Hypothetical Evolutionary Paths: Designing Adaptive Robot Behaviors

Torbjørn S. Dahl
Robotics Research Laboratory
Department of Computer Science
University of Southern California
941 West 37th Place, SAL 214
Los Angeles, California 90089-0781
`phone: 213-740-4523`
`fax: 213-821-5696`
`tdahl@robotics.usc.edu`

Christophe Giraud-Carrier
ELCA Informatique SA
Avenue de la Harpe
CH-1000, Lausanne 13i
Switzerland
`cgc@elca.ch`

Abstract

We present an evolution-inspired design methodology for adaptive robot control algorithms called Hypothetical Evolutionary Paths. This methodology produces complex adaptive algorithms while avoiding centralized hypotheses or policies in the same way that Behavior-Based systems in general avoid the use of centralized world models. The algorithms produced adapt rapidly without suffering from the large search spaces and long convergence times that have made traditional learning algorithms unatractive in situaded systems. The methodology uses a schema-based model of behavioural layers that allows incremental design of increasingly sophisticated adaptive strategies for solving a given prolem. The algorithms produced apply the different strategies concurrently, giving robots a high performance-level during learning and providing graceful degradation on restricted hardware failure. We present two sets of experiments in simulation demonstrating rapidly adapting algorithms produced using the Hypothetical Evolutionary Path methodology in the problem domains of foraging and conflict resolution.

Keywords

Evolution, Learning, Behavior Design, Foraging, Mapping, Conflict Resolution

Running Head

Designing Adaptive Robot Behaviors

1 Introduction

Adaptiveness has been identified as an important factor in intelligent behavior [McFarland and Bösser, 1993], though it is emphasized that the two are not the same. Reactive behavior can be said to be intelligent if it is effective in the environment it is naturally expressed.

Consider an animal that competes with others members of its own species for a limited resource. Such an animal can have a number of increasingly sophisticated strategies for resolving conflicts. One very basic, perhaps implausible, strategy would be to fight until victory or death. Increasingly complex physiologies and cognitive abilities can support increasingly efficient conflict resolution strategies such as, yielding, adherence to pecking order, and coalition building. Each conflict resolution strategy can be seen from a learning point of view, as a solution to a problem of adaptation. As strategies become more sophisticated, so do the related learning problems. The more environmental, social and historical elements an entity can utilize in a strategy, the more successful it can be.

Hypothetical Evolutionary Paths (HEP) is an evolution-inspired methodology for developing adaptive robot control algorithms. The methodology is based on the incremental development of concurrent, increasingly sophisticated problem solving strategies. HEP uses a schema-based behavior model that allows new strategies to use elements of simpler, underlying strategies to restrict the search space of the new learning problem.

HEP improves on existing methodologies in two ways. First by producing algorithms that employ multiple concurrent strategies of different sophistication to a given problem. In the case of limited hardware failures in distributed systems, such algorithms can retain a higher level of performance than algorithms that use a single solution. Secondly, the adaptivity in the algorithms developed using our methodology is highly restricted. Highly restricted adaptivity allows the algorithm to adapt quickly after minimal amounts of experience.

The weakness of the algorithms our methodology produces is that the solutions are highly specialized compared to existing learning algorithms. This weakness is a reflection of the trade-off of generality for efficiency that was the motivation for Behavior-Based (BB) systems [Brooks, 1991]. The conclusions from many years work on learning in BB systems has made it clear that 'To build truly effective embedded systems, we will have to combine *a priori* knowledge or structure, as provided by a human programmer, with the ability to adapt'[Kaelbling, 1993]. A BB approach to learning must necessarily be more of an engineering effort than an automated learning effort. The goal, at least initially, is to develop adaptive capabilities, not to learn behaviors. High level animal and human learning capabilities can produce new behaviors, but such abilities are only present in the most advanced, mammals, primates and humans. From a BB perspective, there is a lot of ground to cover before robust and efficient instances of such capabilities can be implemented in artificial systems.

2 Issues and Related Work

We use the term *behavior* to denote only externally observable displays of movements. We call the solution to a problem provided by a behavior a *behavioral strategy*. We use the term *behavioral layer* to denote the internal structures that give rise to a specific behavior. Controllers are referred to using the highest behavioral layer it contains.

A behavioral layer implements a partial mapping from a set of *stimuli* to a set of *responses*. *Adaptation* is a change in this mapping as a result of experience, i.e. previous stimuli and responses or *actions*.

2.1 Speed of Adaptation

In order for adaptation to be effective in a situated system, it is necessary that the changes in the expressed actions are take place within a meaningful time-frame. Different strategies operate on different time-scales. In terms of conflict resolution, it is important in a fight to learn quickly whether you are stronger or weaker than your opponent in order to yield quickly if you are weaker. However, in a complex social hierarchy, sophisticated strategies of cooperation and coercion can be a matter of gathering information and taking actions over periods of weeks, months, or even years.

There may be many factors that slow down the speed at which a robot adapts both with regards to *action selection* and *learning from experience*, the two sub-problems of adaptation [Maes, 1994]. We look mainly at two ways of increasing the speed at which an algorithm adapts: problem restriction and using prior knowledge.

Problem Restriction There are several ways of reducing a learning problem. Reducing the number of stimuli or the number of actions reduces the dimensionality of the learning problem. Choosing actions that minimize the length of useful action sequences reduces the complexity of the hypotheses. Dividing a large learning problem into several smaller problems also helps to reduce the combinatorial complexity of a problem and hence restricts the total search space presented.

In BB system, learning has been restricted to a set of pre-specified behaviors [Mahadevan and Connell, 1992, Asada et al., 1995], to multiple interacting behaviors [Stone and Veloso, 1999, Andre and Russel, 2001] or to combining behaviors [Maes and Brooks, 1990, Matarić, 1997]. Problem restriction is an important topic for the whole field of ML. In robotics the real time constraints make problem restriction crucial. The layered restrictions introduced by the BB paradigm has also been used by other pragmatic approaches [Stone, 2000]. Bryson [Bryson, 2001] proposes problem restriction principles aligned with HEP, where the context of a behavior restricts the learning problem considered.

Using Prior Knowledge One way of using prior knowledge of a problem domain to increase the speed of adaptation is to express such knowledge in a machine readable formalism and let a learning algorithm use it automatically to improve the way the hypothesis is changed. This approach is common in Explanation Based Learning. Another way of using prior knowledge is to let it to guide the identification of an appropriate hypothesis space to search, i.e. let it influence the choice of exploratory actions. This kind of bias is called *declarative bias* [Russel and Norvig, 1995].

A third approach to speeding up learning by using knowledge about a problem domain is to provide more detailed feedback. Extra feedback can help pruning the search space by indicating whether a learner is on the right track. When this is done in a social context during the development of a child, it is called *learning by scaffolding* [Breazeal and Scassellati, 1998]. When humans used this technique to teach animals novel behaviors, it was called *shaping* [Dorigo and Colombetti, 1998]. Finally when such mechanisms are designed into a system they have been called *progress estimators* or *heterogeneous reward function* [Matarić, 1994b].

Matarić [Matarić, 1994a] stated that a situated learning model should:

1. minimize the learner's state space
2. maximize learning at each trial

HEP follows Matarić's recommendations to the extreme in minimizing the search spaces considered by each adaptive element and making maximum use of The emphasis on design allows a programmer to use prior knowledge of the problem domain to develop efficient solutions. The incremental development methodology improves the speed of adaptation by promoting highly specialized adaptive mechanisms. It also provides explicit guidelines on how to integrate preprogrammed and adaptive elements.

The use of multiple concurrent strategies of different sophistication, and the distributed nature of our behavior model, both improve the robustness of the solutions produced. The quality of our example solutions is evaluated through experiments on simulated robots.

2.2 Biological Inspiration

Many have seen the benefits of taking inspiration from biology when designing and implementing robot controllers. This merger of two scientific fields has created new areas of research such as the fields of computational neuroethology [Cliff, 1991], biorobotics [Beer et al., 1998] and constructive biology [Nehaniv et al., 1999]. Taking inspiration from biology has also been attractive as a way of restricting the search space for learning in autonomous agents and robots.

Bryson and Stein point out the importance of the modularity found in the human brain as a way of reducing complexity [Bryson and Stein, 2001]. Moren et al. [Morén and Balkenius, 2000] model biological mechanisms for adaptation on a neural level for use in robots. Corbacho [Corbacho and Arbib, 1997] uses

Schema-Based Learning to model the adaptive behaviors of frogs. However, as far as we know, no attempts have been made to develop general methodologies for developing biologically inspired adaptive behaviors.

3 The Neural Circuit Model

To support HEP, we developed a modular, schema-based model of behavioral layers that facilitates inter-layer data sharing. Schema theory [Arbib, 1989] represents brain structures as elements in a network of distributed, communicating, concurrently executing nodes or *schemas*. A schema can represent a perceptual structure, a structure for motor control or a structure for any other activity in the brain in general, such as the recognition of different objects, planning or control. Schemas are ultimately defined in terms of interaction with an external environment rather than through reference to pre-set formalism. Schemas have been successfully used as elements of behaviors, providing a finer grain of distributed computing than subsumption-style architectures [Arkin, 1989].

The Abstract Schema Language (ASL) [Weitzenfeld et al., 2000] provides a formalism for describing schema-based architectures. Our model of behavioral layers is based on a similar formalism called Neural Circuits (NCs). NCs correspond closely to neural schemas as defined by ASL, but with two main differences. NCs can pass structured data between each other, though still only uni-directionally. These structures are called *percepts*. NCs also keep excitation or control separate from data. The NCs were developed with a focus on programmability rather than biological credibility. The generalized inter-circuit communication and the separation of the control- and data-streams makes the NCs easier to use for procedural implementation of pre-defined behaviors. However, a procedural implementation of a NC is not necessarily adaptive in the way an artificial neural network (ANN) implementation would be. To emphasize, a NC does not have to contain an ANN. In our examples the functionality of NCs are provided by a procedural implementation in C++.

Most physiological theories of behavior concern different brain areas and their functionality [Carlson, 2000]. The neural circuit model allows us to take inspiration from these theories by allowing us to do cognitive modeling on not only a neural level but also on a super-neural or neural circuit level. This is useful when general effects and algorithms are suggested that can be implemented with a standard programming language although it is not clear how to implement anything equivalent using a low-level neural model.

3.1 Neural Circuits in Behavioral Layers

Consider again our motivating example, in which a small animal had to compete with other members of its species for a scarce resource. One of the simplest conflict resolution strategies one can imagine is to attack all competitors on sight, uncritically. While the evolutionary utility of this strategy is debatable, as an example it demonstrates clearly the use of NCs for behavior design. To implement this strategy, the animal would need to identify and attack its opponent probably using its eyes for identification and legs for propulsion. The simulated Khepera robots we used for our experiments had a K6300 color camera and two controllable wheels providing corresponding physiological functionality. Our design of an *Uncritical Fighting* strategy started with abstracting away the application programming interfaces (APIs) of the camera and the wheels by encapsulating them in NCs.

The NCs of the *Uncritical Fighting* layer are shown as boxes in Figure I. Boxes with rounded corners indicate encapsulations of external APIs. Boxes with dashed lines are circuits from other layers. The arrows connecting the NCs indicate excitation or control, whilst the labels on the arrows indicate the communication of percepts or data. The vertical lines ending in a circle indicate inhibition. Another common notation not shown here is a black dot connected to a drive. This indicates that that drive inhibits all other drives.

Figure I position.

The Webots API for camera control was encapsulated in a **K6300 Sensor** circuit which was always firing, providing an abstract data type, the *Image* percept, which contained the image data provided by the camera. The API for wheel control was encapsulated in two **Wheel Actuator** circuits that accepted any number of signed integers. The circuits added these integers, the sum was bound by the maximum and minimum wheel speeds and used to set the speed of each wheel.

An animal following the *Uncritical Fighting* strategy described above would have to be able to use its visual and locomotional facilities to recognize and attack its competitors. These two skills are instances of more general capabilities: opponent recognition and approach. We imbued the opponent recognition and approach capabilities in the simulated Khepera robot by means of a **Khepera Sense** circuit and an **Approach Competence** circuit. The **Khepera Sense** circuit recognized the color of a Khepera robot in the provided *Image* percept and used the hight and relative position of the robot in the image to estimate the other robot's range and bearing. The range and bearing were used to instantiate a general *Object* percept which the **Khepera Sense** circuit provided as its output. The **Approach Object Competence** circuit accepted a general *Object* percept and used the relative position of the object to servo the robot toward it by requesting the relevant wheel speeds from the **Wheel Actuator** circuits.

The reason for creating a dedicated circuit for a particular sense or capability was to remove functional duplicity. It was clear from our design that a Khepera recognition capability would also be needed for other more advanced conflict resolution strategies and an approach competence would also be needed by feeding related behavioral layers. Separating out these general capabilities and letting different behavioral layers share access to them prevents the reproduction of almost identical functions in different behavioral layers. We used the terminology **sensor** circuit and **actuator** circuit for circuits that encapsulate hardware. For derived capabilities we use the terms **sense** circuit and for general skills we use the term **competence** circuit.

The *Uncritical Fighting* behavior sometimes had to be inhibited to allow the expression of other more pressing behaviors such as obstacle avoidance, and also had to inhibit less pressing behaviors such as foraging. If circuits that are shared between multiple behavioral layers are inhibited, this implies an inhibition of all the related behaviors. In order to allow inhibition to affect only the *Uncritical Fighting* behavior without inhibiting any other behaviors, we introduced a **Uncritical Fighting Drive** circuit. This circuit is unique to the *Uncritical Fighting* layer and is the recipient and origin of any inter-behavioral inhibitory connections.

3.2 General Circuit Classes

The elements presented in Figure I are instances of general circuit classes. The general classes were inspired by the *Edmund* architecture which had explicit elements for sensors, actuators and control [Bryson, 2001]. Concurrent and distributed execution of behavioral layers are two of the strengths of the BB paradigm that help improve efficiency and allow graceful degradation. Work by Nicolescu and Matarić [Nicolescu and Matarić, 2001] introduces another architecture that provides explicit access to elements of behavior for learning purposes. In the NC model, sensors and actuators encapsulate the low-level hardware. Senses and competences are high-level sensor and actuator interpretations respectively. Drives handle interactions between multiple senses and competences and also interact with other behavioral layers through inhibition. Each behavioral layer can contain several **Sensor**, **Sense**, **Competence** and **Actuator** circuits. Drives are unique representations of behavioral layers. They are the only NC that cannot be shared between layers. They typically handle evaluation of data from multiple sensors in order to determine its general excitatory state and then modify this state according to any inhibitions received from other behavioral layers. Firing drives may in turn inhibit other behavioral layers. In the NC model inhibitions are indicated by lines between circuits with a circle at one end indicating the inhibited circuit. This is the notation used in the original subsumption architecture [Brooks, 1986].

4 Designing Adaptive Behavioral Layers

During the design and implementation of different behavioral strategies we noted common processes that were repeated for each strategy. We have generalized these processes to form elements of a methodology for developing adaptive behaviors:

- Sense and Competence Delegation
- Memories as Traces

- Duplication and Specialization

HEP starts with an envisioned top-level behavioral strategy that solves a problem a robot is facing. In our motivating example, the problem is conflict resolution and the top-level behavioral strategy is the *Stylized Hierarchy Formation* strategy, where robots used stylized courtship displays rather than fighting to establish their positions in a hierarchy.

From this starting point, HEP progresses by reducing a behavioral strategy to a set of simpler, underlying strategies from which the top-level strategy could have evolved by the principle of duplication and specialization. Each of the supporting strategies implement a simpler, but complete solution to a problem facing the robot, the underlying problems are often, but not necessarily similar to or related to the problem solved by the top-level strategy. This process is then repeated on each of the supporting strategies until an the supporting strategies have been reduced to simple reactive behaviors that can be implemented directly.

The supporting strategies are hypothetical evolutionary stages, elements in a hypothetical evolutionary chain, connected by hypothetical evolutionary steps leading from simple strategies and simple problems to the top-level strategy and problem. Certain top-level strategies might need multiple supporting strategies, in which case the hypothetical evolutionary path will have one branch for each of the supporting strategies.

The evolutionary path we hypothesized for the *Stylized Hierarchy Formation* strategy is given in Figure VI.

4.1 Sense and Competence Delegation

The first step in identifying supporting strategies for a given top-level strategy is to identify the sense and competence requirements of the top-level strategy. The sense requirements are the features of the environment that must be recognized by the robot to allow it to follow the top-level strategy. Likewise, the competence requirements are the motory skills the robot must have.

The second step is to try to design strategies less complex than the top-level strategy that make use of the same or simpler versions of the senses or skills needed for the top-level strategy.

By delegating as many as possible of the required sense and skill capabilities to supporting layers we minimize the evolutionary step up to the top-level and simplify the incremental development.

In the case of the *Stylized Hierarchy Establishment* strategy, we needed to recognize another robot and be able to tell when it was putting on a display. We also needed the ability to behave according to a place in a hierarchy. We dedicated the robot recognition and hierarchy dependent behavior to an underlying *Pecking Order Adherence* strategy which used a simulating fighting behavior to establish a hierarchy and which implemented the hierarchy dependent behavior. This left only the display recognition to be implemented by the *Stylized Hierarchy Establishment* layer.

4.2 Memories as Traces

In order to facilitate an evolution-inspired development style we used a model of memories as traces or echoes of existing senses. This often allowed the **Sense** circuit to be delegated to an underlying layer, leaving only the handling of the memory to the top-level layer.

In the *Stylized Hierarchy Establishment* strategy needed to analyze the behavior of the other robot over time in order to tell whether it was displaying or not. To do this, the *Stylized Hierarchy Establishment* layer kept a copy of a **Khepera Sense** circuit which was delegated down to the *Uncritical Fighting* layer. This memory was compared to the current output from the **Khepera Sense** circuit and consistent discrepancies were interpreted as motion. The display behavior was to stand still. When a robot made consistant observations of the opponent as stationary it assumed the opponent to be expressing a display behavior.

The basic setup for the *Stylized Hierarchy Establishment* layer is shown in Figure II. The **Memory** circuit is presented using an octagon.

Figure II position.

4.3 Duplication and Specialization

When adding new behaviors to a controller it can be clarifying for behavioral layer design to think in terms of evolutionary stages. Firstly, we imagine that a general, relatively uniform layer of neurons evolves which can store memories of the underlying sense- and action-related neural activity. Supported by these memories, more specialized neural structures can evolve connecting the memories to existing senses and competences. Finally, the circuits involved can be adapted and dedicated circuits added to support the new behaviors.

This way of thinking forces developers to consider what underlying structures must be in place to support new behavioral layers. This ordering of circuit addition is not to be taken as a literal recommendation to implement the intermediate stages when adding a new adaptive behavioral layer, but only as an aid during design.

Evolution is known to work at least in part, through duplication and specialization [Allman, 1999]. When biologists talk about highly evolved structures they refer to the level to which those structured have been specialized for particular purposes compared to similar structures in other species [Strickberger, 1995]. A general indication of the specificity of an structure is its complexity, so it is common to say that the human eye is more ‘highly evolved’ than the light sensitive spots on certain species of fish. Likewise the human brain is more highly evolved than the brain of most other species. This use of the expression ‘highly evolved’ must however, not be confused with closeness to human form. A cat’s whiskers are more highly evolved than human facial hair and the eye of the eagle is more highly evolved than the human eye. The idea of adding a general, uniform memory layer and then adding specialized structures to support new capabilities is inspired directly by these two aspects of evolution, replication and specialization.

During development and experimentation, we found that we were often missing the underlying structures suggested by the ordered circuit addition approach. In this case it is necessary to develop the supporting structures in more basic layers that have their own evolutionary grounding in that they implement improvements on the existing solutions.

Using ordered circuit addition to develop underlying structures generally improves the robustness of suggested solutions and makes them more plausible in an evolutionary cognitive modeling context.

5 The Foraging/Mapping Problem

Foraging, or locating and gathering resources is a fundamental problem that all animals are faced with and which has attracted a lot of attention from the robotics community [Balch, 1999, Goldberg and Matarić, 2001].

When the solution to a foraging problem includes remembering spatial structures and the positions of the resources that are being gathered, the problem is more accurately described as mapping. The discovery of *place cells* in rats, gave birth to a novel solution to the problem of spatial learning or mapping in robots based on associations between places and food [Fuhs et al., 1998, Matarić, 1992]. The different strategies for spatial learning in animals have been shown to follow a path of increasing sophistication from dead reckoning through navigation based on landmarks to navigation based on cognitive maps [Gould and Gould, 1999]. Here we present an experiment with four increasingly sophisticated foraging strategies, starting with a Random Walk and culminating in a strategy that does associative mapping similar to the place cell inspired work mentioned above.

5.1 Experimental Setup for Foraging

Our experiments used a Khepera robot with six Infrared (IR) proximity sensors and a K6300 Color Camera, simulated by the Webots[1] robot simulator. For the experiments on foraging and mapping we used the simulated environment presented in Figure III.

Figure III position.

At the start of each experiment the robot had an energy level of 1.0. Each 64 ms the energy level decreased by 0.02. When the robot was in contact with

[1]Webots is a commercial robot simulator developed by the MicroComputing and Interface Lab (LAMI) at the Swiss Federal Institute of Technology, Lausanne (EPLF) and distributed by Olivier Michel and the Cyberbotics Company who generously provided us with a student license for this work.

a feeder, the energy level rose to 2.0. If the energy level fell to zero, the robot was considered dead and removed from the environment.

5.2 The Controllers

On top of a base controller that did not have any foraging strategies, but which could do *Random Wandering*. We implemented three layers of increasingly sophisticated foraging strategies: *Structured Exploring* and *Feeding Position Approach*. The layers of the complete solution are presented in Figure IV.

Figure IV position.

5.3 The Base Controller (Random Wandering)

As a base controller we used a controller that did random wandering, avoiding obstacles and approaching feeders when visible. The base controller consisted of three simple behavioral layers. These are presented as the three bottom layers in Figure IV.

Motion This layer drives the robot constantly forward. This is done through the presence of a **Motion Drive** circuit that continuously provides the two **Wheel Actuator** circuits with a constant integer. The two **Wheel Actuator** circuits use the sum of their provided integers to set the speed of the respective wheels.

Full Feeder Approach This layer added the ability to approach full feeders. First we added a **K6300 Sensor** circuit that encapsulated the Khepera API for the simulated K6300 Color Camera. The **Full Feeder Sense Circuit** used the image provided by the **K6300 Sensor** circuit to detect the presence of a full feeder based on their unique range of greens. The **Approach Object Competence** circuit used the relative position data in the provided **Object** percept to servo the robot toward that object.

Obstacle Avoidance This layer added the ability to steer away from obstacles and hence keep moving until the robot ran out of energy. To avoid crashing into objects in the world we added two **Proximity Sensor** circuits, one for the left and one for the right side of the robot. Each **Proximity Sensor** circuit encapsulated three simulated IR sensors and provided a single integer representing the proximity of objects on the respective side. The integers from the **Proximity Sensor** circuits were used by the two lateral **Avoid Proximity Drive** circuits that again provided negative integers to the opposite **Wheel Actuator** circuits in order to steer away from any obstacles.

To allow the robot to touch a feeder we introduced the **Close Feeder Sense** circuit which takes an **Object** percept and fires if the height of the feeder is above than a set threshold, indicating that if it is a feeder, it is close to the robot. The **Close Feeder Sense** again activated a **Touch Full Feeder Drive** circuit which inhibited the **Avoid Obstacle Drive** circuit. Inhibiting the feeder whenever a feeder was visible lead to the robot crashing into obstacles while approaching the feeder.

5.4 Exploration and Mapping Strategies

Structured Exploration On top of the base controller we implemented three foraging specific behavioral layers. These are presented as the top three layers in Figure IV. The *Structured Exploration* layer implemented an exploring behavior which surveyed an area by following a hard-coded pattern through the environment, turning and accelerating according to the perceived position.

We implemented a set of four **Corner Sense** circuits which relied on he information in the image provided by the **K6300 Camera** circuit and the unique markings of each corner in the environment. The **Corner Sense** circuits passed **Object** percepts describing the relative position and height of the visible corners to the **Explore Drive** circuit. The **Explore Drive** circuit implemented a pre-defined pattern of exploration by choosing one of the **Object** percepts and passing it on to the **Approach Object Competence** circuit.

Visible Feeding Position Approach To give the robot the basic ability to return to a position where it had previously fed, we implemented a **Feeding Sense** circuit. This circuit took two excitors, the **Empty Feeder Sense** circuit and the **Full Feeder Memory** circuit. The **Full Feeder Memory Circuit** decayed rapidly, so that together these two circuits indicated whether the robot had just emptied the feeder.

Based on the **Corner Sense** circuits from the previous layer, we implemented a **Position Sense** circuit that indicated roughly in which of the four corners the robot was currently situated. Whenever the **Feeding Sense** circuit fired, it would establish a memory in the **Feeding Position Memory** circuit containing the current position and hence indicating to the robot in the future, where feeding had taken place.

We also implemented an **Energy Level Sensor** circuit which provided an integer reflecting the robot's current energy level. This we used to implement a **Low Energy Sense** circuit that fired when the energy level fell below a threshold.

Lastly we implemented an **Approach Visible Feeding Position Drive** circuit. When the **Low Energy Sense** circuit was fired, the **Approach Visible Feeding Position** circuit compared the **Object** percepts provided by the **Corner Sense** circuits with the position provided by the **Feeding Position Memory** circuit. If a match was found, the **Approach Visible Feeding Position Drive** circuit passed the relevant **Object** percept on to the underlying **Approach Object Competence** circuit and inhibited the **Explore Drive** circuit. This lead to the robot abandoning exploration and heading straight for the corner in which it had fed whenever that corner became visible and the robot's energy was low.

General Feeding Position Approach This layer generalizes the ability to approach places where feeding has previously taken place. The *Visible Feeding Position Approach* strategy could only approach such places when they were in sight. The *General Feeding Position Approach* strategy adds the ability to turn toward these positions when they are not visible.

The only circuit we added to generalize the *Visible Feeding Position Approach* strategy to a *General Feeding Position Approach* strategy, was the **Approach General Feeding Position Drive** circuit. This circuit was excited by the **Low Energy Sense** circuit. It got one **Position** percept from the **Position Sense** circuit and the one **Position** percept from the **Feeding Memory** circuit. From this data it calculated where to turn to in order to bring the corner where the feeding took place back into view. The **Approach General Feeding Position Drive** circuit was inhibited by the **Approach Visible Feeding Position Drive** circuit so that when the corner where the feeding took place came into view, the robot would use the underlying *Approach Visible Feeding Position* strategy to take it to the feeder.

5.5 Results from the Foraging Experiments

The Random Wandering behavior implemented by the Base Controller relied on chance to reach different areas of the environment. This could be very inefficient. Depending on the differences produced by imperfect sensors and actuators, two runs following the *Random Wandering* strategy could have wildly varying paths. There were however similarities between many of the paths e.g. one group reached the north wall in Figure V with an orientation less π, hence turning right, another group reached the north wall with an orientation greater than π, hence turning left.

In order to compare the quality of the *Random Wandering* strategy to the *Structured Exploration* strategy, we ran 20 trials with each of the controllers and measured the average *feeding rate*, i.e. the number of times the robot reached the feeder during two minutes. For *Random Wandering*, the mean feeding rate was 1.5/2 minutes with a standard deviation of 0.95. A typical path taken by a robot following the *Random Wandering* strategy is indicated by the dotted line in Figure V.

As indicated by the dashed arrow in Figure V, the pre-defined exploration pattern for the *Structured Explore* strategy was a square along the boundaries of the environments. This path brought the robot back to the feeder at regular intervals. Over 20 trials, the mean feeding rate for robots following the *Structured Exploration* strategy was 2.7/2 minutes with a standard deviation of 0.5. This feeding rate is significantly higher than the feeding rate for the *Random Wandering* strategy on a 99% confidence level.

The thin solid arrow in Figure V shows that when following the *Approach Visible Feeding Position* strategy. able to head for the position where the feeding took place as soon as it came into view. These circuits shortened the average length of the path the robot followed to return to the feeding position. As a result the feeding rate increased. Over 20 trials, the mean feeding rate for robots following the *Visible Feeding Position Approach* strategy was 3.0/2 minutes with a standard deviation of 0.0. This feeding rate is significantly higher than the feeding rate for the *Structured Exploration* strategy on a 99% confidence level.

Our final strategy for foraging was the *Approach General Feeding Position* which allowed the robot to head for the position where the feeding took place any time the **Low Energy Sense** circuit was firing. A typical path taken by the robot when following the Approach General Feeding Position strategy is indicated by the thick arrow in Figure V. It illustrates how the *Approach General Feeding Position* strategy on cut short the path produced by the *Approach Visible Feeding Position* strategy. Over 20 trials, the mean feeding rate for robots following th *General Feeding Position Approach* strategy was 3.6/2 minutes, with a standard deviation of 0.5. This feeding rate is significantly higher than the feeding rate for the *Structured Exploration* strategy on a 99% confidence level.

6 The Conflict Resolution Problem

When two entities compete for a limited resource, there are a number of increasingly sophisticated conflict resolution strategies available to the participants. Here we present implementations of four such strategies and demonstrate through experiments, the consistent improvement in performance produced by each new layer.

6.1 Experimental Setup

The simulated environment in which our experiment on conflict resolution took place is identical to the environment used for the experiment on foraging but with an additional Khepera robot.

For this experiment we added a *strength* value and an *injury* value to the energy value related to the Khepera robot had in the foraging experiments. The two robots were given different and unchangeable strength values from the start. We refer to the robot with the highest strength value as the strong robot and the robot with the lowest strength value as the weak robot.

Injury was accrued whenever the two robots were within a given proximity of each other. As long as the robots remained within close proximity of each other, more injury was received every 64 ms. The added injury was identical to the strength of the other robot. The initial injury level was 0.0 and if a robot's injury exceeded a given threshold, the robot was considered dead and removed from the environment.

The measures used to evaluate the performance of the conflict resolution strategies below was the survival time. The energy reduction rate was for this experiment, set to 0.0, implying that none of the robots would die from low energy levels in these experiments.

6.2 The Conflict Resolution Strategies

We designed four increasingly sophisticated strategies for conflict resolution using HEP and added corresponding behavioral layers on top of the *General Feeding Position Approach* controller presented in detail above. These are presented presented in Figure VI.

Figure VI position.

The Base Controller (Avoidance) The *General Feeding Position Approach* controller generally brought the two robots back to the same place and created ample conflict situations. The robots, however, were oblivious to each other's existence and treated each other as general obstacles to be avoided.

Uncritical Fighting The *Uncritical Fighting* strategy was a simple, reactive strategy for conflict resolution. It had the robots attacking each other on sight.

To implement this behavior, a **Khepera Sense** circuit was added which recognized the color of a simulated Khepera robot in an image from the **K6300 Camera Sensor** circuit and produce an **Object** percept describing the size and relative position of the observed Khepera. The presence of a Khepera Robot excited an **Uncritical Fighting Drive** circuit which inhibited all the underlying drives and passed the **Object** percept describing the Khepera Robot to the **Approach Object Competence** circuit.

Reactive Yielding This strategy added the ability to yield, i.e. to stop fighting, whenever a robot sensed that it was the weaker of the two robots. We implemented a **Strength Sensor** circuit to give the robot access to its own strength value and an **Injury Sensor** circuit that reported the severity of new injuries, indirectly reporting the opponent's strength. We used the information from these two sensor circuits to implement a **Weak Sense** circuit that fired only if the injury received was greater than the robot's strength, i.e. when the robot was weaker than its opponent. Finally we implemented a **Reactive Yielding Drive** circuit which was activated by the **Weak Sense** circuit. The **Reactive Yielding Drive** circuit inhibited the **Uncritical Fighting Drive** circuit presented above, letting the weak robot steer away from the strong one.

Pecking Order Adherence Building on the *Reactive Yielding* strategy we introduced the circuitry necessary to establish a pecking order that would stop the weak robot from attacking the strong one. The only added circuit was the **Weak Memory** circuit. This memory was established whenever the **Weak Sense** circuit in the *Reactive Yielding* layer, fired. This memory circuit inhibited the **Uncritical Fighting Drive** circuit so that the weak robot would always avoid the strong robot after an initial encounter.

Stylized Hierarchy Establishment As our ultimate solution to conflict resolution, we implemented a *Stylized Hierarchy Establishment* strategy that uses a display behavior instead of direct interaction to establish the relationship of strength between the two robots.

We implemented standing still as the display behavior. In order to see whether the other robot was standing still we implemented an **Observe Drive** circuit. This inhibited all underlying drives when the other robot was in sight. To see whether the other robot was displaying, we implemented a **Khepera Memory** circuit that remembered the last **Object** percept provided by the **Khepera Sense** circuit and provided the stored **Object** percept next time it was triggered. By comparing the **Khepera Memory** circuit and the **Khepera Sense** circuit we could find out whether the other robot was moving or standing still. If the two were the same, the other robot was standing still. If they were different, the other robot was moving. We added the **Display Sense** circuit to do this comparison.

To keep a display behavior active for an amount of time that corresponded to the robot's strength, we added a **Display Memory** circuit. This memory could only be established once within a given time period and had a strength that corresponded to the value provided by the **Strength Sensor** circuit. It decayed slowly so that the time it would take to reach zero, i.e. to be forgotten, corresponded directly to the robot's strength.

To sense whether the robot was strong or weak, we implemented the **Weak Display Sense** circuit. If the **Display Sense** circuit was still active when the display period was over, this would indicate that the other robot was displaying for a longer time and hence was stronger. Having the **Weak Display Sense** circuit being excited by the **Display Sense** circuit, but inhibited by the **Display Memory** circuit mirrored these circumstances. The excitation of the **Weak Display Sense** then established a memory in the **Weak Memory** circuit which again was made to inhibit further displays.

6.3 Results from the Conflict Resolution Experiments

The *Uncritical Fighting* strategy allowed the strong robot to kill off the weak one and significantly increase the number of times it fed when compared to the *Avoidance* strategy. The average survival time for the weak robot during these experiments was 63.4 seconds, with a standard deviation of 25.0 seconds.

The position and orientation of the two robots during a typical trial following the *Uncritical Fighting* strategy is plotted in Figure VII with the strong robot represented by a solid line and the weak robot represented by a dashed line.

Figure VII position.

The *Reactive Yielding* strategy reduced the rate at which the weaker robot sustained injury by unilaterally aborting its fighting behavior. The average survival time for the weak robot following the *Reactive Yielding* strategy was

73.9 seconds with a standard deviation of 37.1 seconds. The student t-test indicates that on a 75% confidence level, the survival time is consistently longer for robots that follow the *Reactive Yielding* strategy than it is for robots that follow the *Uncritical Fighting* strategy. The position and orientation of the two robots during a typical trial following the *Reactive Yielding* strategy is plotted in Figure VIII

Figure VIII position.

By learning its place in the pecking order and using this information to avoid the strong robot, the weak robot significantly increases its lifespan, represented by the number of times it is able to feed. The average survival time for the weak robot following the *Pecking Order Adherence* strategy was 95.4 seconds with a standard deviation of 32.95 seconds. The student t-test indicates that on a 95% confidence leveli, the survival time is consistently longer for robots that follow the *Pecking Order Adherence* strategy than it is for robots that follow the *Reactive Yielding* strategy. The plots for the X position, the Z position and the orientation of the two robots using the *Pecking Order Adherence* strategy are shown in Figure IX.

Figure IX position.

The average survival time for the weak robot using the *Stylized Hierarchy Establishment* strategy was 104.6 seconds with a standard deviation of 54.7 seconds. The student t-test indicates that, on a 60% confidence level, the survival time is consistently longer for robots that follow the *Stylized Hierarchy Establishment* strategy than it is for robots that follow the *Pecking Order Adherence* strategy. The plots for the X position, the Z position and the orientation of the two robots using the *Stylized Hierarchy Establishment* strategy are shown in Figure X.

Figure X position.

7 Conclusions and Future Work

Our experiments show that complex adaptive robot control algorithms can be developed using HEP. The results also show that the algorithms adapt quickly, with a minimum of experience. This demonstrates that 'bottom-up' approach to learning can provide solutions to problems where traditional Machine Learning algorithms such as Reinforcement Learning night not be appropriate due to slow convergence rates.

Today, learning is typically applied in unknown or highly-complex domains with dynamics that are difficult or impossible to model. HEP algorithms cannot at the moment, be used for this kind of system optimization through blind search.

There are several theories available on the evolution of memory and learning [Moore, 1996, Johnson and Hirst, 1993, Arbib, 2000]. We aim to build on the initial results presented here to develop more complex adaptive mechanisms by taking inspiration from the evolutionary history of learning. Increasing the adaptivity of our algorithms also increases their general applicability. Our goal is to find a way of doing this while avoiding a combinatorial explosion in the problem space.

To reproduce, in robots, the pervasive adaptivity found in animals, it is necessary to replace the procedural implementation of NCs with a more adaptive ANN-based implementations. We will be looking at different ways of achieving this without loosing the expressiveness of procedural languages.

Acknowledgments

This work was done at the Machine Learning Research Group in the Department of Computer Science, University of Bristol. It was sponsored in part by Hewlett-Packard Laboratories, Bristol.

Figure Captions:

- Figure I Circuits for Uncritical Fighting
- Figure II Circuits for Adaptive Behavior
- Figure III Webots Environment for Foraging and Mapping
- Figure IV The Behavioral Layers of the Base Controller
- Figure V Paths Taken by the Different Foraging Controllers
- Figure VI Behavioral Layers for Conflict Resolution
- Figure VII Positions and Orientations for Uncritical Fighting
- Figure VIII Positions and Orientations for Reactive Yielding
- Figure IX Positions and Orientations for Pecking Order Adherence
- Figure X Positions and Orientations for Stylized Hierarchy Establishment

References

[Allman, 1999] Allman, J. M. (1999). *Evolving Brains.* Scientific American Library.

[Andre and Russel, 2001] Andre, D. and Russel, S. J. (2001). Programmable Reinforcement Learning Agents. In *Advances in Neural Information Processing Systems 13, Proceedings of the 13th Conference on Neural Information Processing Systems*, pages 1019–1025. MIT Press.

[Arbib, 2000] Arbib, M. (2000). The Mirror System, Imitation, and the Evolution of Language. In Nehaniv, C. and Dautenhahn, K., editors, *Imitation in Animals and Artifacts*, pages 111–666. MIT Press.

[Arbib, 1989] Arbib, M. A. (1989). *The Metaphorical Brain 2.* Wiley.

[Arkin, 1989] Arkin, R. C. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112.

[Asada et al., 1995] Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1995). Vision-based reinforcement learning for purposive behavior acquisitions. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'95)*, pages 146–153.

[Balch, 1999] Balch, T. R. (1999). The impact of diversity on performance in multi-robot foraging. In Etzioni, O., Müller, J. P., and Bradshaw, J. M., editors, *The proceedings of the Third International Conference on Autonomous Agents*, pages 92–99, Seattle, WA. ACM Press.

[Beer et al., 1998] Beer, R. D., Chiel, H. J., Quinn, R. D., and Ritzmann, R. E. (1998). Biorobotic approaches to the study of motor systems. *Current Opinion in Neurobiology*, (8):777–782.

[Breazeal and Scassellati, 1998] Breazeal, C. and Scassellati, B. (1998). Infant-like social interactions between a robot and a human caregiver. *Adaptive Behavior*, 8(1):49–74.

[Brooks, 1986] Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, (2).

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without Representation. *Artificial Intelligence*, (47):139–159.

[Bryson, 2001] Bryson, J. J. (2001). *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents.* PhD thesis, Massachusets Institute of Technology, Department of Electrical Engineering and Computer Science.

[Bryson and Stein, 2001] Bryson, J. J. and Stein, L. A. (2001). Modularity and specialized learning: Mapping between agent architectures and brain organization. In Wermter, S., Austin, J., and D.Willshaw, editors, *Emergent Neural Computational Architectures Based on Neuroscience*, pages 98–113. Springer Verlag.

[Carlson, 2000] Carlson, N. R. (2000). *Physiology of Behaviour*. Allyn and Bacon, seventh edition.

[Cliff, 1991] Cliff, D. T. (1991). Computational neuroethology: A provisional manifesto. In Meyer, J. A. and Wilson, S. W., editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB'91)*, pages 29–39, London, England. MIT Press.

[Corbacho and Arbib, 1997] Corbacho, F. J. and Arbib, M. A. (1997). Schema-based learning: Biologically inspired principles of dynamic organization. In Mira, J., Moreno-Díaz, R., and Cabestany, J., editors, *Biological and Artificial Computation: From Neuroscience to Technology, Proceedings of the International Work-Conference on Artificial and Natural Neural Networks (IWANN'97), LNCS 1240*, pages 678–687, Lanzarote, Spain.

[Dorigo and Colombetti, 1998] Dorigo, M. and Colombetti, M. (1998). *Robot Shaping*. MIT Press.

[Fuhs et al., 1998] Fuhs, M. C., Redish, A. D., and Touretzky, D. S. (1998). A visually driven hippocampal place cell model. In Bower, J., editor, *Computational Neuroscience: Trends in Research*, pages 101–106. Plenum Publishing.

[Goldberg and Matarić, 2001] Goldberg, D. and Matarić, M. J. (2001). Design and evaluation of robust behavior-based controllers for distributed multi-robot collection tasks. In Balch, T. and Parker, L. E., editors, *Robot Teams: From Diversity to Polymorphism*, pages 315–244. A K Peters Ltd.

[Gould and Gould, 1999] Gould, J. L. and Gould, C. G. (1999). *The Animal Mind*. Scientific American Library.

[Johnson and Hirst, 1993] Johnson, M. K. and Hirst, W. (1993). Mem: Memory subsystems as processes. In Collins, A. F., Gathercole, S. E., Conway, M. E., and Morris, P. E., editors, *Theories of Memory*, pages 241–286. Lawrence Erlbaum Associates.

[Kaelbling, 1993] Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press.

[Maes, 1994] Maes, P. (1994). Modeling adaptive autonomous agents. *Artificial Life Journal*, 1(1–2):135–162.

[Maes and Brooks, 1990] Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviours. In *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI'90)*.

[Mahadevan and Connell, 1992] Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):304–312.

[Matarić, 1992] Matarić, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312.

[Matarić, 1994a] Matarić, M. J. (1994a). *Interaction and Intelligent Behavior*. PhD thesis, Massachusets Institute of Technology.

[Matarić, 1994b] Matarić, M. J. (1994b). Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*.

[Matarić, 1997] Matarić, M. J. (1997). Reinforcement Learning in the Multi-Robot Domain. *Autonomous Robots*, 4(1):73–83.

[McFarland and Bösser, 1993] McFarland, D. and Bösser, T. (1993). *Intelligent Behavior in Animals and Robots*. MIT Press.

[Moore, 1996] Moore, B. R. (1996). The Evolution of Imitative Learning. In Heyes, C. M. and Galef, B. G., editors, *Social Learning in Animals: The Roots of Culture*, pages 245–265. Academic Press.

[Morén and Balkenius, 2000] Morén, J. and Balkenius, C. (2000). A Computational Model of Emotional Learning in the Amygdala. In Meyer, J.-A., Berthoz, A., Floreano, D., Roitblat, H. L., and Wilson, S. W., editors, *From Animals to Animats 6: Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior (SAB'2000)*, Paris, France. MIT Press.

[Nehaniv et al., 1999] Nehaniv, C. L., Dautenhahn, K., and Loomes, M. J. (1999). Constructive Biology and Approaches to Temporal Grounding in Post-Reactive Robotics. In *Sensor Fusion and Decentralized Control in Robotics Systems II, Proceedings of the International Society for Optical Engineering (SPIE)*, number 3839, pages 156–167.

[Nicolescu and Matarić, 2001] Nicolescu, M. and Matarić, M. (2001). Experience-based representation construction: learning from human and robot teachers. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robots and Automation (CIRA'01)*, pages 463–468, Banff, Canada.

[Russel and Norvig, 1995] Russel, S. and Norvig, P. (1995). *Artificial Intelligence, A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall.

[Stone, 2000] Stone, P. (2000). *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press.

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork. *Artificial Intelligence*, 110(2):241–273.

[Strickberger, 1995] Strickberger, M. W. (1995). *Evolution.* The Jones and Bartlett Series in Biology. Jones and Bartlett Publishers, second edition.

[Weitzenfeld et al., 2000] Weitzenfeld, A., Peguero, O., and Gutiérrez, S. (2000). Nsl/asl: Distributed simulation of modular neural networks. In *Proceedings of the Mexican International Conference on Artificial Intelligence (MICAI'2000), Lecture Notes in Artificial Intelligence 1793*, pages 326–337, Acapulco, Mexico.

Figure I:

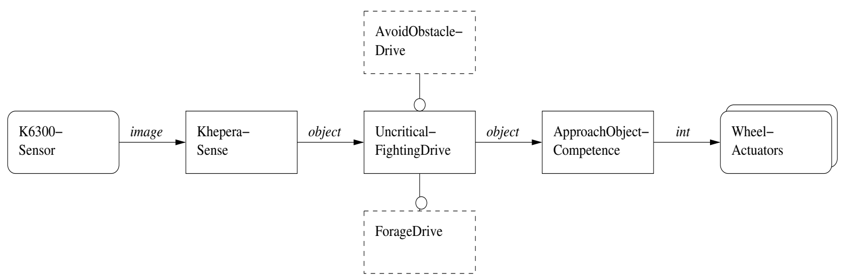

Figure II:

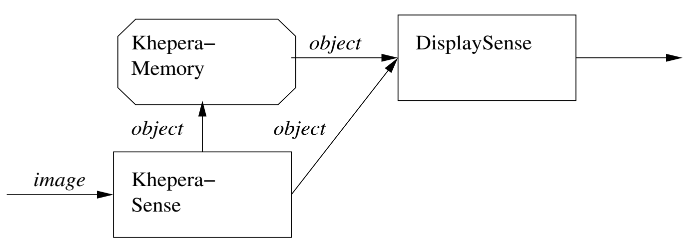

Figure III:

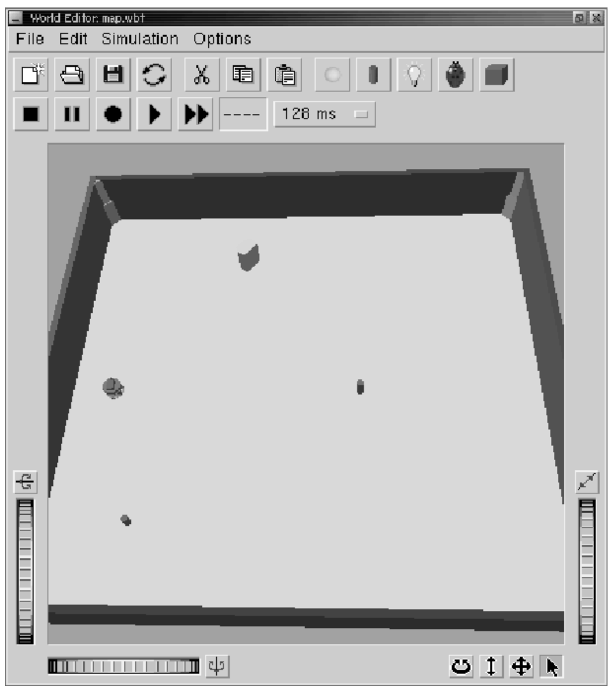

Figure IV:

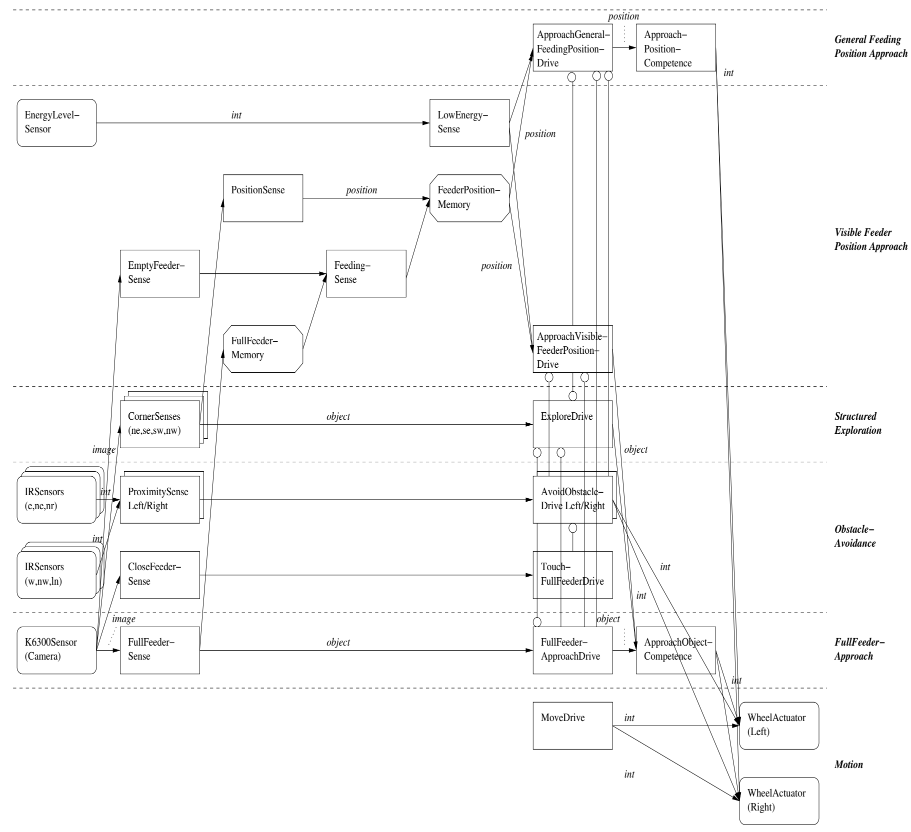
position
ApproachGeneral–FeedingPosition–Drive
Approach–Position–Competence
int
General Feeding Position Approach
EnergyLevel–Sensor
int
LowEnergy–Sense
position
PositionSense
position
FeederPosition–Memory
Visible Feeder Position Approach
EmptyFeeder–Sense
Feeding–Sense
position
FullFeeder–Memory
ApproachVisible–FeederPosition–Drive
CornerSenses (ne,se,sw,nw)
object
ExploreDrive
Structured Exploration
image
object
IRSensors (e,ne,nr)
int
ProximitySense Left/Right
AvoidObstacle–Drive Left/Right
Obstacle–Avoidance
int
IRSensors (w,nw,ln)
int
CloseFeeder–Sense
Touch–FullFeederDrive
int
image
object
K6300Sensor (Camera)
FullFeeder–Sense
object
FullFeeder–ApproachDrive
ApproachObject–Competence
FullFeeder–Approach
int
MoveDrive
int
WheelActuator (Left)
Motion
int
WheelActuator (Right)

Figure V:

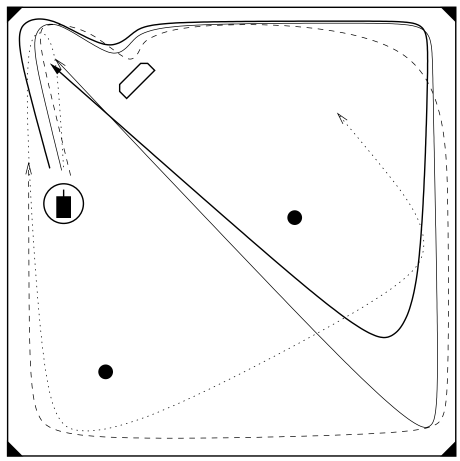

Figure VI:

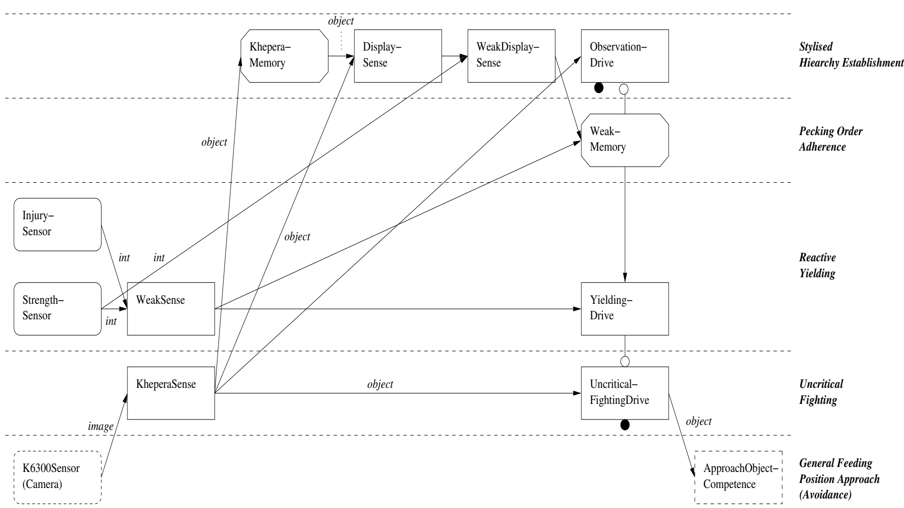

Figure VII:

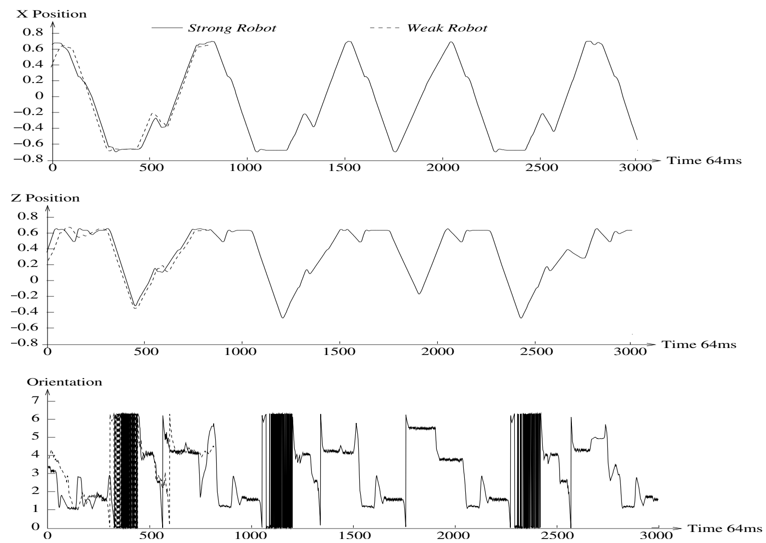

Figure VIII:

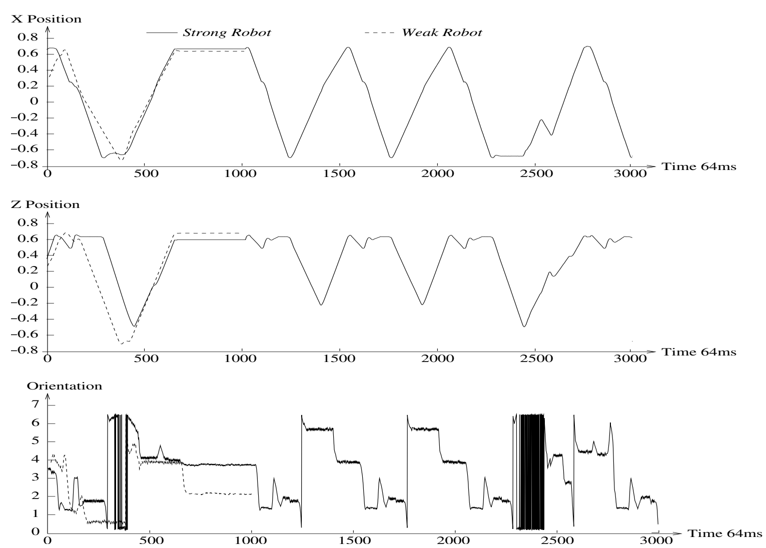

Figure IX:

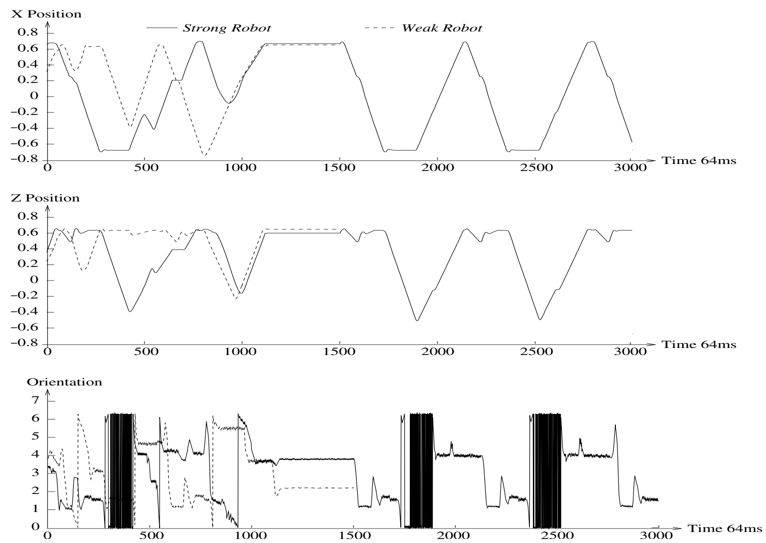

Figure X: