Submitted to *Adaptive Behavior*
July, 2004

Incremental Development of Adaptive Behaviors using Trees of Self-Contained Solutions

Torbjørn S. Dahl
Norwegian Defence Research Establishment
Postboks 25, 2027 Kjeller, Norway
`phone: +47 63807782`
`fax: +47 63807715`
`Torbjorn-Semb.Dahl@ffi.no`

Christophe Giraud-Carrier
ELCA Informatique SA
Lausanne 13, Switzerland
`cgc@elca.ch`

Running Head

Incremental Development of Adaptive Behaviors

Abstract

We present a simple methodology for incremental development of complex adaptive behaviors in robots. The methodology decomposes a given root strategy into a tree of self-contained supporting strategies that can be fully implemented and tested before the next strategy is added. The methodology also uses comparative performance tests for each new strategy relative to its predecessor. The methodology assumes the use of skill modules that can be shared by multiple behavioral layers and produces learning mechanisms that are highly specialized and context dependent. Two example applications of the methodology are presented using simulated robots in the domains of foraging/mapping and conflict resolution. The examples are implemented by hand using a decomposed model of behavior that allows skill modules to be shared while retaining a unique representation of each strategy for excitation and inhibition. Finally we discuss how the solutions produced using this methodology differ from existing behavior-based solutions.

Keywords

Incremental Development, Learning, Foraging, Mapping, Conflict Resolution

1 Introduction

Incremental development is a well-known approach in Software Engineering (SE) [Larman and Basili, 2003] and prominent idea in Behavior-Based (BB) systems [Brooks, 1986, Bryson, 2001]. Incremental development has also been used as a way of facilitating automated development through Machine Learning (ML). *Evolutionary Robotics* (ER) [Nolfi and Floreano, 2000] and *Epigenetic* or *Developmental Robotics* (DR) [Weng et al., 2000] are two such automated development strategies both inspired by incremental processes found in nature. However, currently existing research on BB systems contains relatively little work on top-down design methodologies that can support incremental development.

We have produced a simple incremental SE methodology based on the idea that partial solutions to a given problem can themselves be self-contained systems. They can either be solutions of lower quality than the final solutions or they can be solutions to a sub-problem related to the given problem. We used the methodology to develop adaptive behaviors for foraging, mapping and conflict resolution and studied the restrictions and possibilities this approach entailed. Here we present the main contributions of this work to the area of adaptive behavior.

The methodology decomposes a given problem solving strategy into a complete tree of self-contained, testable, partial solutions, with minimal incremental steps between each solution. The methodology works by recognizing key perceptual and behavioral skills required by the target strategy. These skills can represent salient features of immediate and previous experience or general capabilities such as taxis. The implementation of the required skills are delegated, as much as possible, to a number of *supporting strategies* that make them available across behavioral layers through an abstraction mechanism such as *Robot Schemas* [Lyons and Arbib, 1989].

To support our methodology we have developed a decomposition of the layered control structures common in BB controllers. This decomposition is also presented in this article. Our behavioral components support the abstraction and reuse of perceptual and behavioral skills while retaining a unique presence for every behavioral strategy in order to support subsumption-style action selection through excitation and inhibition.

The solutions produced using the methodology presented have three main strengths. First, they employ multiple strategies of different sophistication concurrently to solve a given problem. During learning and in the case of limited hardware failures, such algorithms can retain a higher level of performance than algorithms that use a single strategy. Second, the learning that takes place in the strategies developed is highly restricted. Highly restricted learning allows the algorithms to adapt quickly after minimal amounts of experience. Third, the solutions reuse general skill modules. This leads to less implementation work, fewer bugs and more compact systems.

For clarity, we use the term *behavior* to denote only externally observable displays. We call the solution to a problem provided by certain behaviors a *behavioral strategy* or just *strategy*. We use the term *behavioral layer* or *layer* to denote the internal structures of a robot involved in supporting a specific behavioral strategy. Controllers are generally referred to using the most sophisticated behavioral layer it contains, e.g., the pecking order adherence controller. A behavioral layer implements a partial mapping from a set of *stimuli* to a set of *responses*. We would like to emphasize that we use the term *learning* to describe any change in this mapping as a result of experience. In our terms an entity that recognizes its level of strength relative to a competitor has done learning.

2 Motivation

In SE it has long been recognized that incremental development reduces the risk of spending effort on undesirable or infeasible designs [Larman and Basili, 2003]. The popularity of approaches such as *Extreme Programming* (XP) [Beck, 1999] and the broader class of related methods called *Agile Methods* [Cockburn, 2001] illustrates the importance of this reduction in risk. People with experience in *Open Source Development* [Raymond, 1999] also emphasize the importance of incremental development as a way to expose bugs and bad design. Broadly speaking incremental development is: '*feedback driven refinement with customer involvement and clearly delineated iterations*' [Larman and Basili, 2003].

In a learning system, it is generally desirable to have a high initial performance level that increases gradually rather than a performance level that remains close to that of random actions until a certain amount of information has been gathered. Correspondingly, it is desirable for a system to display *graceful degradation*, i.e., a gradual loss of performance rather than complete failure in a system. Evidence from biology [Gould and Gould, 1999], indicates that animals achieve these two things by having available to them a spectrum of increasingly sophisticated behavioral strategies for solving given problems. There is also evidence that the presence of simple strategies based on simple forms of learning, such as Pavlovian conditioning, are prerequisites for more sophisticated strategies based on more advanced forms of learning such as imitation [Moore, 1996]. Consider an animal that competes with other 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 sophisticated conflict resolution strategies such as, yielding, adherence to pecking order, and coalition building. Strategies of different sophistication will necessarily depend on varying levels of experience and have different execution time. Depending on the animals experience and situation, a subset of these strategies might be applicable.

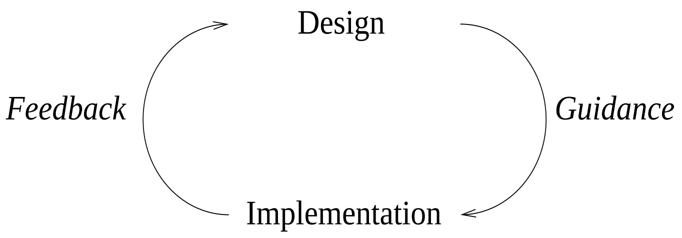
BB architectures divides complex control problems into a set of competing behaviors. Traditionally there are different behaviors for different problems such as wall following, obstacle avoidance and taxis. However, in order to support incremental development and provide robust and fast-learning controllers, we use different behaviors that solve the *same* problem. According to the resources and experience available to a robot, it applies different behaviors, each constituting a different solution to the same problem. When each of these strategies can also function independently, they provide partial but self-contained solutions to the original problem, i.e., they are delineated iterations for incremental development.

In order to bring together the benefits of incremental development and the advantages of having multiple concurrent strategies for the same problem, we developed a methodology for the incremental development of complex adaptive behaviors using trees of self-contained solutions. The methodology is a set of recommendations related to the order, the structure and the content of a series of partial solutions leading up to a final desirable robot behavior. The recommendations are presented here together with a general decomposition of a behavioral layer that supports the kind of skill abstraction necessary. Our goal with this work is to provide and demonstrate a functional methodology that can benefit everyone who are doing incremental development in the area of BB systems, including those using ML-based approaches.

3 Methodology

Incremental development describes a process of interaction between a system's design and its implementation. This process runs over a number of iterations where the design guides implementation and where implementation and testing provides verification of and improvement on, the design. The process is illustrated graphically in Figure I.

Figure I: The Incremental Development Process



Our approach to the incremental development of arbitrary behavioral strategies assumes that it is possible to construct a tree of supporting strategies connecting the top-level or *root strategy* to a set of simpler starting points or *leaf strategies*. We call such a tree a *strategy tree*. We have formalized the main elements of the construction of strategy trees into a set of rules for incremental development of complex adaptive behaviors.

1. **Design**

   Starting with a strategy tree containing only the root strategy, for each non trivial strategy in the tree, we do the following:

- *Skills* - Identify the perceptory and behavioral skill requirements
- *Support* - Identify supporting strategies to be added as new branches to the strategy tree, delegate skills, refine requirements
- *Interaction* - Identify the inhibition structure and shared schemas
- *Evaluation* - Specify an evaluation metric that captures the advantage of this strategy and define an environment in which the advantage is expressed

2. **Implementation**

   From the bottom up, for each strategy in the complete path:

   - *Feasibility* - Test the feasibility of the design through implementation
   - *Viability* - Test the viability of the strategies through experimental comparison of performance
   - *Revision* - Revise the design according to feedback from implementation and testing

3.1 The Design Phase

Our methodology starts with a strategy tree containing only the envisioned top-level strategy for solving a given problem. Each strategy in the tree that is not directly implementable is expanded by adding a set of supporting strategies as leaves until all the leaf strategies are directly implementable. Each supporting strategy should implement one or more of the skills required by the supported strategy. Each supporting strategy should itself be a self-contained solution to the given problem or one of its sub-problems. By 'self-contained strategy' we mean a strategy that can be implemented and evaluated without implementing its sibling strategies or its parent strategy.

Skills We have found that identifying the perceptory and behavioral skills required to support a given behavioral strategy can be done using simple use-cases [Pressman, 2001]. The use-cases should describe the different behaviors of the strategy in different scenarios and the criteria that that decides which behavior should be applied. The behaviors described in the use-cases are roughly the behavioral skill requirements. The perceptual skill requirements are the decision criteria.

Using an animal display strategy as an example we get the following use-case: An animal recognizes a competitor puts on a display, recognizes whether the competitor is stronger or weaker and flees or attacks accordingly. From this scenario we get the following perceptory skill requirements: *opponent recognition* and *display interpretation*. We also get the following behavioral skill requirements: *displaying*, *fleeing* and *attacking*.

Support A supporting strategy is a child strategy that implements some of the perceptory or behavioral skills identified in the previous step. By delegating as many of the required sensory and behavioral skills as possible to supporting layers, we minimize the incremental step between the strategies. This facilitates development by reducing the risk of producing infeasible solutions during design and the risk of producing bugs during implementation. Skills that can not be delegated to supporting strategies must be implemented in a layer dedicated to the supported strategy.

To identify supporting strategies we look at the skill requirements identified in the previous step and try to come up with solutions to the given problem that use some of the required skills but that perform a litte worse than the supported strategy or solves only a sub-problem of the given problem. The design process often goes through a number of iterations and good supporting strategies can suggest beneficial adjustments to the supported strategies with corresponding refinements of the original skill requirements.

In the case of stylized displays, we were able to delegate the fleeing and attacking behaviors to a simpler conflict resolution strategy based on direct interaction rather than displays. That strategy would also implement the opponent recognition skill. This left the display and display interpretation skills to be implemented by the stylized display layer. The display interpretation used memories of previously recognized opponents to analyze their behavior over time. These memories were copies of the percepts produced by the opponent recognition skill.

A Note on Memory and Learning In general we have found it helpful to restrict memories to be copies or *echoes* of percepts that are themselves further delegated. Strategies often have historical records of certain percepts as a part of their perceptual requirements. In these cases it is possible that a supporting strategy can be found that uses the required percept directly so that only a memory schema is needed to provide the requirement of the parent strategy. These kinds of memories are generally called *deictic variables*, and assigning values to such variables is called *imprinting*.

Often, all that is needed by a new strategy is a record of whether a certain event has happened. One example is the record of whether an opponent has been recognized as stronger. To provide the underlying percept for this memory we came up with a supporting strategy which assumed that the opponent was weaker and fled only when it could sense directly that the opponent was stronger. Keeping a record of whether this sense had fired and refraining from attacking stronger opponents was clearly a superior strategy. This kind of one-shot learning is typical of the solutions produced using this methodology and it provides a very efficient learning mechanism.

Interaction Supporting and supported strategies can both be applicable in certain situations. A supported strategy often provides a new behavior for a subset of the states in which the supporting strategy was applicable. To allow strategies to be expressed at the right time it is necessary to identify their interactions in terms of excitation and inhibition.

The stylized display strategy needed to inhibit the underlying layers when an unknown robot was visible in order to evaluate the relative strength of the other robot though mutual displays. In all other states control was yielded to the underlying layers.

Evaluation Each node on the strategy tree provides an improved solution to a problem relevant to the root strategy. For each new strategy we suggest defining a measure of performance that can objectively discriminate between the quality of the child and the parent strategies. This metric is used during the implementation phase as an indication of progress being made toward the final strategy.

The performance metric for the stylized display strategy was the robot's survival time. This was expected to increase as the robot was able to preserve some of the energy previously spent on fighting.

3.2 The Implementation Phase

Whereas the design phase is mainly top-down, the implementation phase is strictly bottom-up. The strategy tree provides abstract descriptions of solutions that are not necessarily robust or efficient enough to work in situated systems. In order to avoid flights of fancy we suggest that each strategy is fully implemented and tested for performance before any work is done on higher-level strategies.

Test Feasibility through Implementation Implementation tests feasibility and allows non-comparative performance tests. This form of verification is closely aligned with the practices in SE approaches such as XP, Agile Methods and Open Source Development. The prerequisite for using a *frequent release* approach is *testability*. By striving for self-contained strategies, we design a series of testable systems. This gives us the possibility of doing frequent reality checks and improvements of our designs. Such reality checks reduces the risk of investing in infeasible or undesirable solutions.

Test Viability through Experimental Comparison This evaluation step corresponds closely to fitness testing in evolutionary approaches where improved fitness is the driving force of development. In developmental approaches and approaches based on hand-coding comparative testing is not strictly a necessary step. Comparative testing does however provide an indication that progress is being made. Thinking in terms of improving fitness also opens up for inspiration from biological systems.

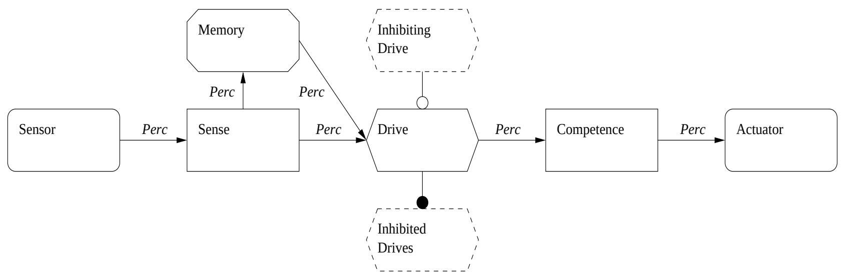
Revise Path Implementation invariably produces new ideas for strategy decomposition both for feasible and infeasible designs. When improvements to existing designs become apparent, the designs should be updated or *refactored* accordingly.

4 Decomposing Behavioral Layers

In order to reuse skill modules and support action selection based on excitation and inhibition, we decomposed behavioral layer into a collection of schemas. The schemas can send and receive structured data referred to as *percepts*. A schema requests data from other schemas only when it has received a certain amount of *activation*. When activated, our schemas will, typically request data. According to the the data received, the schema will either *fire* or not. When a schema fires, it sends activation to a set of registered receivers. It has, in this case, typically produced a further abstraction of the received data which it again provides to other schemas on request. This separation of activation and data allowed us to limit the amount of computation done and keep down execution time. The generalized communication and the separation of the control- and data-flows makes schemas well suited for procedural implementation of pre-defined behaviors. To emphasize, a schema does not have to contain an ANN. In our examples the functionality of schemas are provided by procedural implementations.

We have found it beneficial to divide schemas into six classes: sensor schemas, sense schemas, drive schemas, memory schemas, competence schemas, and actuator schemas. A typical configuration of the different schema classes within a behavioral layer is presented graphically in Figure II. For simplicity we use one arrow to indicate the presence of either or both an activation and a data channel. When a data channel is present, the arrow is labeled with the percept communicated. In accordance with the notation used in the original subsumption architecture [Brooks, 1986] inhibitions are indicated by lines with a circle in one end. Another notation we use is a black dot connected to a drive. This indicates the inhibition of all underlying drives. Schemas from other behavioral layers are presented as dashed.

Figure II: The Components of a Behavioral Layer



Sensor schemas and actuator schemas encapsulate low-level application programming interfaces (APIs). These schemas are presented graphically as boxes with rounded corners. Sense schemas and competence schemas are high-level sensor and actuator abstractions respectively. These schemas are presented graphically as square boxes. The reason for creating a dedicated schema for a particular sense or capability was to remove duplicity. It was clear from our designs that in addition to the underlying APIs, certain general perceptory and behavioral capabilities would be needed by several behavioral layers. By separating out these capabilities we let different behavioral layers share access to them and saved ourselves from having to reproduce their functionality. Memory schemas are used to store data. In order to study the role of memory in adaption we have imposed the restriction that only memory schemas may store their state between activations. This restriction emphasizes the adaptive elements of behaviors but is somewhat of an overkill in simple cases. Last, drive schemas handle the integration of sense and competence schemas within a behavioral layer. Drive schemas also handle the interaction between behavioral layers through inhibition. A drive schema is a unique representation of a behavioral layer and as such it cannot be shared. Each behavioral layer can contain multiple sensor, sense, competence, actuator, and memory schemas that may be shared with other layers. However if shared schemas are inhibited, all the related strategies would be inhibited. This problem is solved by the unique, unsharable drive schemas. A drive schema is represented graphically as a hexagon.

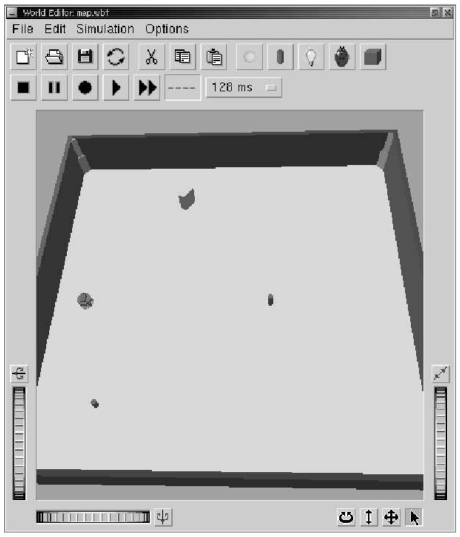
5 Strategies for Foraging and Mapping

Foraging or locating and gathering resources is a fundamental problem that most animals are faced with. When the solution to a foraging problem includes remembering spatial structures and the locations of resources, the problem becomes one of *mapping*. We decided to study these two domains as there should be a natural progression from one to the other and as mapping is a domain where biologists have recognized an abundance of strategies in animals. These strategies 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]. There is also some neurological evidence of how the animal brain does mapping [O'Keefe, 1996]. This knowledge about animal strategies for foraging and mapping indicated that an incremental approach to these problems had a good possibility of being successful. We decided that a reasonable test of an incremental approach would be to show how such a methodology can provide a path that ties foraging to mapping in a demonstrably feasible way.

5.1 Simulated Environment

Our implementations used a Khepera robot with six infrared (IR) proximity sensors and a K6300 color camera, simulated by the Webots robot simulator. We used a square 1.5m by 1.5m simulated environment presented graphically in Figure III. The environments contained an energy source or *feeder*, two fixed obstacles and a Khepera robot. Each corner was marked with a unique color and functioned as a landmark.

Figure III: The Simulated Webots Environment



The complete design for the final solution is presented in Figure IV. The shared schemas are outlined in gray.

5.2 Designing Strategies for Foraging and Mapping

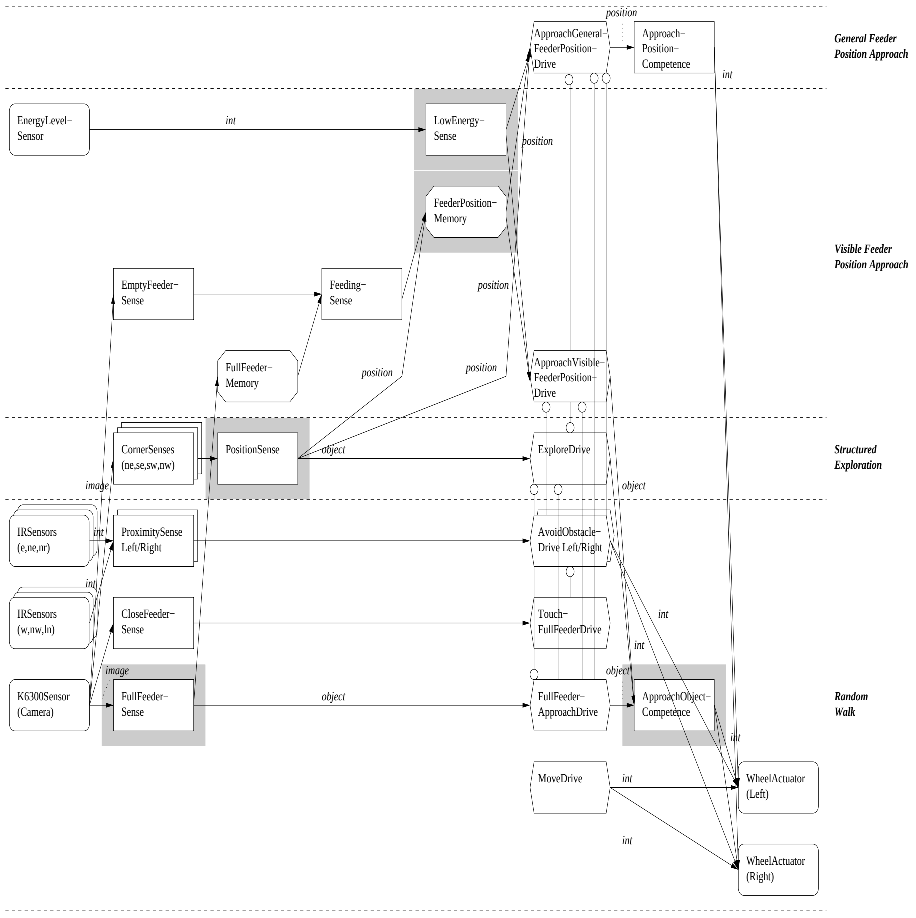
We started with a general mapping strategy in mind, where a robot would explore an environment and be able to revisit previously observed food sources or feeders. For simplicity, we assumed an environment where all landmarks were globally visible. The metric for comparing the viability of the strategies was the *feeding rate*, i.e., the number of times the robot reached a feeder during a set time period. The application of the rules of our methodology as presented in Section 3 is indicated by their labels.

Approach General Feeder Position The general mapping strategy implied remembering the location of feeders and returning to these at strategic times.

Skills - In order to follow such a strategy we recognized that the robot would need to know its current location and the location of the observed feeders. It would also have to know how to return to any given location. Finally, it would need to know when its energy levels dropped to a certain level, in order to head for a feeder at the optimal time.

Support - The problem of returning to a known location was divided into two strategies. the approach *general* feeder position strategy turned the robot

Figure IV: The Behavioral Layers for Foraging and Mapping



toward a known location, while the approach *visible* feeder position strategy approached a known location when it was visible.

In order to orientate the robot toward a known food location, it was necessary to know the robot's current position and pose. This localization skill was delegated to the structured exploring strategy, a strategy that used this information to do efficient exploration. The ability to remember the feeder locations was delegated to the approach visible feeder approach strategy.

The skill left to be implemented by this layer was the ability to turn toward given locations.

Interaction - The approach general feed strategy inhibits the underlying search strategies, i.e., the random walk strategy and the structured exploration strategy. It is inhibited by the underlying strategies for approaching feeders when found, i.e., the feeder approach element of the random walk strategy and the approach visible feeder position strategy. It is also inhibited by the obstacle avoidance element of the random walk strategy. The information requirements identified three key schemas to be provided by the supporting strategies, a low energy sense schema, a position sense schema and a feeder position memory schema. As the top-level strategy, the approach general feeder position strategy did not need to present any schemas to other layers.

Evaluation - The ability to immediately orientate toward known food locations at optimal times rather than waiting for these locations to become visible should provide the robot with a performance advantage by increasing its feeding rate. This advantage does not need any environmental features other than those needed by its supporting strategies in order to be expressed.

Approach Visible Feeder Location The approach visible feeder location strategy brought the robot back to a known feeder location whenever such a feeder location was visible and the robot's energy levels were below a given threshold.

Skills - To follow this strategy the robot needed to be able to remember and recognize feeder locations. This again meant that it needed to know its own location when it encountered them. Lastly, it needed to know how to approach things. A robot's location was defined in terms of the visible landmarks. Hence, the information available in a position percept also indicated which locations were currently visible.

Support - Localization was already delegated to the structured exploration layer. The feeder recognition and approach object competence was delegated to a random walking strategy that would recognize and approach feeders.

The skills to be implemented by this layer was the ability to sense the current energy level and the ability to remember feeder positions.

Interaction - Like the approach general feeder position strategy, this strategy inhibits the underlying structured exploration strategy for searching. It is likewise inhibited by the feeder approach and obstacle avoidance elements of the random walk strategy. In addition to the position sense schema presented by the structured exploration layer, a full feeder sense schema and an object approach competence schema would be presented by the random walk layer.

Evaluation - The ability to return to known feeders when energy got low rather than continuing exploration should give the robot a performance advantage in terms of an increased feeding rate. This advantage does not need any environmental features other than those needed by its predecessors in order to be expressed.

Structured Exploration The structured exploration strategy lead the robot around the given environment according to a pre-defined pattern.

Skills - We found that this strategy needed localization skills and also needed to know how to follow a set pattern of exploration.

Support - Approaching the landmarks that defined the exploration pattern could be handled by the object approach competence schema already delegated to the full feeder approach layer.

The skills left to be implemented by this layer was the localization and the ability to follow a set sequence of landmarks forming a pattern of exploration.

Interaction - Like the other search strategies, this strategy was inhibited by the feeder approach and obstacle avoidance elements of the random walk strategy. No new shared schemas were identified.

Evaluation - A structured search is a more efficient way of exploring an environment that a random walk, and the ability to do this should provide a performance advantage in terms of an increased feeding rate. This strategy needed landmarks to be present in the environment so that a pattern of exploration can be defined.

Random Wandering The random wandering strategy made the robot go forward until an obstacle was encountered and then made it turn around. The sensor and actuator noise made this strategy relatively unpredictable. The strategy was further divided into four basic behavioral layers: move, approach full feeders, touch full feeders, and avoid obstacles, but for brevity we here present these layers as one.

5.3 Implementing the Foraging and Mapping Strategies

We implemented the foraging and mapping strategies presented above starting with the leaf strategies. In order to verify the improvements in performance for each new strategy implemented we performed experiments that compared the performance of each new controllers with its predecessor.

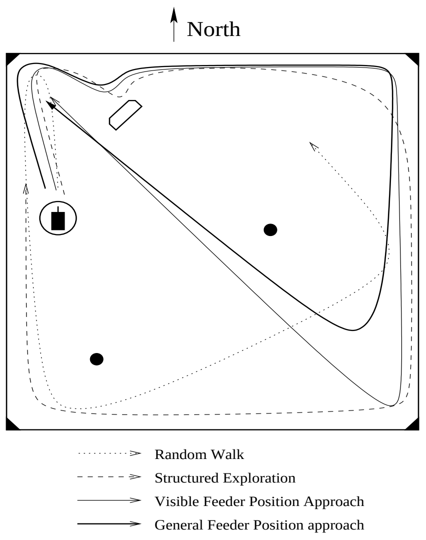
Experimental Evaluation In order to compare the quality of a strategy to its predecessor, we ran 20 trials with each of the controllers and measured the average feeding rate over the first two minutes. At the start of each experiment the robot had an energy level of 1.0. Each 64 ms the energy level decreased by 0.02. When the robot was in contact with a feeder, the energy level rose to 2.0.

If the energy level fell to zero, the robot was considered dead and immediately removed from the environment.

Results The random wandering behavior implemented by the base controller relied on chance to reach different areas of the environment. This strategy could be very inefficient. Due to the nature of this strategy, some similarities existed between the paths taken by the robots following it. Depending on the orientation of a robot when approaching the northwest corner, i.e., the corner where the feeder was situated, it would turn either left or right in order not to hit the walls. Turning left in this situation invariably meant that the robot would not find the feeder before it ran out of energy.

For random wandering, over 20 runs, the mean feeding rate was 1.5/120 seconds with a standard deviation of 0.95/120 seconds. The dotted line in Figure V describes a typical path taken by robots following the random wandering strategy when approaching the northwest corner. The path is typical for the trials where the robots turned left and missed the feeder.

Figure V: Paths Taken by the Foraging and Mapping Controllers


North
Random Walk
Structured Exploration
Visible Feeder Position Approach
General Feeder Position approach

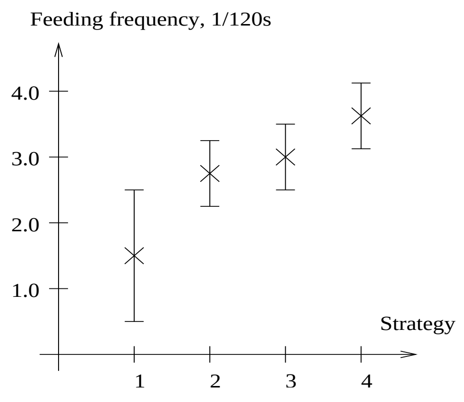
As indicated by the dashed arrow in Figure V, the pre-defined exploration pattern for the structured explore strategy was a square along the boundaries of the environments. This strategy enforced a right turn at every corner and brought the robot back to the feeder at regular intervals. Over 20 trials, the mean feeding rate for robots following the structured exploration strategy was 2.7/120 seconds with a standard deviation of 0.5/120 seconds. 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, the robots were able to head for the position where the feeding took place as soon as it came into view. These schemas 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 3.0/120 seconds with a standard deviation of 0.5/120 seconds. 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 general feeding position approach which allowed the robot to head for the position where the feeding took place any time the low energy sense schema 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 typically produced by the approach visible feeding position strategy. Over 20 trials, the mean feeding rate for robots following the general feeding position approach strategy was 3.6/120 seconds, with a standard deviation of 0.5/120 seconds. This feeding rate is significantly higher than the feeding rate for the structured exploration strategy on a 99% confidence level.

The performance of the different strategies are presented together graphically in Figure VI. It shows the mean number of feeding events per two-minute trial, for each of the strategies, with standard deviations indicated by error bars.

Figure VI: Foraging and Mapping Strategy Performances



6 Strategies for Conflict Resolution

When two entities compete for a limited resource, there are a number of conflict resolution strategies available. We decided to study this domain for three reasons: first, because of its richness in biological evidence [Enquist, 1985, Hauser, 1996]. Second, it is inherently a multi-robot problem as opposed to mapping, and, third, communication as a precursor for language is an important element in high-level adaptive capabilities. Stylized displays are a form of behavior that is explicitly communicative. It would be a good demonstration of our methodology if it could provide a feasible path from reactive interaction, via implicit communication to explicit communication.

6.1 Simulated Environment

The simulated environment in which our experiment on conflict resolution took place was identical to the environment used for the experiment on mapping but with an additional Khepera robot. The energy reduction rate was for this experiment, set to 0.0, implying that none of the robots would die from low energy levels during these experiments.

For this domain we added a *strength* value and an *injury* value to the state of the Khepera robot as used in the mapping domain. The two robots were given different and unchangeable strength values from the start. 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 proportional 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.

The behavioral layers for conflict resolution were added on top of the general feeding position approach controller presented in Section 5. The new layers are presented presented in Figure VII.

6.2 Designing Strategies for Conflict Resolution

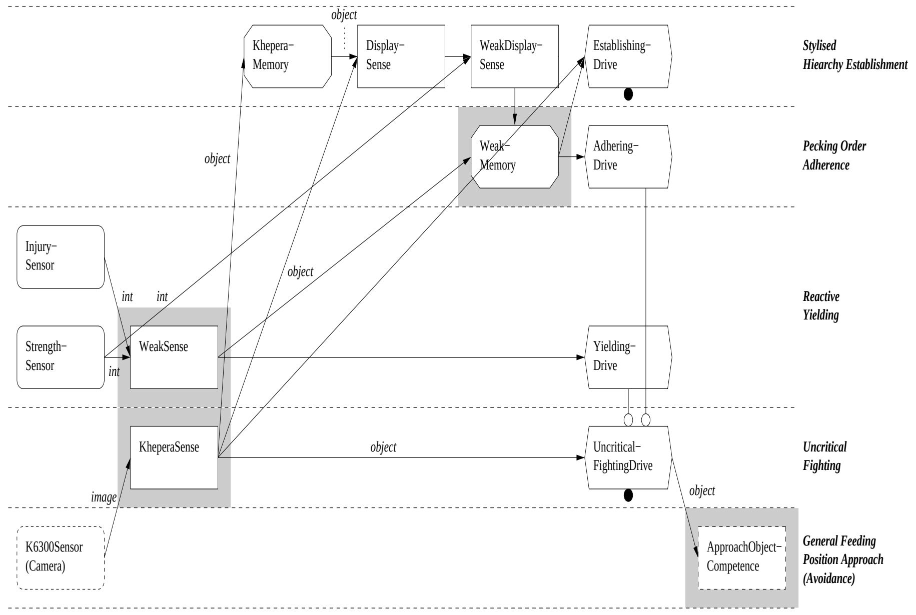
The root strategy we considered was one where fighting was avoided between two robots with different fighting abilities or *strength* by each robot signaling its corresponding social rank. The metric used to evaluate the performance of the conflict resolution strategies was the *survival time*, of the weak robot, i.e., the amount of time it was able to keep its injury levels below the given threshold.

Stylized Hierarchy Formation The stylized hierarchy formation strategy allowed robots to establish a strength-based hierarchy without going through a physical test of strength. The stylized display was to remain motionless. The period the display lasted was proportional to the strength of the robot.

Skills - In order to follow this strategy, the robots needed to be able to perform and recognize a display behavior, to remember their position in a hierarchy, and to attack and avoid the opponent according to this position.

Support - During the display behavior, the stylized hierarchy formation strategy would inhibit all the underlying strategies. We identified a supporting strategy where a hierarchy would be formed through physical interaction, the pecking order adherence strategy. We delegated the ability to remember social position and act accordingly to the layer implementing that strategy. We delegated the

Figure VII: Behavioral Layers for Conflict Resolution



ability to recognize another Khepera to an uncritical fighting strategy. The skill left to be implemented by this layer was the ability to interpret stylized displays.

Interaction - We identified a weak memory schema as a shared schema. This schema would indicate whether the robot was weaker than its opponent. Providing the stylized hierarchy formation layer with the possibility of establishing this memory meant that the hierarchy adherence behavior implemented by the pecking order adherence strategy, could be reused. We also identified a Khepera sense schema as a shared schema to be provided by the uncritical fighting layer.

Evaluation - The elimination of a physical test of strength should increase the survival time for the weak robot by allowing it to start avoiding the strong robot with a lower injury value. No extra environmental features were needed for the robots to follow this strategy.

Pecking Order Adherence This strategy allowed the robots to form and adhere to a strategy based on a physical test of strengths.

Skills - To be able to follow this strategy, the robot needed to be able to perform a physical test of strength and to avoid the opponent if the opponent was identified as stronger.

Support - We identified a support strategy where the robots would engage in physical tests of fights, but where the weak robot would stop fighting or yield only when it sensed directly that it was the weaker. This strategy was called reactive yielding. The fighting behavior and the sense of weakness was delegated to the reactive yielding strategy.

The skill left to be implemented by this layer was the ability to remember the robot's place in the social hierarchy.

Interaction - We chose to make the yielding behavior simply an inhibition of the fighting behavior. As such it could be handled though drive inhibition and no new shared schemas had to be identified. This inhibition took place only after it had been established that the robot was the weaker than its opponent. This solution was possible due to the underlying layers which would treat other robots as obstacles and avoid them by default. A weak sense schema was to be presented by the reactive yielding layer.

Evaluation - The hierarchy established allowed the weak robot to avoid the strong robot in situations where the strong robot is visible to the weak robot without the weak robot being visible to the strong robot. This should improve the performance of the weak robots by reducing the frequency of fights and hence increasing its survival time. No extra environmental features were necessary for the robots to follow this strategy.

Reactive Yielding This strategy allowed the weak robot to abort a fight and escape further injury.

Skills - To follow this strategy the robots needed to be able to attack and escape from another robot. They also needed to know whether they were the weaker or stronger robot.

Support - We identified the uncritical fighting as the supporting strategy for reactive yielding and delegated the fighting behavior to that layer.

The skill left to be implemented by this layer was the ability to sense whether the robot was stronger or weaker than its opponent.

Interaction - Again, the yielding behavior was an inhibition of the fighting behavior. For this strategy the inhibition took place only when the robot could sense immediately that it was weaker than its opponent.

Evaluation - By aborting fights, the weak robot can reduce the average duration of fights. This leads to an increased survival time. No extra environmental features were needed for the robots to follow this strategy.

Uncritical Fighting This strategy lead the robots to attack each other on sight and to keep on fighting until one of the robots reached the set injury threshold, i.e., *died*. It was implemented directly on top of the approach general feeder position controller presented in Section 5 and used the approach object competence schema as a shared schema. Since we interpreted close proximity as physical interaction, the approach object competence was sufficient as a fighting behavior.

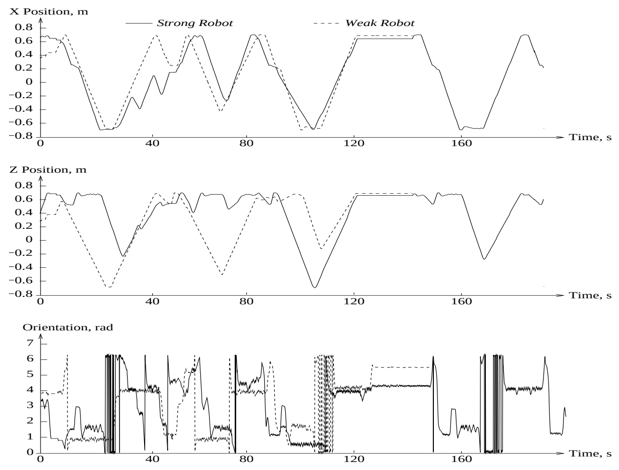
6.3 Implementing the Conflict Resolution Strategies

On top of the existing layers for foraging and mapping we implemented the final solution for conflict resolution as presented in Figure VII. Starting from the bottom, each layer was fully implemented and tested.

Experimental Evaluation For each strategy evaluation experiment we ran 20 trials and calculated the average survival time and its standard deviation for the weak robot. The survival time for the strong robot was constant as it never died.

Results The average survival time for the weak robot following the uncritical fighting strategy was 63.4 seconds, with a standard deviation of 25.0 seconds. The position and orientation during a typical trial for two robots following the stylized hierarchy establishment strategy, is plotted in Figure VIII with the strong robot represented by a solid line and the weak robot represented by a dashed line. The x and z axis are in units of 1 meter and have Origo in the center of the simulated environment. The positions are given in radians with 0 indicating north as illustrated in Figure V. The plot shows the typical tendency of the strong robot to find and follow the weak until it has cornered it.

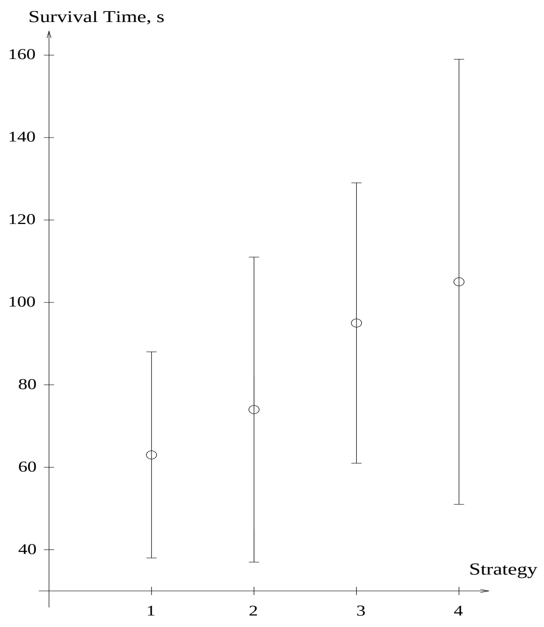
Figure VIII: Typical Robot Poses During Conflict Resolution



The decreasing distance between the solid and the dashed lines from the start of the trial to 120s into the trial, indicates that the strong robot is gaining on the weak robot. In the period from 120 seconds to 140 seconds there is not change in position, indicating that the strong robot has cornered the weak robot. During this period the strong robot inflicts more injury on the weak robot than vice versa due to their relative strengths. The discontinuation of the dotted line after 140 seconds indicates that the weak robot's injury levels reached the set threshold and that it was removed from the environment. 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. This average survival time was significantly longer than the average survival time for robots that followed the uncritical fighting strategy, on a 75% confidence level. 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. This survival time was significantly longer than the average survival time for robots that followed the reactive yielding strategy, on a 95% confidence level. The average survival time for the weak robot when following the stylized hierarchy establishment strategy was 104.6 seconds with a standard deviation of 54.7 seconds. This average survival time was significantly longer than the average survival time for robots that followed the pecking order adherence strategy, on a 60% confidence level. The average survival time with standard deviation for each strategy are plotted together in Figure IX.

Figure IX: Conflict Resolution Strategy Performances



7 Conclusions

Section 5 and 6 presented example applications of our methodology to the problems of foraging/mapping and conflict resolution. The examples demonstrate that it is possible to decompose complex strategies into trees of self-contained testable supporting strategies in a way that facilitates incremental development. The examples also demonstrate that it is possible to design partial solutions with relatively small incremental steps between them. This is a good thing as it allows frequent testing when this is desirable. Straddling several increments in one step is always possible if a reduced testing frequency is desirable. The performance evaluation experiments show how the methodology provides a simple but general learning mechanisms. The adaptive solutions can make immediate use of salient environmental features to change their behavior and increase performance relative to their less adaptive predecessors.

8 Related Work

The work presented in this article makes contributions in three main areas: incremental development, learning, and behavior-representation. Below we discuss in detail how our work relates to other work in those areas and what we contribute.

Incremental Development Bryson, in her PhD thesis [Bryson, 2001], describes a iterative development methodology called Behavior Oriented Design (BOD). Our methodology is closely related to BOD in that both methodologies decompose high-level behaviors into more basic activities. Both methodologies are also instantiations of the traditional design-implement circle for incremental development presented in Figure I. The main difference between our methodologies is that ours, being inspired by evolution, requires a tree of *self-contained* solutions. We require each partial solution to be a self-contained solution to a problem related to the root strategy. This results in partial solutions who's performance can be tested comparatively. This indicates that it might be possible to use our solutions to guide automated development using e.g., ER or DR. While BOD is a more direct approach, our approach also provides solutions with the additional qualities of high performance during learning and graceful degradation on partial failure.

ER [Nolfi and Floreano, 2000] is a general methodology for developing biologically inspired adaptive behaviors through a process of artificial evolution. Work on ER has traditionally been limited to demonstrating the evolution of a specific capability such as vision rather then a series of increasingly sophisticated solutions integrating multiple sensory modalities. Another active branch of ER research is the evolution of adaptive controllers in the form of ANNs, that are well suited for epigenetic development. It is not possible, using currently existing ER techniques, to produce controllers as specialized as the ones presented in this paper.

DR [Weng et al., 2000] is another automated development approach. Robots developed using this approach have been successful in learning complex tasks, e.g., vision-based navigation. DR is based on Piaget's theory of epigenetic development and has produced several architectures that reproduce the staged capabilities described by this theory. This work has produced complex robot behavior through an incremental process starting by developing associations for each sensing modality, progressing by developing inter-modal associations and finally developing associations between sensors, context and behavior. Other work on DR has learned among other things causality in billiard-ball bouncing [Cohen et al., 2002]. However, work on DR does not provide general guidelines on how to structure goal-oriented incremental development.

Efficient Learning In robotics the real time constraints make problem restriction crucial. Learning in BB systems has previously been restricted to a set of separate, 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]. Though these solutions use restricted problem spaces, they are based on learning algorithms with long convergence times, typically Reinforcement Learning (RL). Bryson [Bryson, 2001] proposes problem restriction principles aligned with our methodology, where the context of a behavior restricts the learning problem considered. Similar to Bryson's work, our approach implements the minimal amount of adaptivity needed to supporting a given strategy. As a result our solutions are far less adaptive but far more efficient than solutions based on general ML mechanisms such as RL.

Behavior Decomposition BB architectures have previously divided layers into *modules* [Brooks, 1986] or schemas [Arkin, 1989], though there are also many examples of BB architectures that do not subdivide layers [Werger, 2000, Nicolescu and Matarić, 2001]. The ALLIANCE architecture [Parker, 1998] explicitly classifies the elements of behavioral layers into regular and motivational behaviors. The term behaviors in the ALLIANCE architecture denotes an element of a behavioral layer. A motivational element monitors multiple sources of input and adjust the activation strength of a set of related, regular, elements according to its state. Similar motivation specific elements have also been used in the BB architecture supporting the emotion-based learning model on Kismet, a robotics head that can make facial expressions [Breazeal and Scassellati, 1998]. The architecture used in Kismet also classifies some of its behavioral structures as perceptory or motory. Bryson and Stein's Basic Reactive Plans (BRPs) [Bryson and Stein, 2001] also divide the *how* and the *when* of behaviors into behaviors and plans respectively. BRPs however have an action selection mechanism based on explicit behavior priorities rather than mutual excitation and inhibition. Our modularized behavior model is similar to the models used in ALLIANCE and Kismet in that it separates motivation/activation and regular elements of behavior and uses excitation and inhibition for action selection.

9 Discussion

Currently robotic systems are not so complex that the risk reduction provided by incremental development approaches can be expected to have a major impact on the field. However, in BB robotics there is currently a lot of reimplementation of basic skills. The work presented here contributes insight into behavior representation that will help to standardized representations of skill modules for incremental development. This standardization will have a great impact on the efficiency with which new algorithms can be implemented.

Using multiple concurrent strategies of varying sophistication is an intuitive way of ensuring performance during learning and graceful degradation. The examples presented here demonstrates this approach using simple learning mechanisms but it can also be used with more general ML techniques such as RL. It is a natural way to provide performance while waiting for the convergence of such methods and thus help to make these methods more generally applicable.

Finally, in order to develop increasingly complex BB systems it is necessary to understand more about the issues concerning behavior interaction, in particular how general skill modules are shared by multiple behaviors and how they are refined by natural incremental processes such as evolution and epigenetic development. Using hand coded solutions, it is possible to study aspects of these problems that are difficult to study using automated methods, e.g., issues of skill abstraction for shared skill modules and deictic variables as presented in this paper. The insight gained can feed back and help to improve the automated methods.

10 Future Work

One of the challenges ahead for this work is to apply it to more complex and demanding problems to see whether the advantages demonstrated here can be transfered to other domains. Another interesting result of a wide application of our methodology would be the identification of a commonly useful skill sets. A limited set of generally useful perceptual and behavioral skills will form a limited search space for ML techniques and will make it easier to rapidly learn new behavioral strategies.

Another path for this work is to increase the adaptive capabilities to include more complex forms of learning than imprinting, e.g., conditioning. We are particularly interested in introducing associative memories in order for a robot to learn new behavioral strategies from the available skill modules. Associative learning would tie in with other areas of research, in particular the area of RL.

11 Summary

We present a methodology for incremental development of complex adaptive behaviors. The work was motivated by two factors: first, the possibility of reducing the risk of spending resources on infeasible design solutions. Second, and more importantly, the possibility of increasing the performance levels and fault tolerance levels of BB systems by applying multiple concurrent solutions of different sophistication to a given problem.

Like incremental development approaches in general, our recommended process forms a loop of design and implementation. In addition to this general process we presented recommendations on how to construct a tree of self-contained solutions where the less sophisticated solutions provide support to the more sophisticated solutions in terms of abstract modules for perceptory and behavioral skills. We also present a modularization of behavioral layers that supports, at the same time, abstract skill sharing and action selection through excitation and inhibition.

Two demonstrations of the application of the methodology are presented. Complex behaviors are developed incrementally and performance tested through experiments in simulation. The demonstrations show that it is possible to construct strategy trees for incremental development of complex adaptive behavior.

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.

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 free student license for this work.

References

[Andre and Russel, 2001] Andre, D. and Russel, S. J. (2001). Programmable Reinforcement Learning Agents. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Proceedings of the 13th Conference on Neural Information Processing Systems (NIPS'01)*, pages 1019–1025. MIT Press, Vancouver, Canada.

[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.

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained: Embrace Change.* Addison-Wesley, Reading, Massachusetts.

[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*, 2:14–23.

[Bryson, 2001] Bryson, J. J. (2001). *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents.* PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Massachusetts.

[Bryson and Stein, 2001] Bryson, J. J. and Stein, L. A. (2001). Modularity and design in reactive intelligence. In Nebel, B., editor, *Proceedings of the 17th Intenational Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 1115–1120, Seattle, Washington.

[Cockburn, 2001] Cockburn, A. (2001). *Agile Software Development.* Addison-Wesley, Boston, Massachusetts.

[Cohen et al., 2002] Cohen, L. B., Chaput, H. H., and Cashon, C. H. (2002). A constructivist model of infant cognition. *Cognitive Development*, 17:1323–1343.

[Enquist, 1985] Enquist, M. (1985). Communication during aggressive interactions with particular reference to variation in choice of behavior. *Animal Behavior*, 33:1152–1161.

[Gould and Gould, 1999] Gould, J. L. and Gould, C. G. (1999). *The animal mind.* Scientific American Library, New York.

[Hauser, 1996] Hauser, M. D. (1996). *The Evolution of Communication*. MIT Press, Cambridge, Massachusetts.

[Larman and Basili, 2003] Larman, C. and Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6):47–56.

[Lyons and Arbib, 1989] Lyons, D. M. and Arbib, M. A. (1989). A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280–293.

[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)*, pages 796–802, Boston, Massachusetts.

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

[Matarić, 1997] Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83.

[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, San Diego, California.

[Nicolescu and Matarić, 2001] Nicolescu, M. and Matarić, M. J. (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.

[Nolfi and Floreano, 2000] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, Massachusetts.

[O'Keefe, 1996] O'Keefe, J. (1996). Geometric determinants of the place fields of hippocampal neurons. *Nature*, 381:425–428.

[Parker, 1998] Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. *IEEE Transactions on Robotics and Automation*, 14(2):220–240.

[Pressman, 2001] Pressman, R. S. (2001). *Software engineering: a practitionars approach*. McGraw-Hill, Boston, Massachusetts, 5th edition.

[Raymond, 1999] Raymond, E. S. (1999). The cathedral and the bazar. In Raymond, E. S., editor, *The Cathedral and the Bazar: Musings on Linux and Open Source by an Accidental Revolutionary*, pages 19–64. O'Reilly and Associates, Sebastopol, California.

[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.

[Weng et al., 2000] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., and Thelen, E. (2000). Autonomous mental development by robots and animals. *Science*, 291(5504):599–600.

[Werger, 2000] Werger, B. B. (2000). Ayllu: Distributed port-arbitrated behavior-based control. In Parker, L. E., Bekey, G., and Barhen, J., editors, *Distributed Autonomous Robotic Systems 4, Proceedings of the 5th International Symposium on Distributed, Autonomous Robotic Systems (DARS'00)*, pages 25–34, Knoxville, Tennessee.