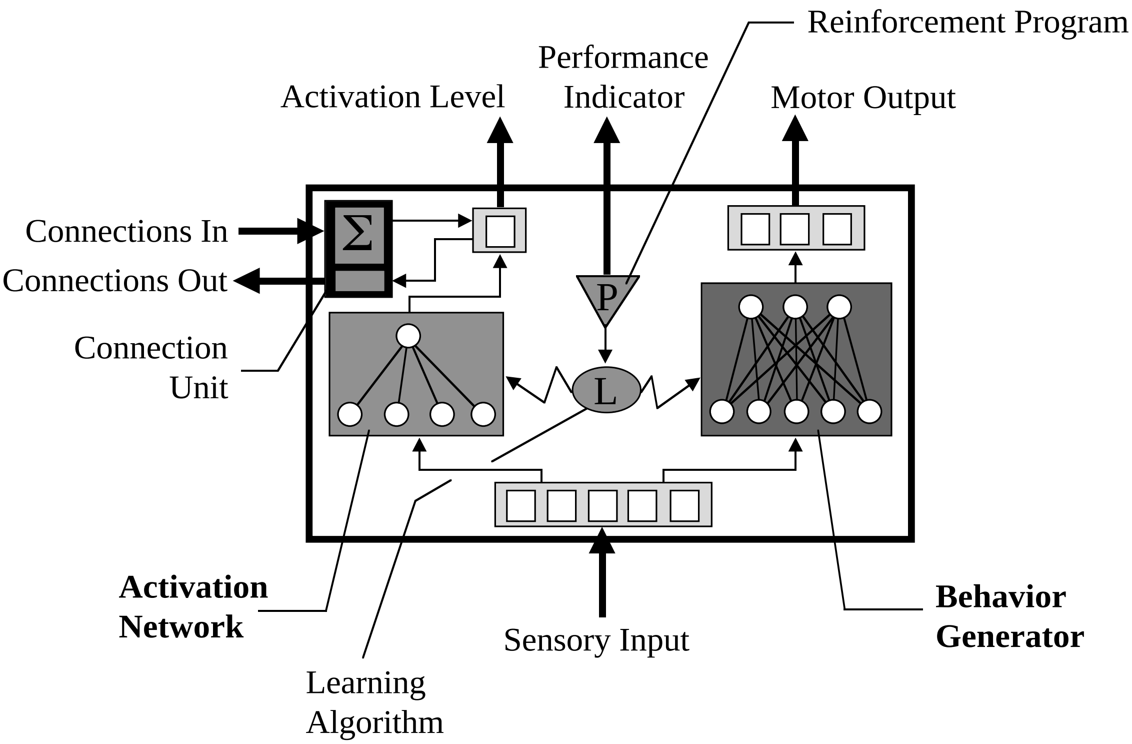

Reinforcement Program
Performance Indicator
Activation Level
Motor Output
Connections In
Connections Out
Σ
P
Connection Unit
L
Activation Network
Behavior Generator
Sensory Input
Learning Algorithm

Figure 1:

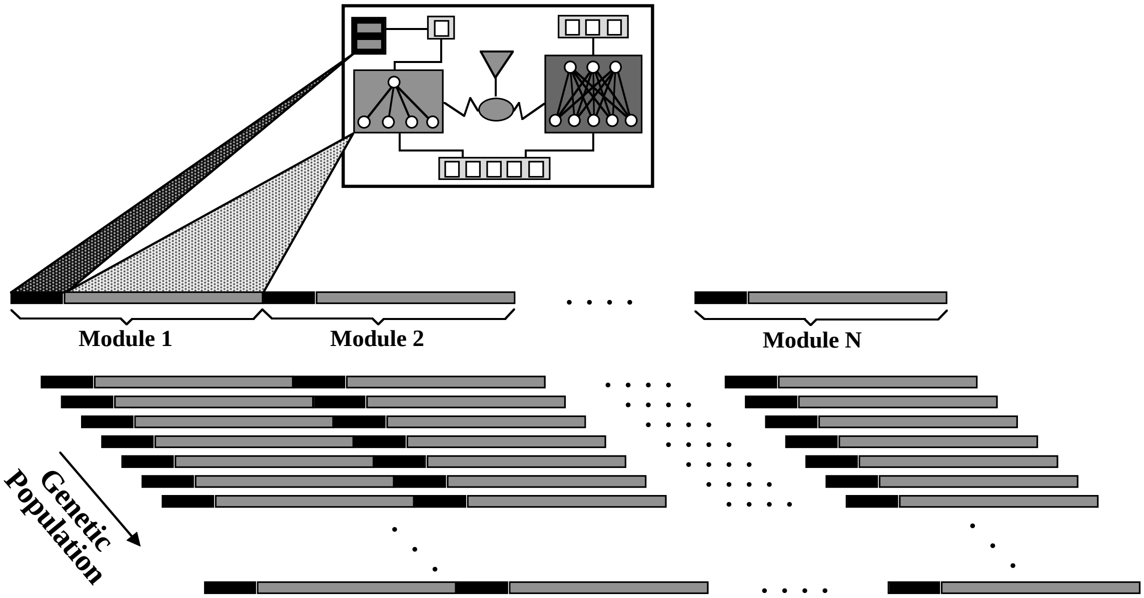


Figure 2:

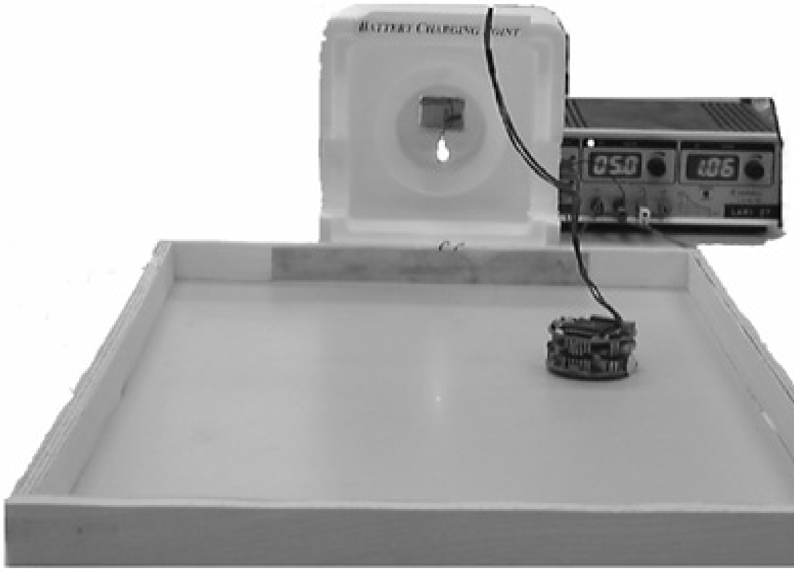


Figure 3:

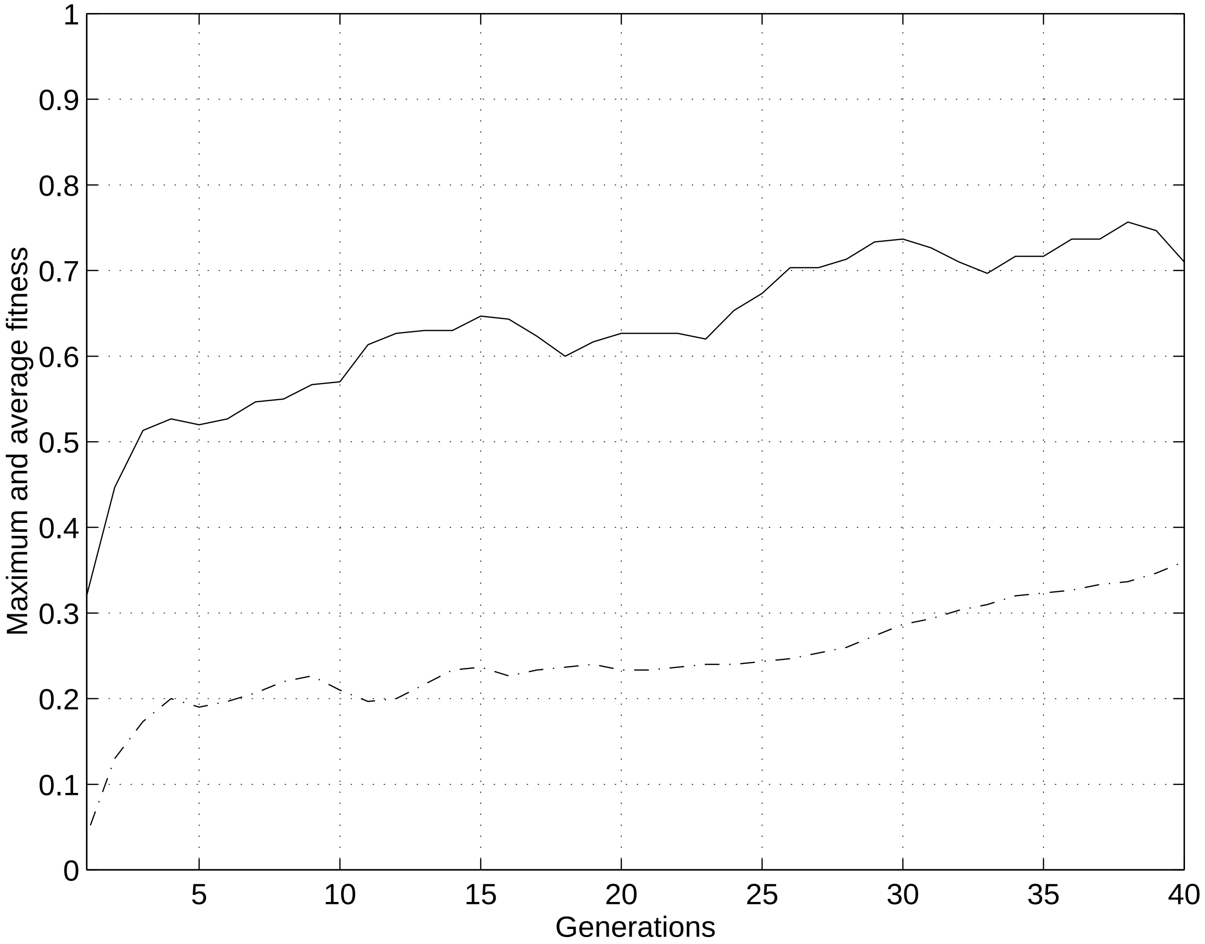

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Maximum and average fitness
5
10
15
20
25
30
35
40
Generations

Figure 4:

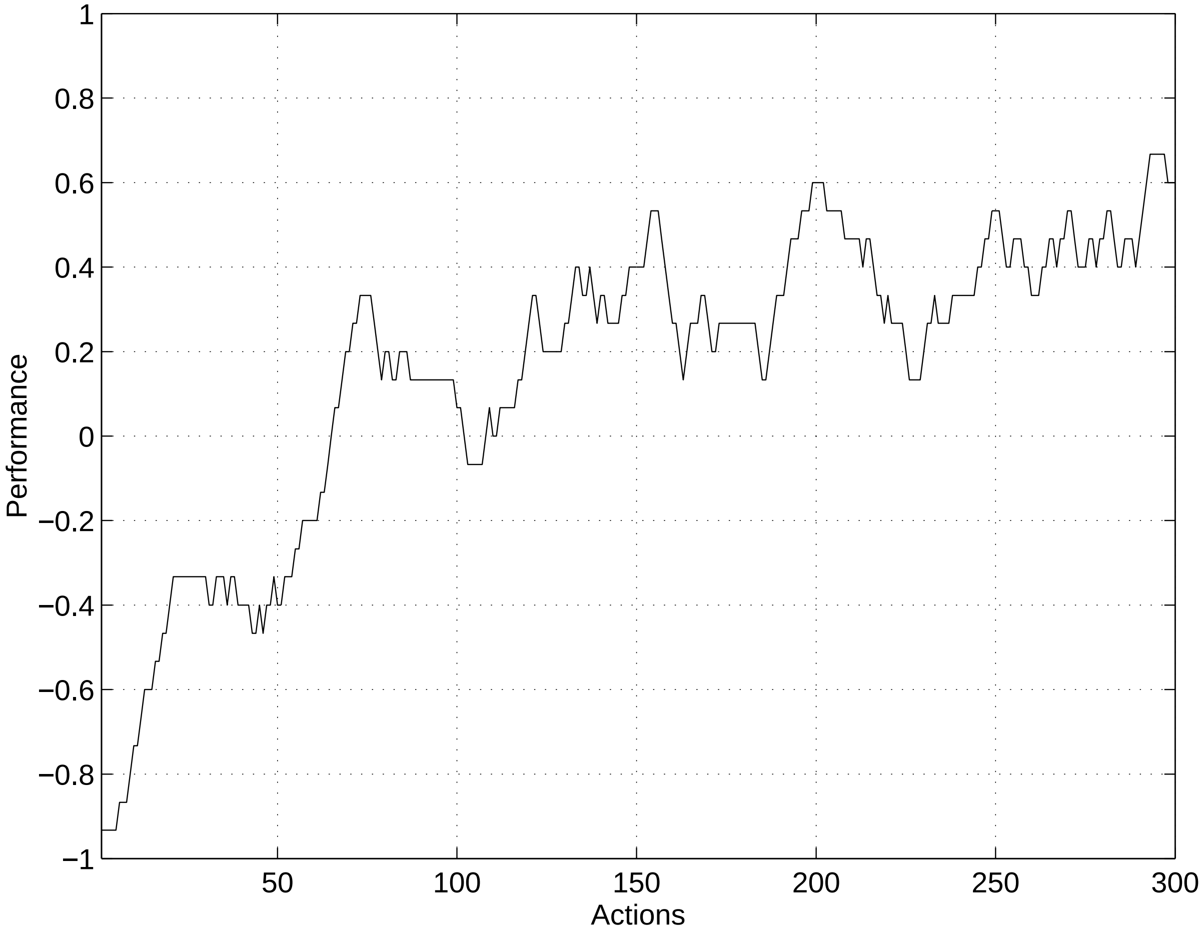

Performance
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
50
100
150
200
250
300
Actions

Figure 5:

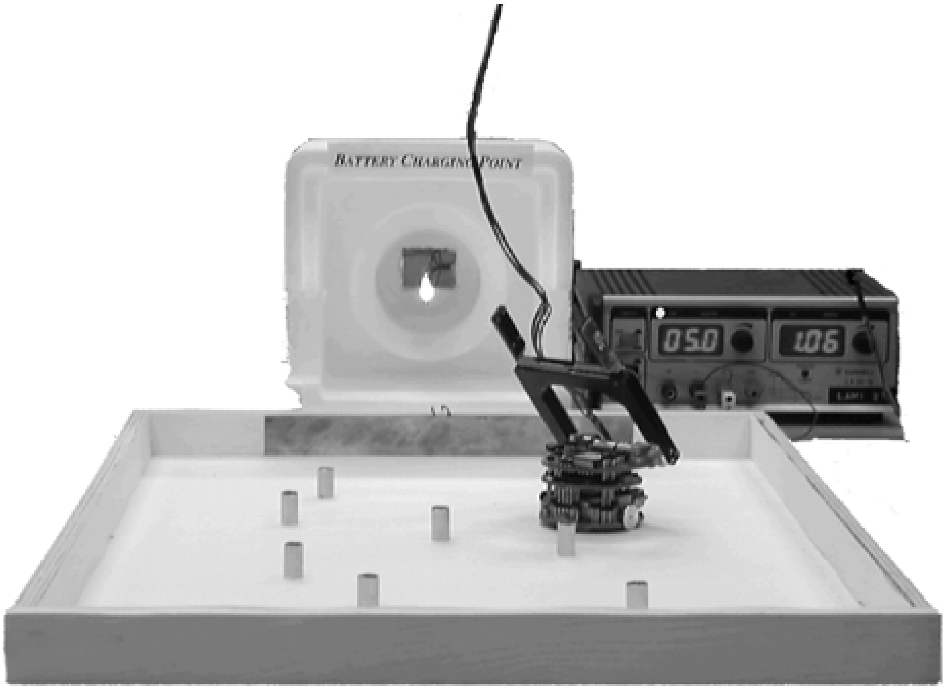


Figure 6:

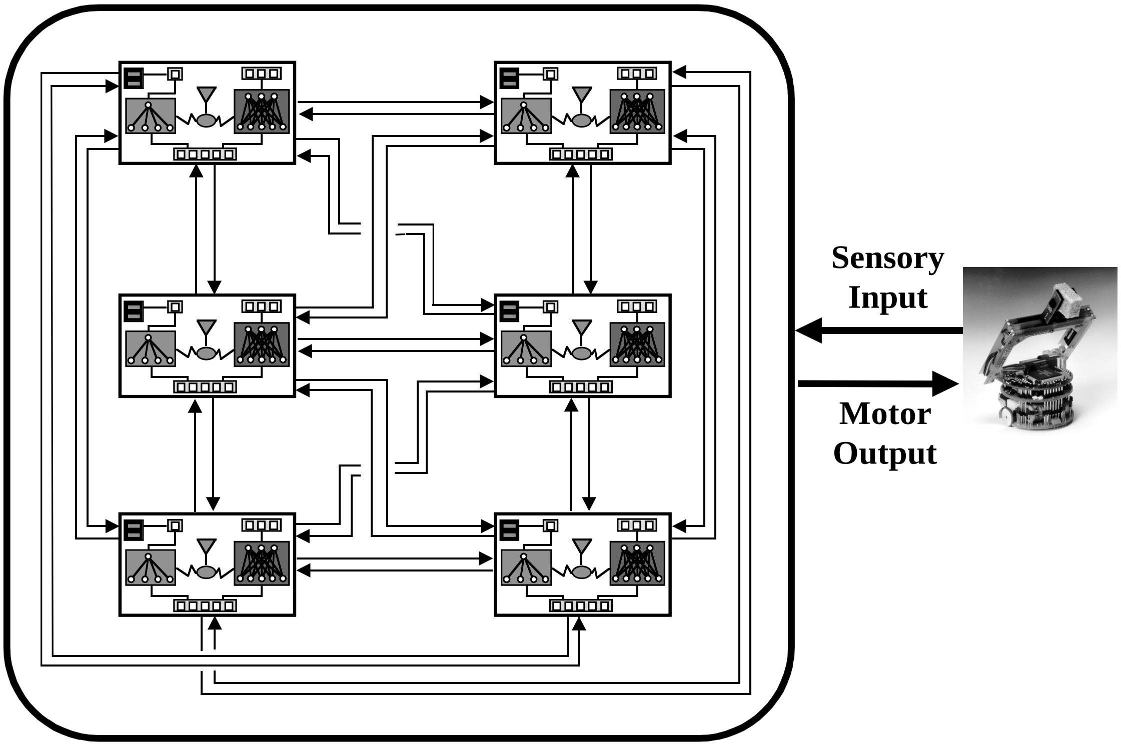
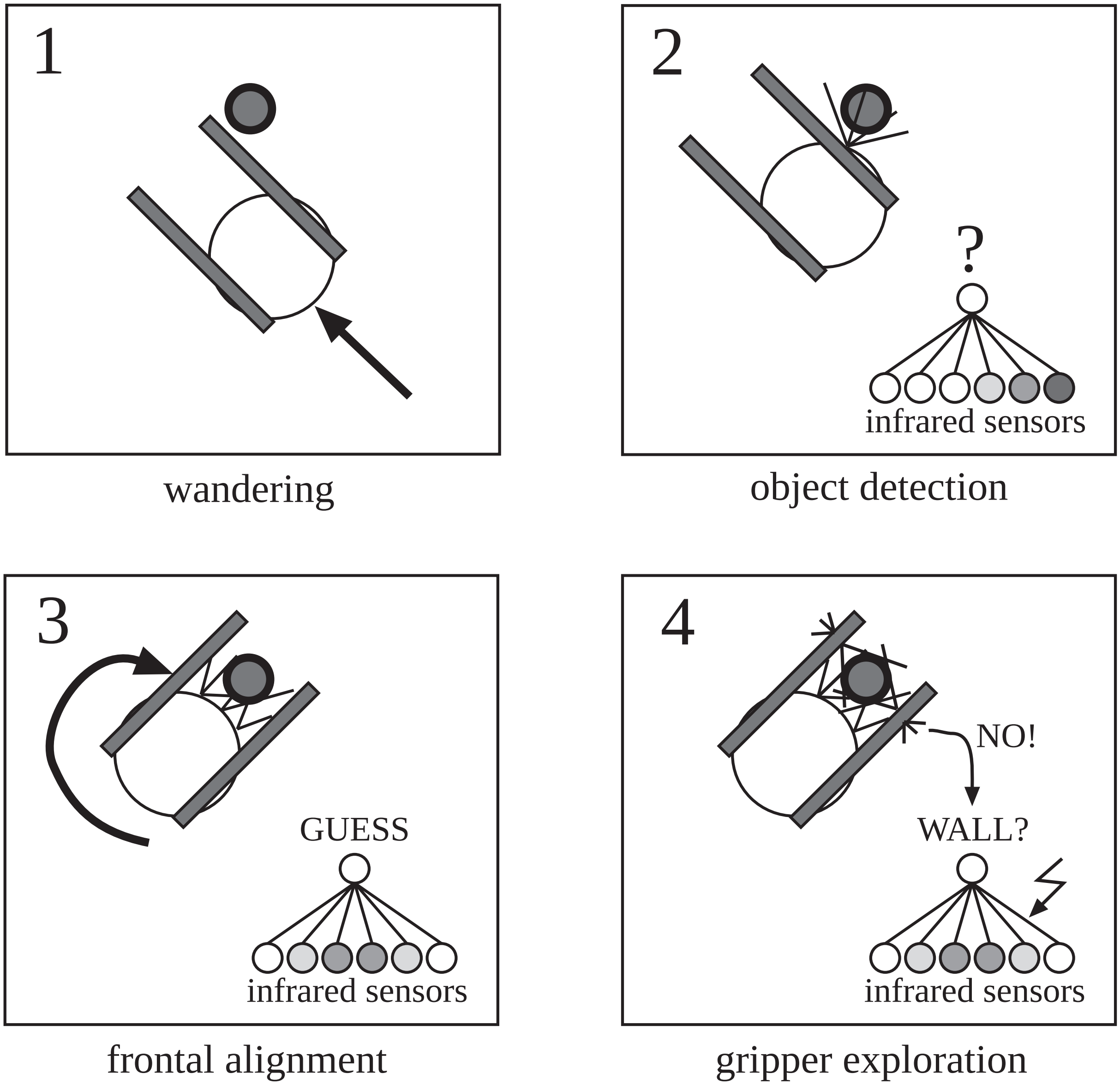


Figure 7:



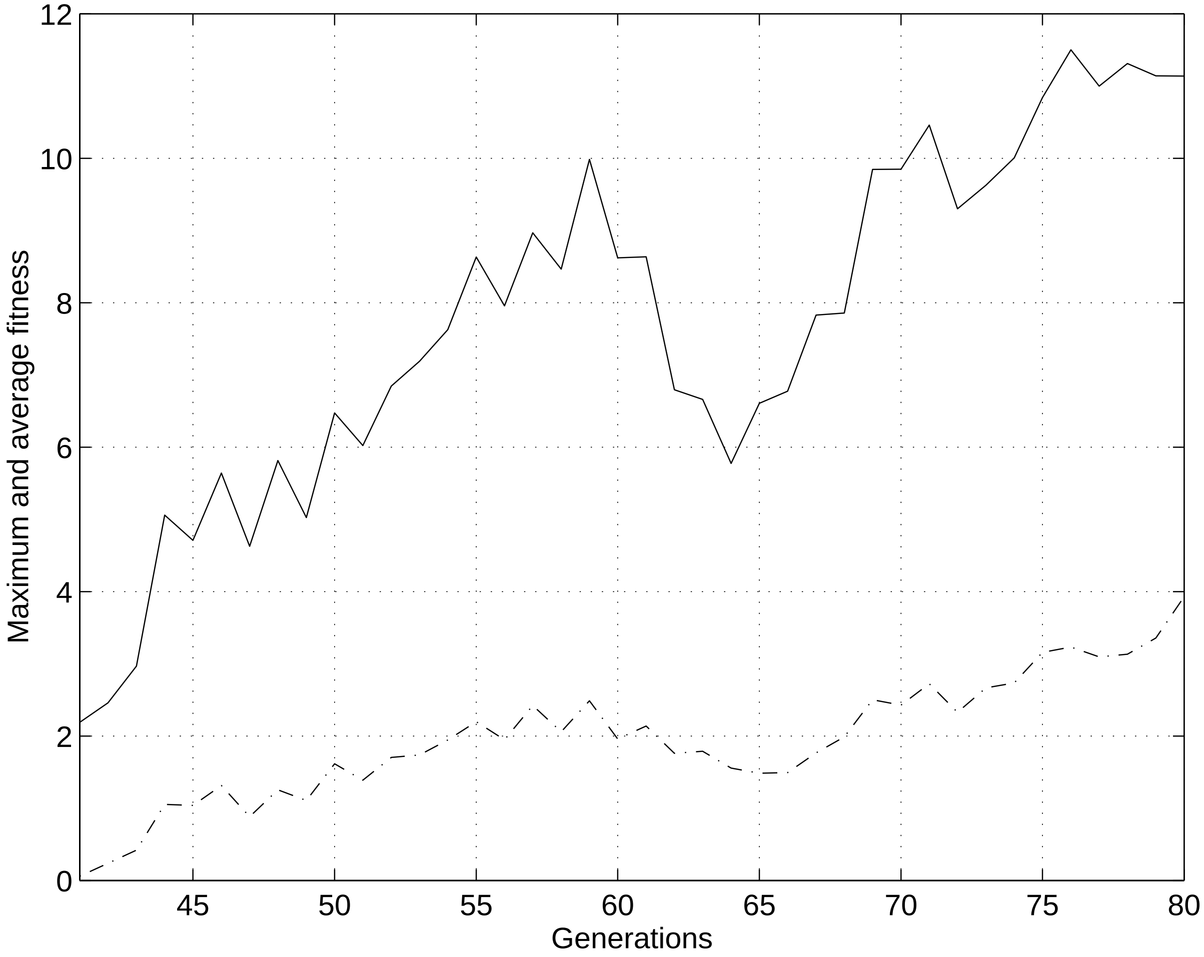
wandering

object detection

frontal alignment

gripper exploration

Figure 8:



12
10
8
6
4
2
0
Maximum and average fitness
45
50
55
60
65
70
75
80
Generations

Figure 9:

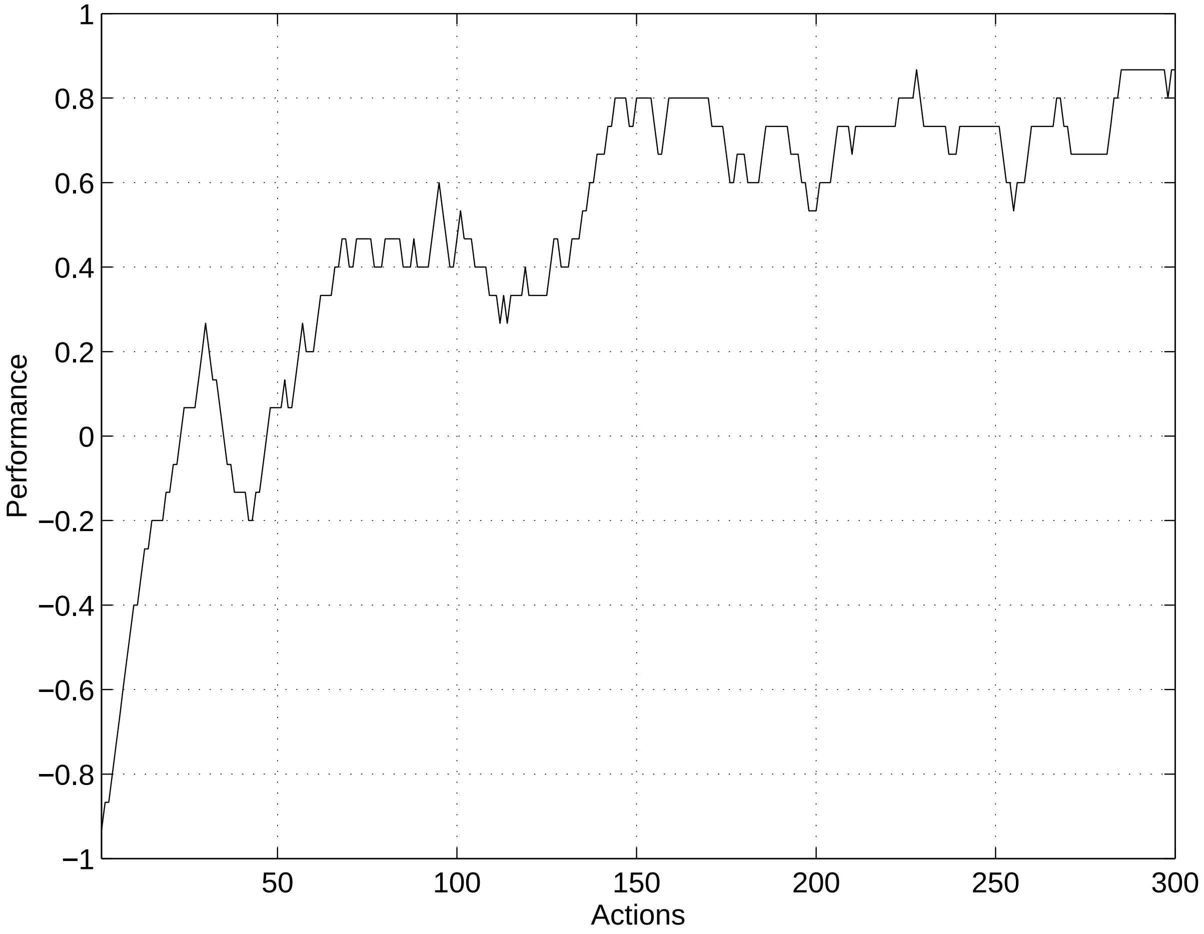

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
Performance
50
100
150
200
250
300
Actions

Figure 10:

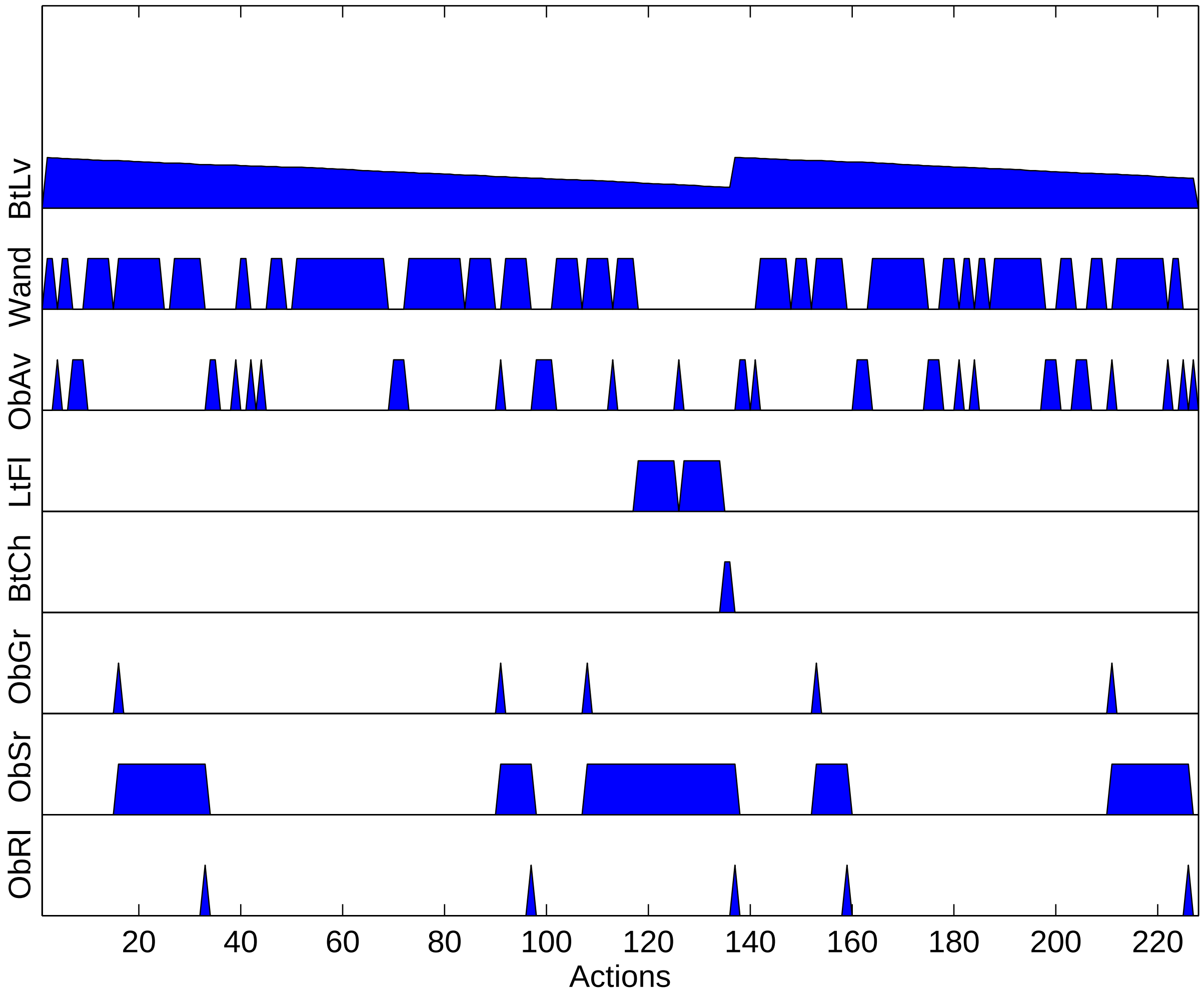

BtLv
Wand
ObAv
LtFl
BtCh
ObGr
ObSr
ObRl
20
40
60
80
100
120
140
160
180
200
220
Actions

Figure 11:

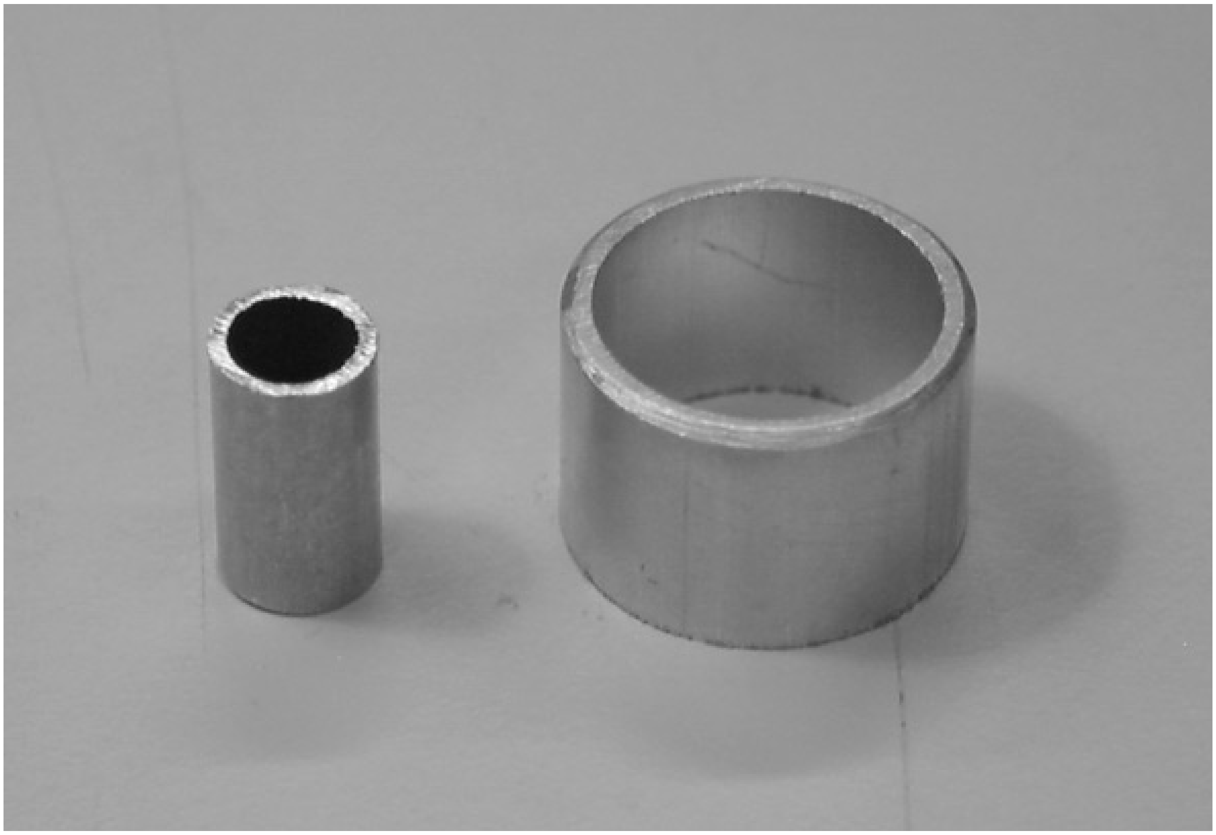


Figure 12:

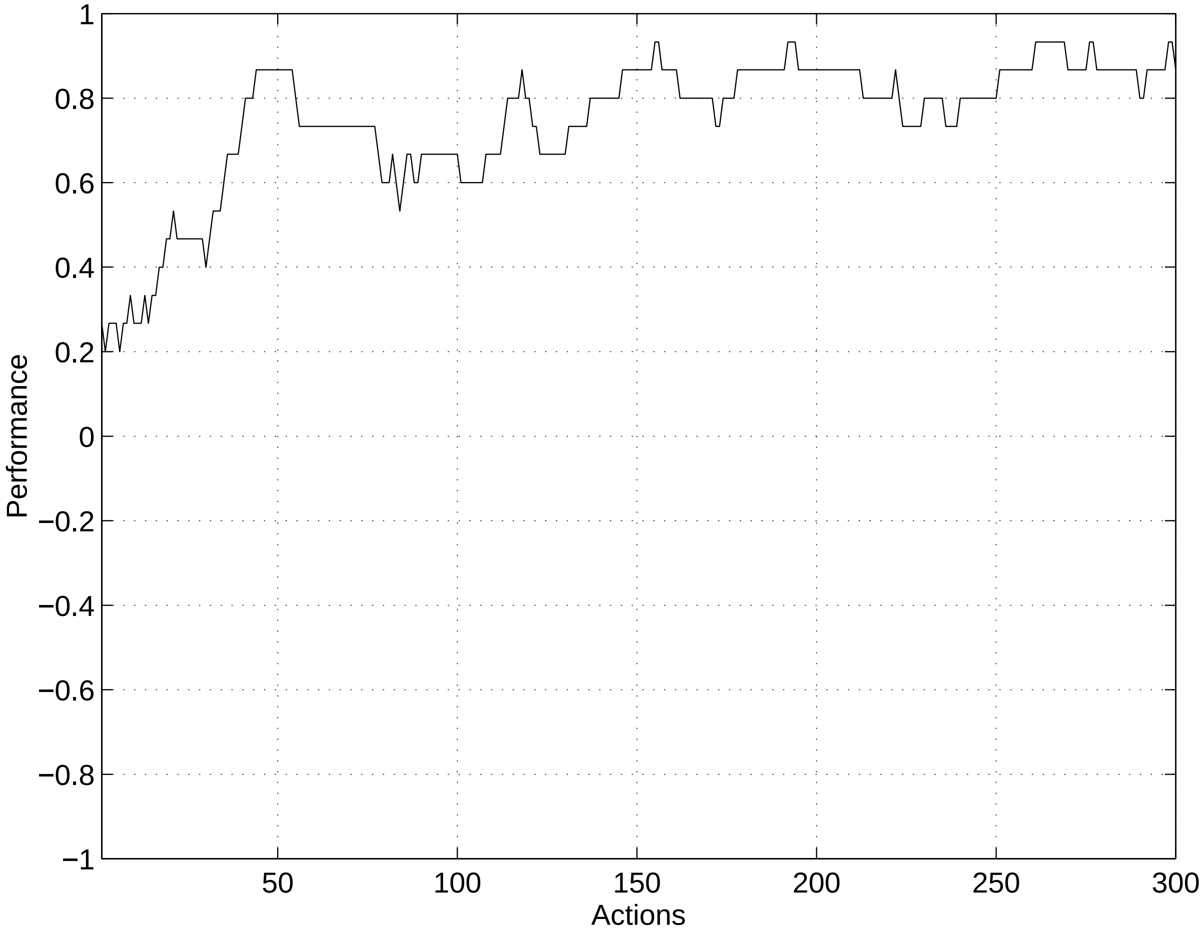

1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
Performance
50
100
150
200
250
300
Actions

Figure 13: