TECHNICAL REPORT
NATICK/TR-04/003

AD_____

SHARED SEMANTIC REPRESENTATIONS FOR COORDINATING DISTRIBUTED ROBOT TEAMS

by
Ian Horswill

Northwestern University
Evanston, IL 60201

December 2003

Final Report
June 1998 – June 2002

Prepared for
U.S. Army Research, Development and Engineering Command
Natick Soldier Center
Natick, Massachusetts 01760-5020

DISCLAIMERS

The findings contained in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of trade names in this report does not constitute an official endorsement or approval of the use of such items.

DESTRUCTION NOTICE

For Classified Documents:

Follow the procedures in DoD 5200.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX.

For Unclassified/Limited Distribution Documents:

Destroy by any method that prevents disclosure of contents or reconstruction of the document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
05-12-2003	Final Report	June 1998 - June 2002

4. TITLE AND SUBTITLE
SHARED SEMANTIC REPRESENTATIONS FOR COORDINATING DISTRIBUTED ROBOT TEAMS

5a. CONTRACT NUMBER
C-DAAN02-98-C-4023

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Ian Horswill

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northwestern University
Computer Science Department
1890 Maple Avenue
Evanston, IL 60201

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Sponsor: Defense Advanced Research Projects Agency (DARPA)
Microsystems Technology Office (Elana Ethridge)
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NATICK/TR-04/003

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
distribution is unlimited.

13. SUPPLEMENTARY NOTES
Monitor: US Army Research, Development & Engineering Command, Natick Soldier Center,
ATTN: AMSRD-NSC-SS-MA (T. Gilroy), Kansas Street, Natick, MA 01760-5020

14. ABSTRACT
Most physically implemented multi-robot controllers are based on extensions of behavior-based systems. While efficient, such techniques suffer from a paucity of representational power. Symbolic systems, on the other hand, have more sophisticated representations but are computationally complex and have model coherency issues. In this report, we describe HIVEMind (Highly Interconnected Verbose Mind), a tagged behavior-based architecture for small teams of cooperative robots. In HIVEMind, robots share inferences and sensory data by treating other team members as virtual sensors connected by wireless links. A novel representation based on bit-vectors allows team members to share intentional, attentional, and sensory information using relatively low-bandwidth connections. We describe an application of the architecture to the problem of systematic spatial search.

15. SUBJECT TERMS

SEMANTICS	COMPUTERIZED SIMULATION	ROBOTS	ARTIFICIAL INTELLIGENCE
ROBOTICS	COMPUTER ARCHITECTURE	COOPERATION	SENSES(PHYSIOLOGY)
BANDWIDTH	COMPUTER COMMUNICATIONS	NATURAL LANGUAGE	

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Thomas Gilroy
U	U	U	SAR	39	19b. TELEPHONE NUMBER *(Include area code)* 508-233-5855

Table of Contents

List of Figures

List of Tables

Preface

This report documents the research conducted by the Northwestern University Computer Science Department, Evanston, IL under contract C-DAAN02-98-C-4023 sponsored by the Defense Advanced Research Projects Agency (DARPA), Arlington, VA and monitored by the U.S. Army Soldier Systems Center, Natick, MA. The research was conducted between June 1998 and June 2002.

The work under this contract provided support for structured representations and simple cognitive processing in multi-robot systems, leading to greatly improved component reusability. This can allow humans to direct robots with useful high-level instructions and allow solution of dramatically more complex tasks. Single-agent architectures, known collectively as role-passing architectures, can be extended to multi-robot systems by linking team members with a wireless network to exchange sensor and inference data. This can provide team members with a shared situation assessment and limited on-demand access to the sensory data of other team members. The architecture can enable improvements in situational awareness, team coordination, and human-robot coordination, as well as software component reusability, reduced system complexity and performance of dramatically more sophisticated tasks.

SHARED SEMANTIC REPRESENTATIONS FOR COORDINATING DISTRIBUTED ROBOT TEAMS

Summary

Northwestern University's Computer Science Dept. has developed Highly Interconnected Verbose Mind (HIVEMind), a multi-robot control architecture that supports efficient sharing of symbolic information between team members. Based on role-passing [1], HIVEMind can efficiently implement quantified inference over unary predicates and provide hard real-time guarantees for synchronization of knowledge bases between team members and the world. Objects in a team member's working memory are bound to a small, fixed set of roles such as *agent, patient, source, destination*, etc. When a role is bound to an object, a visual tracking system is dynamically allocated to it and tagged with the name of the role. Since the number of roles is relatively small, we can represent sets of roles, and thereby sets of objects, as bit-vectors, with one bit representing each role.

This representation allows inference to be compiled directly to straight-line machine code consisting only of load, store, and bit-mask instructions. While more limited than a full logic-programming system, it does allow us to express most of the kinds of inference used on physical robots today. The inference rules are completely rerun on every cycle of the system's control loop, allowing the robots to respond to contingencies as soon as they are sensed. The compiled code is sufficiently efficient that inference is effectively free – 1000 Horn clauses of 5 conjuncts each can be completely updated at 100Hz using less than 1% of a current CPU.

The bit-vector representation also allows for very compact representation of the robot's knowledge base. Unary predicates are stored in one machine word. Function values are represented using small arrays indexed by role. For the kinds of tasks currently implemented by multi-robot teams, this representation is sufficiently compact to allow the *entire knowledge base* to fit into a single User Datagram Protocol (UDP) packet. Robots can therefore share information by periodically broadcasting the knowledge base. This simple communication and coordination model provides each robot with transparent access to every other robot's state, a kind of "group mind." It allows the team to efficiently maintain a shared situational awareness and to provide hard real-time response guarantees; when a team member detects a contingency, other members respond in O(1) time without the need for complicated negotiation protocols.

We have tested the architecture with a three-member team consisting of two robots and an operator control unit running the same architecture (without sensors or effectors) to monitor and control the team. Sensing, inference and control decisions are performed at 10Hz, allowing the robots to move at 1 m/s. The team can be dynamically tasked on simple natural language to perform systematic spatial search tasks ranging from informing everyone in a building of an announcement to finding and containing an intruder.

1. Introduction

Traditional robot control architectures are firmly split between those architectures that support sophisticated high-level inference and those architectures that have been run successfully on real teams of robots. This report describes a novel, implemented, multi-robot control architecture that cleanly joins symbolic reasoning with sensory-motor control.

1.1. Problems with previous reactive and behavior-based architectures

Reactive control architectures allow the designer to specify a functional mapping from sensor readings to effector commands. Typically, this mapping is specified in terms of a set of stimulus/response pairs together with some arbitration mechanism (see [2] for a survey). The stimulus is some predicate over possible sensor values, and the response is some effector command. The need for arbitration arises because the environment may generate multiple stimuli at one time, causing the control system to feed contradictory commands to the effectors. The arbitration mechanism reads all proposed effector commands and uses some sort of conflict resolution strategy, such as weighted averaging (motor Schemas), weighted voting (the Carnegie Mellon University (CMU) Distributed Architecture for Mobile Navigation (DAMN) architecture), or absolute prioritization (subsumption). Behavior-based architectures are essentially the same, except they allow the modules implementing stimulus/response pairs (aka "behaviors") and/or the arbitration mechanism to include internal state.

Reactive and behavior-based architectures limit the designer to very simple representations. Communication between modules is typically limited to simple signals in which a simulated wire carries a binary or numeric value whose value is recomputed with each cycle of the control system. This has the twin advantages of being efficient and of interfacing well with sensors and effectors. However, it makes it nearly impossible to express high-level abstractions such as predicate/argument structure. In other words you can have a *distance* wire that carries the distance to the nearest object, but not a *distance(x)* predicate that tells the distance of x for any given x.

The worst effect of these limitations is a loss of modularity. Rather than implementing one *get(x)* behavior, a behavior-based system might implement separate *get-red-puck*, *get-green-puck*, and *get-blue-puck* behaviors. Behavior-based solutions have been proposed for problems requiring predicate/argument structure (Maes' blocks world solution [3], Nippon Telephone and Telegraph (NTT) reactive dialog system [4]); however, they all involve using duplicate behaviors for each possible predicate/argument combination, and so have only been implemented in simulation. Maes' system, for example, requires a control system whose size is cubic in the number of objects in the world.

The loss of modularity increases system complexity, which in turn increases cost, size, power consumption, and debugging time. It also hinders code reuse since modules must be highly specialized.

The loss of the ability to express abstractions and predicate/argument structure also hinders the *instructability* of the system. Because all representations are expressed in terms of low-level sensor values rather than high-level semantic roles or predicates with arguments, it is effectively impossible to build components like parsers or natural language generators. As a result, human interfaces are typically restricted to systems that directly dump sensor data and behavior activation levels. True high-level instructability and monitoring require the commander to be able to issue orders and receive reports in terms of meaningful high-level structures such as explicit situation descriptions (attacking a specified target, fallback to a specified position, search for a specified object, evasion, control of a specified perimeter, etc.).

1.2. Problems with symbolic control architectures

The attraction of symbolic control architectures is that they promise to allow just these kinds of high-level abstractions to be represented and communicated to the user. However, the success of reactive control architectures is due in large part to the difficulty of implementing symbolic control architectures on real robots.

Symbolic architectures typically assume an articulated world model consisting of every true fact that might be relevant to the decisions the system must make. Making this work on real robots introduces serious engineering problems:

- The perceptual system must build the model somehow
- The representations used by the perceptual system and the problem solver are typically very different; mediating between them can be an expensive operation.
- Since the perception system can't build a model of every true fact about the environment, it must be given some kind of guidance about what information is relevant to the problem solver's current goals
- Since the world is changing continually, the perceptual system must have some way of informing the problem solver of changes in the environment
- The problem solver must then keep a record of which inferences depend on which elements of the sensory data, so the inferences can be updated when the sensory data changes.

These problems are serious. No general, workable solution to them has been proposed. In practice, the linkage between the problem solver and the peripheral systems is dealt with in an ad-hoc manner. The problem solver provides primitives for updating parts of the world model by forcibly triggering different perceptual operations. It is left largely to the programmer to make sure that the right things are checked at the right times. If they are not, the robot can do stupid things, sometimes with disastrous consequences.

Another problem with symbolic systems is their cost. Full-blown planners and problem solvers typically require a full lisp environment running on a workstation-class machine. Placing a workstation-class machine inside of a mobile robot and running it off of batteries is extremely cumbersome given today's technology. For example, the Real-World Interface (RWI) B14 system ($15K) is shipped with a Pentium-class machine

running Unix. While convenient for programming, the system is not terribly well adapted to signal processing problems like real-time vision. Moreover it has a battery life of only an hour or two (assuming batteries in optimal condition). As a result, labs typically use machines like the RWI B21, which has considerably more compute power and over a kilowatt-hour of battery power (allowing it to run for 4-6 hours). It is, however, considerably more expensive ($40-60K). It is also somewhat larger than human-size, making it non-trivial for it to get through a standard 36" doorway without running into the dead-zones of its sonars.

The final issue with symbolic systems is speed. Reactive systems are fast in part because they restrict themselves to computations that are easily parallelized and that run in constant time. Most problem solvers rely on exponential-time search techniques. Inference is also hard to parallelize. For example, the unification algorithm used by nearly all problem solvers to match rule patterns to database entries is believed to be unparallelizable because it is P-complete and so its efficient parallizability would imply the parallelizability of arbitrary polynomial-time algorithms, see [5].

1.3. Tiered Architectures

The traditional solution to the problems of these architectures is to join a traditional reactive system and one or more traditional symbolic systems together. The result is called a "tiered" or "hybrid" architecture (see [2] for a survey). Tiered architectures combine many of the advantages of the two classes of architectures. They provide real-time response for sensory-motor primitives yet allow more advanced representations to be used.

However, tier architectures also combine many of the *disadvantages* of both architectures. Since there is still no theory of how the world model is to be updated, it is still left to the individual programmer to ensure that the symbolic system runs the right operators (or posts the right daemons to wait to the right events) at the right times so as to ensure that the world model is up to date. It is also up to the programmer to add explicit failure and cleanup clauses to the symbolic system to force a non-local return if an event occurs that invalidates a decision made by a supergoal of the current goal.

The problem is exacerbated when multiple symbolic systems are used. The most common tiered architecture consists of a behavior-based system that implements sensory-motor primitives, a planner that chooses sequences of primitives, and an executive (on-line problem solver) that issues the primitives and attempts to perform error detection and recovery. Performing error detection and recovery (EDR) is difficult unless the executive is given explicit knowledge of the domain, the system's overall goal, and the logic of how the plan is supposed to achieve the goal. Without this knowledge, the executive cannot know what results in the world would actually constitute a plan failure, much less how to recover from it. However, with the knowledge, the executive would be nearly able to solve the task itself, leaving little work for the planner. To make matters worse, the

planner and executive typically each have their own symbolic world models, which must not only be kept in sync with the world, but with each other.

In practice, the problems are severe enough that the planner is either never used, or is used on hand-entered data that is known in advance not to be changeable (such as a floor-plan). This alleviates the problem of generating the planner's model from sensor data. The "three-tiered" (3T) architecture [6], which is in wide use for National Aeronautics and Space Administration (NASA) crewed space projects, uses its planner in only about 5% of tasks; their biggest engineering problem is keeping the models up to date.

1.4. Role-passing architectures

We have developed a class of architectures, called *role-passing architectures* that cleanly implement a limited but highly useful subset of symbolic inference using parallel feed-forward networks whose inputs are easily generated by sensory systems. The result is a system with the look-and-feel of a traditional symbolic architecture and the performance characteristics of a behavior-based architecture: *the system continually recomputes all inferences at sensor rates, responding immediately to contingencies by switching plans and subgoals as the need arises.*

A role-passing system represents the agent's task and environment in terms of a set of predicates and a set of bindings of external objects to *roles*. Predicates, as in first-order logic, describe the properties of objects. However, in role passing, the domain of a predicate (the set of things you can apply it to) is limited to a finite set of indexical *roles*. The bindings of the roles to actual objects in the world are implemented by pools of sensory systems (such as visual trackers) or memory systems (such as images of objects). When the problem-solver directs a sensory system is to track a given object, it also provides a role to bind the object to. A single object can be represented in many different pools, provided that it is bound to the same role in each pool. In effect, roles are simply an attention mechanism. Real agents have limited perceptual bandwidth and can only attend to a limited set of objects at one time. Role passing simply makes this restriction visible within the problem solver.

Communication between subsystems is performed mostly in terms of roles. The sensory and memory systems report to the problem solver what roles they have bound. They also report what roles satisfy various predicates, such as *near(x)*, *facing(x)*, etc. When the problem solver wants the motor control system to follow a given object, it provides the motor system with the role name of the object to be followed. The motor system then forwards the role to sensory systems that might be able to provide it with the appropriate information about the object. Whichever sensory system is bound to that role will then route the appropriate signals to the motor system in real time.

The architecture makes it possible to compile inference rules written in a subset of modal logic with equality into fast feed-forward networks that can be evaluated and updated in parallel. Such networks are easily and compactly implemented on a variety of

architectures, including conventional microcontrollers, distributed networks of microcontrollers, true parallel machines, and even field programmable gate arrays (FPGAs). Current microcontroller technologies, such as the Intel X-Scale architecture could conservatively easily run rule base of 1000 inference rules at 1000 Hz (1000 complete revisions of the knowledge base per second) on a power budget on the order of 100mW. FPGAs could reach the megahertz level, if there were an application which warranted it.

Role passing allows the designer to program using declarative representations that specify the behavior of the domain directly, rather than specifying canned plans for achieving particular tasks. This allows the agent to dynamically infer the correct next action as sensory data changes, seamlessly switching between error recovery actions and normal activities as the world changes. Although not as powerful as a full planning system, it provides dramatically more power than a behavior-based system and dramatically better performance, both in speed and robustness, than either a symbolic reasoner or a hybrid system.

1.5. Shared high level representations for effective robot teamwork

Many advanced multi-agent planning systems are moving toward the use of *explicit representations of teamwork* (e.g. STEAM [7]). In these systems, each team member has a representation of a set of shared beliefs, intentions, goals, and plans. While such systems are highly effective in simulation (e.g. the IJCAI-97 RoboCup simulated soccer tournament), they have not yet been demonstrated on real robot hardware. One assumes that this is due in large part to the difficulty of porting a large symbolic AI system (STEAM runs under the SOAR production system architecture) and interfacing its database to real sensors.

Role passing offers the possibility of implementing such high-level coordination simply and cheaply. Since the architecture already consists of a parallel network of signals (sensor readings, inferences, etc.), it is a straightforward matter to distribute the network across robot bodies. We simply choose a set of signals within the network to make available to other members of the team. Each robot periodically collects together the values of its signals and transmits them as a broadcast packet on a wireless network. As each robot receives broadcast packets, it stores them in memory and uses them as the source of selected signals of its own network. In effect, the wireless network implements a large set of wires tying the internal networks of the different robots together. This kind of network is very robust with respect to network failure, since packet loss simply causes a temporary failure to update other robots, rather than blocking, deadlock, or complete system failure. Robots can also keep time stamps for the packets they've received so that they can tell when information is stale. This will allow team members cut off from the network to explicitly represent what team members they are cut off from and how accurate their current data is. They can then explicitly reason about what data to trust and what actions to take to regain contact with the group (if any).

Distributing the network across team members provides explicit representations of teamwork for free. The group can collaboratively maintain a shared situational assessment that tracks

- Shared goals and objectives
- Classification of global situation (advancing, fallback, search, evade, ambush, ...)
- Threats
- Team member status (on-line, dead, damaged)
- Team member role within current operations (team leader, point, rear guard, advance patrol, ...)
- Shared plans for attaining objectives

For example, if a team member encounters an enemy team member, it can activate the *ambush* situation for the entire team and bind the *location* role to its own Global Positioning System (GPS) coordinates. Other team members will then have immediate access to that information and can begin to act accordingly.

Again, these features can be implemented in hard real-time using small, low cost, low power components.

1.6. High level human interface

The shared plans, goals, and situational assessment used to coordinate the team internally are just exactly the type of data that a human commander would want to make command decisions. The architecture thus provides all the hooks necessary to implement a high-level multi-media command station that will allow human commanders to quickly and efficiently determine the status of their team, gather further data, and communicate a decision.

We have developed a laptop command station that provides dynamic status information. The command station gathers information by listening to the broadcast stream from the team members and displaying the appropriate information in the appropriate modality. The station is implemented as a role-passing control system. The same language used to program the robots is used to program the command station.

Control of the team is achieved by injecting traffic into the network. To instruct the team to search for a green ball, the command station need only assert the *find* command and bind the *target* role to the red-green-blue (RGB) coordinates for the green ball. A human commander can signal this simply by typing:

```
Find green.
```

The command station parses the command, broadcasts the find command, and the role bindings used by it.

1.7. Related work

Most implemented multi-robot systems are limited to simple reactive architectures [8,9,10]. These architectures interface cleanly to sensors and effectors, support limited incremental development, and can support real-time performance on low cost compute hardware [11]. However, they limit the designer to simplistic representations such as numeric signals. These limitations make design reuse difficult by permitting only very restricted parameter passing. Moreover, they severely restrict the space of tasks that can be performed *at all*, since tasks that require inference, natural language understanding, etc. do not map cleanly into these representations.

Behavior networks are a common example [3]. While convenient for problems such as arbitrating between feeding and predator avoidance, behavior networks become combinatorially explosive when implementing behaviors such as picking up or stacking objects that are logically parameterized by an object in the world. For these tasks, separate behaviors must be implemented for each possible object. Even simple problems such as block stacking require impractically large networks, and so have never been implemented on real hardware [3]. Similar problems occur when applying these technologies to human/computer or human/robot communication. In a human/computer dialog system, for example, Hasegawa et al. [4] were forced to use a separate behavior for each possible utterance of each possible speaker.

Rosenschein and Kaelbling [12] implemented propositional inference in a subset of modal logic by compiling it to fast logic networks. The system is very similar to our system, but does not support any kind of variable binding or predicate/argument structure. Agre and Chapman [13] argued that the need for variable binding could be reduced by the use of *deictic representation*, a technique in which the pre-existing attention mechanisms of the perception system are used to implement a kind of variable binding. By pushing variable binding out to the peripheral systems, they were also able to implement reasoning with a simple combinational logic network. Role passing can be seen as an addition of a small amount of variable binding to deictic representation. Nilsson [14] directly addressed the variable-binding problem in his work on teleo-reactive trees. He built a system that incrementally grew a Rosenschein-and-Kaelbling-style inference network as it executed a conventional Lisp program. Whenever the program encountered a procedure call with novel arguments, it would add circuitry to the network to implement it. It can be seen as similar to the SOAR architecture, where chunking is replaced by the growing of the network. The work did not attempt to address the issues of grounding a growing network in the perception system. Shastri and Ajjanagadde [15] describe an inference system called SHRUTI that implements a reasonably full version of inference with frames and data structures. SHRUTI is based on current neurological theories of binding in the neocortex. Although the motivations (and limitations) of SHRUTI and role passing are somewhat different, they are substantially similar in spirit: both implement a limited form of variable binding in which the total number of outstanding bindings is limited. Minsky [16] also describes a proposal for *c-lines*, a frame implementation mechanism similar in spirit to role passing, although the details were never worked out. None of these proposals, except Rosenschein and Kaelbling's system, have been implemented on real robot hardware.

Traditional symbolic architectures [17] and hybrid planner/reactor architectures [6] support the sophisticated structured representations required for high level cognitive tasks but require all sensory data used by the symbolic system to be fused into a central symbolic representation. To date, no general-purpose approaches to this problem have been proposed. In practice, symbolic architectures that are deployed on real robots use carefully hand-coded domain-specific rules to mediate between sensors and the symbolic model. This is a cumbersome process that must often be repeated for every new task. The models so built fall far short of building the kinds of elaborate, detailed models required by real AI planners. As a result, even when full-blown Stanford Research International Planning System (STRIPS) planners are implemented on real robots, they are rarely, if ever, executed. When they are used, they are typically used with world models that are hand-entered and assumed to be unchanging. For example, in the 3T architecture [6], which appears to be becoming the standard for NASA, the planner is only used for a few percent of the applications [18], and then only for resource scheduling.

One particularly extreme project is the GOLOG project at the University of Toronto. This is an attempt to develop a complete architecture based on first-order logic with the full situation calculus. The focus of this project is inferential power, not real-time performance or sensor grounding. Although the group has made some initial tests on real hardware, the majority of the work has been done in simulation.

There are a number of projects that are focusing on explicit representation of teamwork. The most applicable example is Tambe's STEAM system [7] that provides high-level group coordination for teams of robot helicopters and robot soccer players. Again, this project depends on a full traditional symbolic inference system, so porting it to run on real robots involves a large engineering effort. However, the group is actively working on a real-hardware entry for the RoboCup soccer tournament. Other implemented collaboration systems for software agents are Jennings' GRATE* system [19], and Rich and Sidner's COLLAGEN system [20], which is an application-independent toolkit for human/computer interaction with software agents based on explicit models of shared intentions. All of these systems are based on the frameworks of shared intentions [21] or Shared Plans [22].

Mataric [8] has proposed to achieve robustness, reliability, and (implicitly) code reuse in distributed multi-robot systems by using *basic behaviors*. The idea is to develop a fixed set of sensory-motor systems that are sufficient to jointly generate the entire space of desired behaviors. This reduces programming to a problem of designing the arbitration unit, for which Mataric used variants of reinforcement learning. Basic behaviors drastically simplify the learning problem, although it is hard to know when one has a good generating set, or whether a universal generating set might exist.

2. Methods, Assumptions, and Procedures

This section describes the basics of variable binding and inference in role passing. For specific information on sensory-motor systems, the natural language parsers, or the details of the current implementation, see [1,23].

Ultimately, the most expensive operation in inference is variable binding. Inference without variables (propositional inference) is easily parallelized. Inference with variables but no term expressions or other data structures (e.g. DATALOG) can be performed in polynomial time, but is believed not to be parallelizable (because it's P-complete). Inference with term expressions then becomes progressively more expensive as different combinations of quantifiers are allowed. Full first-order inference is only semi-decidable. That is, although algorithms exist that can prove any true theorem, no algorithm can identify all false theorems, even in principle.

The task, then, in designing efficient inference systems is to find limited forms of variable binding that can be efficiently implemented. We have chosen to develop the most expressive representation we can that can still be completely parallelized. By compiling the inference rules into a parallel network, we not only get good performance, but automatic updating of inferences as sensor data changes. A simplified data flow during inference is shown in Figure 1.

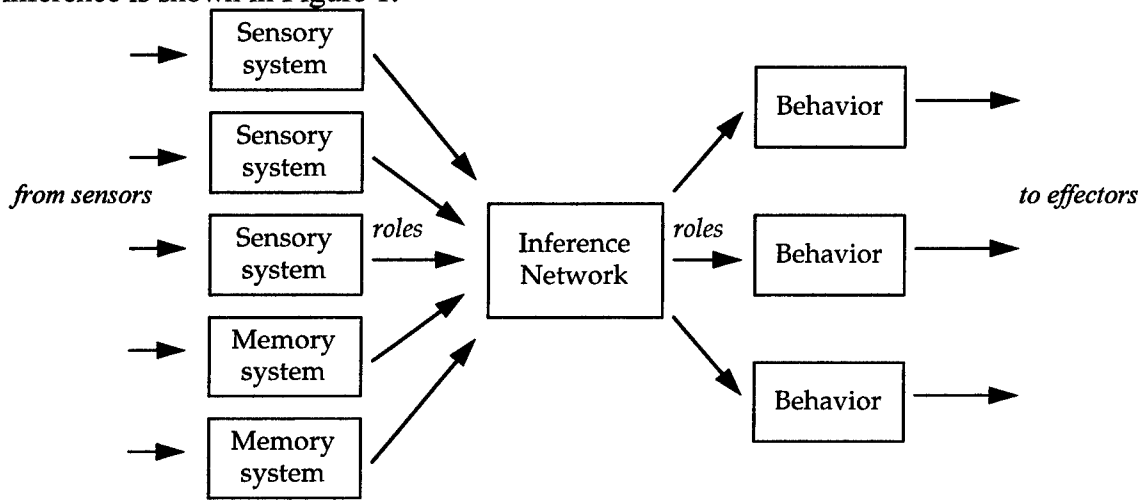

Figure 1. Simplified data flow during inference

2.1 Binding variables in sensory and memory systems

We implement variable binding by allowing a sensory or memory system to be tagged with the name of the variable to which the object it represents is bound. We refer to these tags as *roles* because they correspond to the semantic roles of frame structures (e.g. *agent*, *patient*, *source*, *destination*, etc.). When a sensory/memory system is initially allocated to track/represent an object, a role is also specified for that sensory/memory system. We then say that that system, and the object it represents, is *bound* to that role.

We constrain the set of allowable roles in advance to some reasonably small set (32 in our current implementation, although a few hundred would be manageable). We can then represent an arbitrary *set* of roles as a bit-vector. A bit vector representation of role sets is shown in Table 1.

Table 1. Bit-vector representation of role sets

Agent	Patient	Source	Destination
1	0	1	1

We can represent an individual role by a singleton set (a set with only one element) and so we can represent the role to which a given sensory or memory system is bound by a bit-vector (this also allows us to bind an object to multiple roles, if we so desire).

Now suppose we have some perceptual systems that can determine whether the objects they track are nearby (e.g. a set of color blob trackers in an active vision system). We can compute the set of objects that are near the robot simply by ORing together the bit vectors of those systems whose targets are nearby. For example, suppose we have four color-blob trackers bound to agent, source, patient, and destination, respectively. A computation of the bit vector representation of near(x) is shown in Table 2.

Table 2. Computing the bit-vector representation of near(x)

	Near?	Agent	Patient	Source	Destination
Tracker 1	**Yes**	**1**	**0**	**0**	**0**
Tracker 2	No	0	0	1	0
Tracker 3	**Yes**	**0**	**1**	**0**	**0**
Tracker 4	No	0	0	0	1
Bitwise OR		**1**	**1**	**0**	**0**

This last row gives us the set of roles of the objects that have been measured to be nearby. It is effectively the extension of the predicate *near(x)* under a closed world assumption. We can rapidly compute this extension on any computer architecture, serial or parallel, given the nearness measurements of the trackers and the bit-vectors to which they are bound. We can use the same basic technique to compute scalar-valued functions, like *distance(x)*, however we must use a vector of numbers rather than a bit vector to represent the output. Each tracker drives the output for *distance(r)* with its measured distance iff it is bound to *r*. Again, this can be computed efficiently on both parallel and serial machines.

2.2. Inference in role-passing systems

Now suppose we have sensory systems that compute both the predicate *near(x)* and the predicate *facing(x)*. Then we can implement the inference rule as shown in Table 3:

For all x, grabable(x) $\Leftrightarrow$ near(x) and facing(x)

using bitwise and:

Table 3. Inference using bit vectors

Predicate	Agent	Patient	Source	Destination
near(x)	1	1	0	0
facing(x)	0	1	1	1
grabable(x)	0	1	0	0

Note, again, that this is efficiently computable on arbitrary machine architectures, from microcontrollers to FPGAs. On stock hardware, the inference would compile to the C code:

```
grabable = near & facing;
```

assuming there are no more than 32 roles, so that an integer variable can store the complete bit-vector for the extension of a predicate.

2.3. Reasoning about knowledge and goals

This representation for predicates makes no distinction between x not being near, and x 's distance being unknown (e.g. because no tracker is tracking it). Since modern robots have active perception systems whose attentional resources must be explicitly allocated to specified objects, it is important for those robots to be able to represent and reason about their own attentional states and states of knowledge. We can do this by computing a set of valid bits for each predicate in parallel with its extension by ORing together the bit vectors to which each tracker is bound (regardless of whether the tracked object is near or not). For example, as shown in Table 4:

Table 4. Computing the know() modal for a sensory primitive

	Near?	Agent	Patient	Source	Destination
Tracker 1	Yes	**1**	0	0	0
Tracker 2	No	0	0	**1**	0
Tracker 3	Yes	0	**1**	0	0
Tracker 4		0	0	0	0
know(near(x))		**1**	**1**	**1**	0

(here we have assumed that tracker 4 is as yet unallocated, so it is not bound to any role). We can compute the valid bits for derived predicates, like *grabable(x)*, from the valid bits of the predicates from which they are derived as shown in Table 5:

Table 5. Computing the know() modal for an inferred predicate

Predicate	Agent	Patient	Source	Destination
know(near(x))	1	1	1	0
know(facing(x))	1	1	1	0
know(grabable(x))	1	1	1	0

(This assumes the axiom *know(a(x) and b(x))* $\Leftrightarrow$ *know(a(x) and know(b(x))* ; stronger axioms can also be implemented).

It is also desirable to introduce modalities to represent whether *near(x)* is a goal of the system, or even whether *know(near(x))* is a goal. Again, each of these can be represented with a single machine word, so the total space used by the system is four machine words per predicate in the system. Using this representation, it is possible to write inference rules like:

for all x, goal(know(near(x))) $\Rightarrow$ goal(see(x))

which says that seeing an object is a way of knowing its distance.

2.4. Motor control

Ultimately, the robot has to do things in the world. To fire actions, the inference network computes the bit-vector of the object to perform the action on and passes it as input to a conventional behavior-based system that implements the action. The behavior will need to obtain sensory data about the specified object, so it will pass on the bit-vector to an appropriate set of perceptual systems that can provide the information the behavior needs. The perceptual systems then compare the bit-vector to the bit-vectors they are bound to. When a perceptual system finds a match, it routes its information to the behavior. If no perceptual system has the information needed, no information will be sent to the behavior, and the behavior will be disabled.

The inference system can then implement control rules like:

if goal(in-hand(x)) and grabable(x) then grab(x)

which corresponds to the chunk of circuitry in Figure 2.

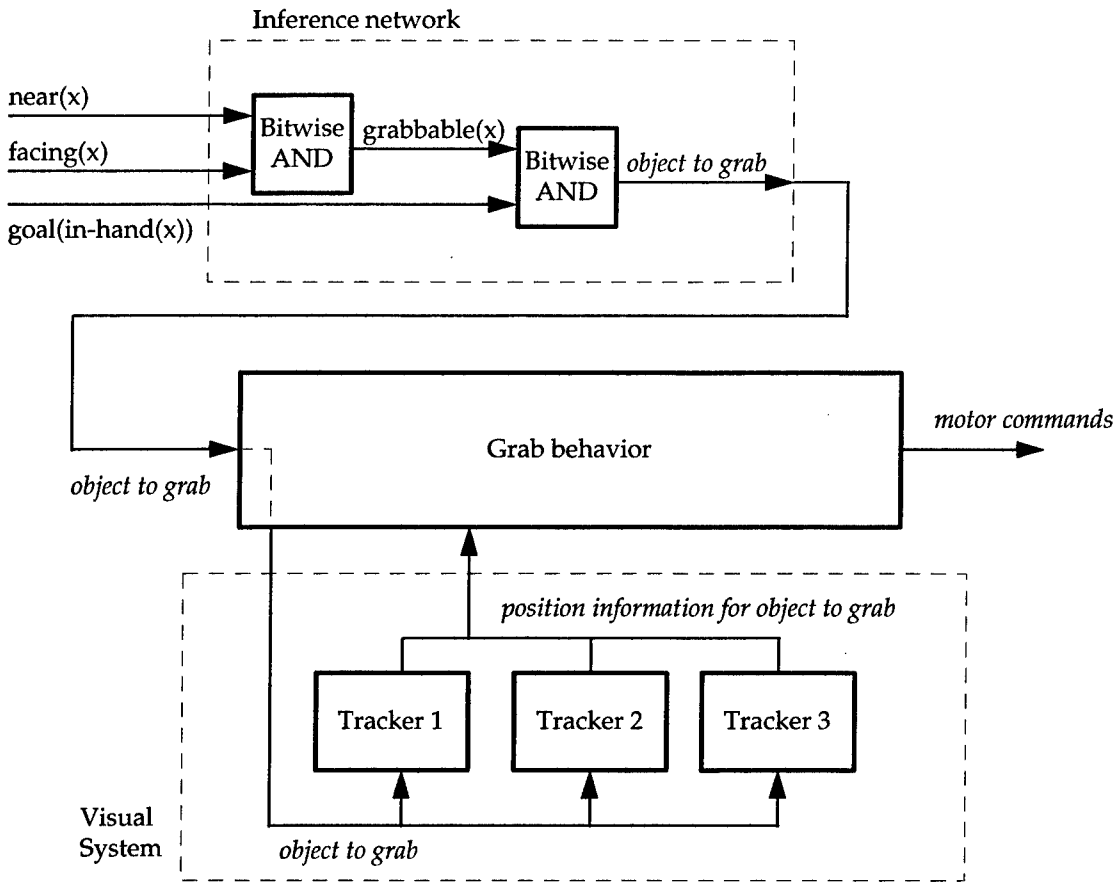

Figure 2. Data flow for firing the grab behavior

The inference network computes the bit-vector for the object to grab (if any) from the bit-vectors for near(x), facing(x), and goal(in-hand(x)). The bit-vector for the object to grab is sent as input to the grab behavior, which forwards it to the visual trackers. The trackers compare it and route the appropriate information back to the behavior. If there is no object the system wants to grab (or can grab), then the bit-vector routed to the grab behavior will be all zeros. In this case, no tracker can match it, and so the behavior is disabled. More complicated control rules are also possible, such as:

if goal(see(x)) and not see(x) and know(color(x)) then start-visual-search-for(x)

which states that if the robot is trying to see an object and knows its color, it should tell the visual system to allocate a tracker to it an initiate a visual search for it

2.5. The HIVEMind System

We have developed a distributed role passing architecture called *HIVEMind* (Highly Interconnected Very Efficient Mind) that supports very efficient sharing of symbolic information between team members. It combines the efficiency and responsiveness of traditional behavior-based systems and the high-level expressiveness of symbolic systems.

Figure 3 shows an abstract HIVEMind configuration for a two-robot team. Each team member has its own inference network. The network is driven both by its own sensory system and by the incoming data from the other team members. Outputs from the current robot's sensory systems are fed into aggregation functions on other team members. The output from those aggregation functions is then fed into the inference rules that drive the motor behaviors.

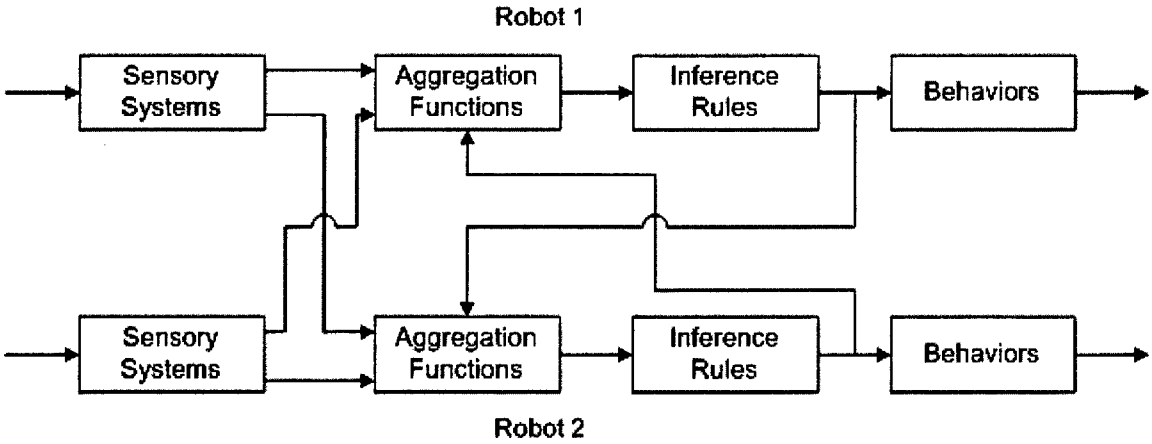

Figure 3. Abstract view of HIVEMind

The aggregation functions are used to combine information from teammates and sensors into a single coherent output for the inference rules to reason over. In an n robot team, each robot's inference network has n distinct sets of inputs, one generated internally, and the rest received from the robot's teammates. These distinct inputs are first fused into a single set of inputs:

$$K = \beta(k_1, k_2, \ldots, k_n)$$

where the k_i are the tuples of inputs from each robot, K is the final fused tuple, and β is some aggregation function that performs the fusion. For example, if a particular component of the input was a proposition, the aggregation function might simply OR together the corresponding components of the k_i. Thus the robot would believe the proposition if and only if some robot had evidence for it. In more complicated cases, fuzzy logic or Bayesian inference could be used. Real-valued data is likely to require task-specific aggregation. For example,

1. The team is assigned to scout an area and report the number of enemies observed. Each team member has a slightly different count of enemy troops. In this case, the best solution is probably to average the disparate counts.
2. The task is "converge on the target". Each robot's sensors report a slightly different position for the target. In this situation, it appears to make sense that each team member rely on its own sensor values to track the target and only rely on other robots when the robot's own sensors are unable to track the target, e.g. the target is out of sight.

Figure 4 shows how aggregation is performed in the actual system. As packets arrive on from other robots, they are unpacked into buffers for their respective robots, replacing whatever data had been stored previously for that robot. In parallel with this process, the main control loop of the robot aggregates the inputs from each robot and reruns the inference rules on the result. These inference rules then enable and disable low-level behaviors for sensory-motor control. Since the main control loop is performing real-time control, it runs much faster than the 1Hz update used for communication (10Hz in our current implementation).

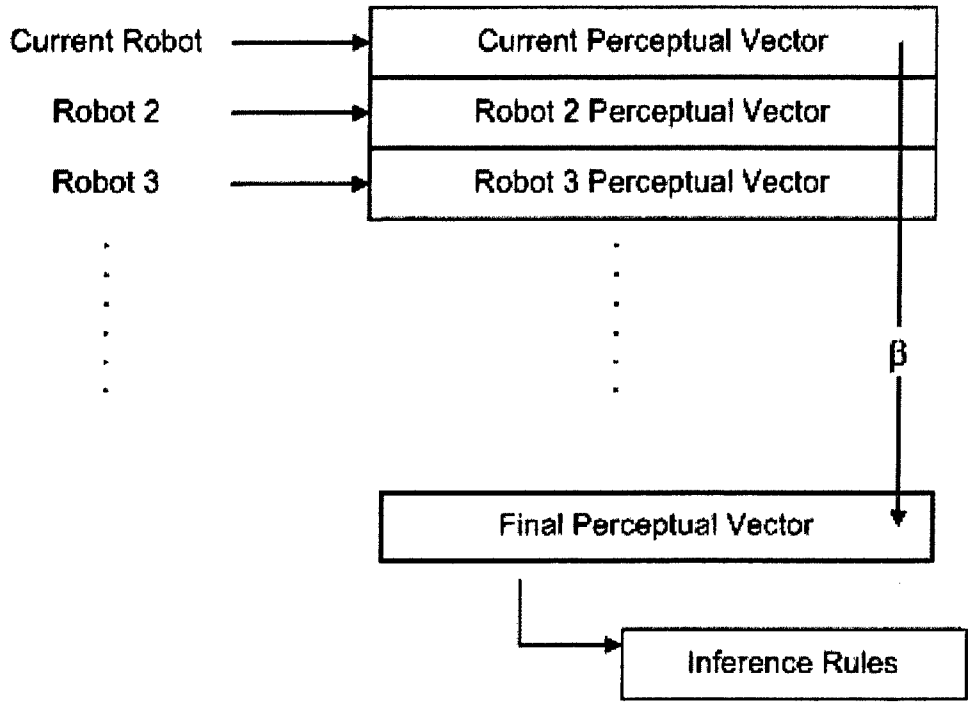

Figure 4. Data aggregation in HIVEMind

The entire HIVEMind can be considered a single, parallel control network whose components happen to be distributed between the different robot bodies being controlled. Wires crossing between bodies are simulated using the RF broadcast mechanism, so that each member of the team is "connected" to every other member in a web-like structure of virtual wires. In our current implementation, each robot broadcasts its sensory data and state estimates in a single UDP packet at predefined intervals. Presently, broadcasts are made every second. Faster or slower rates could be used when latency is more or less critical. However, 1Hz has worked well for our applications. To reiterate, we expect that currently implementable robot systems could store all the sensory inputs to the inference system in a single UDP packet (1024 bytes). Current autonomous robots are severely limited in their task capabilities, and hence their communication needs, by their sensors and actuators. Therefore, we feel that there is plenty of available bandwidth for communication in the foreseeable future. As robots develop more complicated sensoria, it may be necessary to use more complicated protocols, perhaps involving multiple packets, or packets that only contain updates for wires whose values have changed since the last transmission. For the moment, however, these issues are moot.

Given the current single-packet-protocol, the aggregate bandwidth required for coordination is bounded by 1KB/robot/sec, or about 0.1% of a current radio frequency (RF) local area network (LAN) per robot. Thus robot teams on the order of 100 robots should be practical from a communication standpoint. However, hardware failure limits most current robot teams to less than 10 members, so scaling limits are difficult to test empirically.

2.6. SEEKers

We have implemented the HIVEMind system on a robot team that performs three tasks involving the systematic traversal of space:

1. Find Static Object
   The team systematically searches for a brightly colored object in a known environment. Team members explore the environment in a systematic manner until one of the team members locates the object or all searchable space is exhausted. When the object is found, all team members converge on its location.
2. Town Crier
   This task involves making announcements in the same known environment. The team cooperatively travels to each landmark on a map and makes an announcement at every landmark.
3. Capture Evading Target
   Similar to task 1, except now the target can maneuver. The team members must cooperate to trap the evading object, which in this case, is a human.

Task 1 is considered solved if all team members arrive at the general location of the target and halt. Task 2 is complete when all the landmarks have been visited. Finally, task 3 is solved when the target can no longer maneuver, i.e. every immediate corridor the target could turn to is occupied by a robot. The team automatically loses if the target ever moves to an escape point and leaves the search area. In Figure 5, there is only one exit point – the Gauntlet. So reaching Andrew Ortony's office would be considered victory for the target.

The commander specifies the task to perform using an operator control unit (OCU). The commander can monitor and direct the team through the OCU, which appears as an additional, albeit non-performing, member of the team. The OCU passes command and control data to the team through the same virtual wires mechanism used for inter-robot communication. We have tested all tasks with a two-robot team.

The robots have a topological map of the area to be searched, which is assumed to be an indoor office environment. All tests discussed here were performed in the west wing of 1890 Maple Ave (Figure 5). A robot team on search mission is shown in Figure 6.

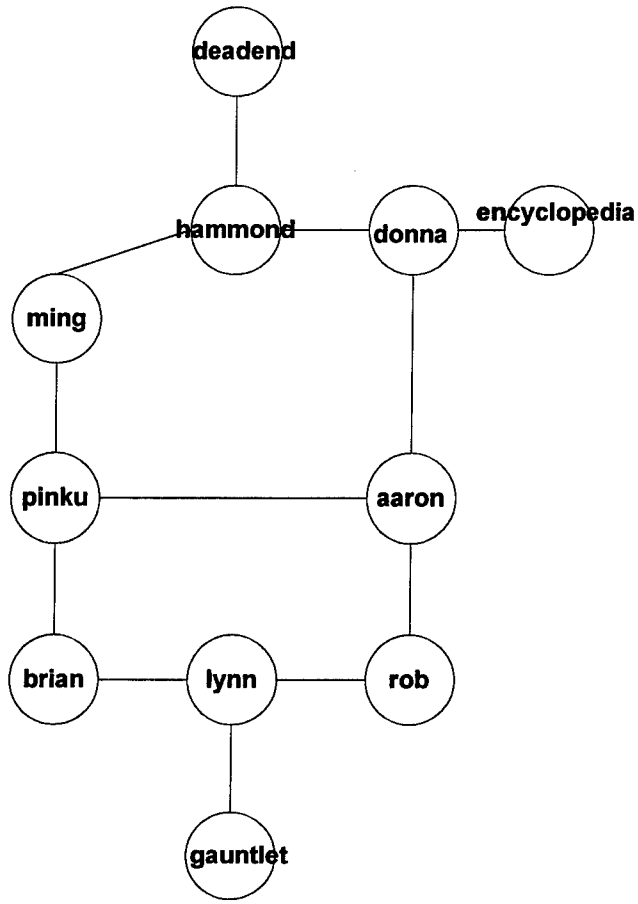

Figure 5. The Computer Science Department West Wing

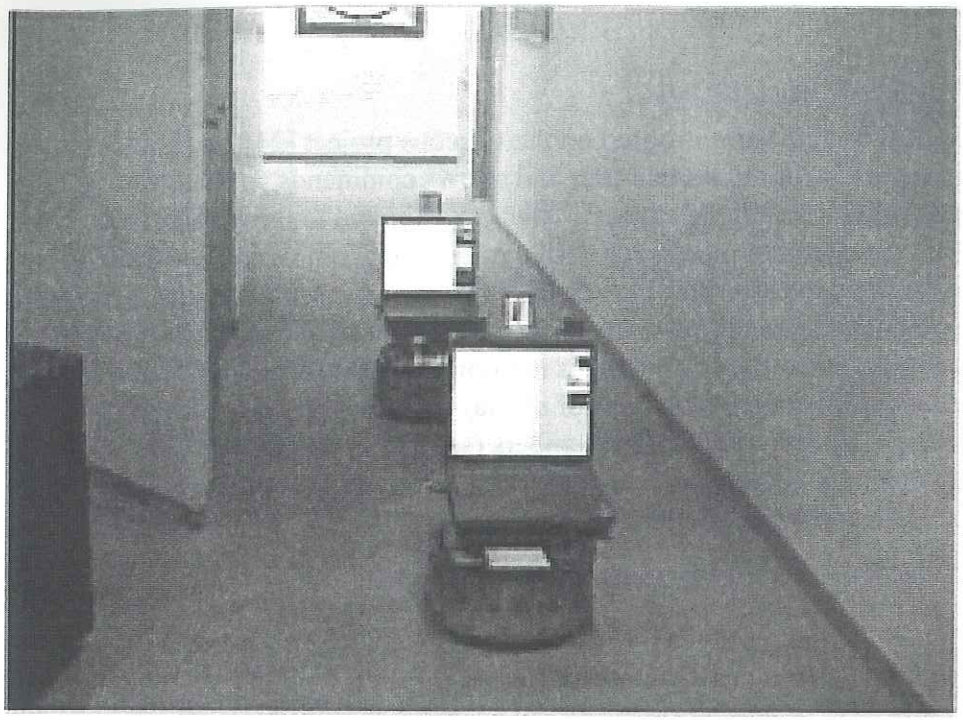

Figure 6. Robot team on search mission

2.7. Hardware infrastructure

The robotic bases used in this experiment are first generation Real World Interface (RWI) Magellan bases. The Magellan provides sonars, infrared sensors and bump switches, a total of 16 each, arrayed around the circular base. Visual sensing is provided by a ProVideo Charge-Coupled Device (CCD) camera connected by a Nogatech frame grabber to a laptop. The computing platforms used were COTS laptops – Dell Latitude CPis with 450MHz Pentium II processors, 384Mb of RAM and 11Gb hard drives. The laptops ran Windows98, and communicate with the base through a serial cable. RF COMS used the IEEE 802.11b standard running at 11Mb/s (Lucent Orinoco Silver PCMCIA cards).

2.8. Software infrastructure

High level control, including the role-passing system, was implemented in Generic Robot Language (GRL), an architecture-neutral language for behavior-based systems [25]. GRL programs take the form of high-level descriptions of parallel networks of signals that are updated continuously. The GRL compiler constructs a custom circuit-simulator for these networks and compiles it to a single while-loop containing straight line code. The output code can be C, C++, BASIC or Scheme (a dialect of LISP).

Lower-level code for image processing, network I/O, and serial I/O were written largely in C++, with some additional Scheme code.

2.9. Operator Control Unit (OCU)

The OCU for the HIVEMind team is based on the Cerebus project [23]. It provides a natural-language interface for the human user and allows commands such as 'find green ball' or 'announce "talk at 7!"' to be entered. The task is bound to the `activity` role, and any arguments are bound to other appropriate roles, e.g. green would be bound to `object` in the former example. The current bindings are represented in alist form and transmitted on a virtual wire to all members to the team. The OCU appears as another robot to other team members, albeit one that is not doing any physical work. The OCU also provides status information in the form of display windows based on the broadcast knowledge it is receiving from other team members (see Figure 7). Using this interface, the human commander can inject new information into the team, as well as receive data about the current state of the "group mind".

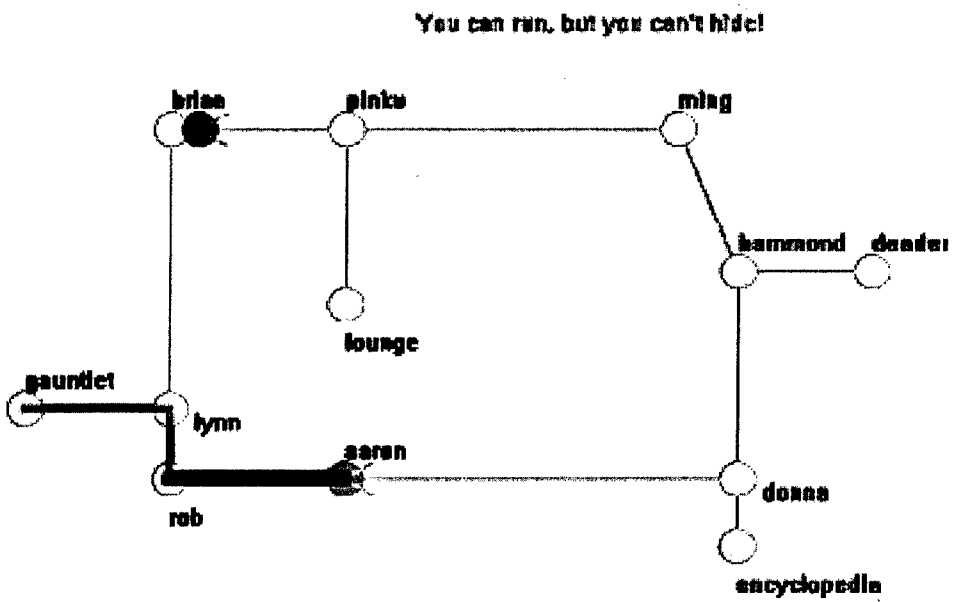

Figure 7. Map display from OCU

2.10. Sensory Systems

Most sensory systems in HIVEMind are grouped into pools. Trackers are allocated from these pools on demand whenever a robot needs to attend to an object. The pools drive the inference rule network, which in turn drives the low-level behaviors that actually control the robot. Figure 8 shows a high-level view of the system.

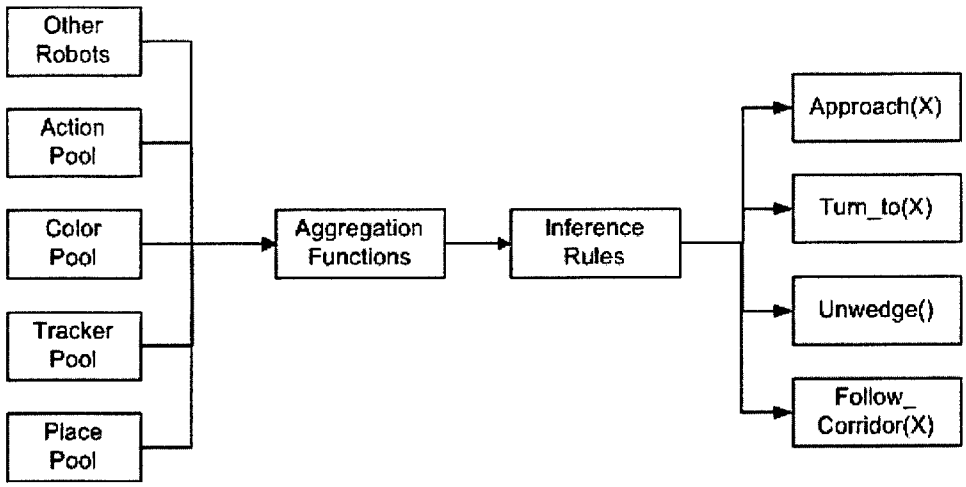

Figure 8. Control flow from sensors to behaviors in a single robot

The *tracker pool* consists of a set of visual trackers that utilize a variant of k-means clustering for tracking blobs of color in the robots visual image. The trackers can drive low-level behaviors with image-plane coordinate of the objects they track. In addition, they generate the low-level predicates `see-object(X)` and `near-object(X)` for input to the inference network. The trackers are not used in the Town Crier task.

The color pool is a memory system stores color coordinates of different objects in a format suitable for use by the visual tracking system. Although not a sensory system *per se*, it allows colors to be specified to the tracking system by binding them to roles. For example, when the user directs the team to seek a green ball, the OCU binds the color pool entry to green to the appropriate role. The bindings are then automatically passed over the network to the robots.

The *place pool* is a probabilistic localization system that uses a topological, i.e. landmark-based, map. Roles can be bound to landmarks and the system can determine the next appropriate waypoint in order to reach a landmark specified by role. The place pool also records the set of landmarks that have been visited with high probability and can determine the closest unvisited landmark. The current map contains 11 landmarks distributed over the west wing of the 3rd floor of the Northwestern Computer Science Department.

The *action pool* is used to allow the robot to "sense" the state of its own internal behaviors, as well as to control them. It is used to allow the OCU to explicitly invoke behaviors.

2.11. Communication

The following information is communicated during performance of all tasks:

1. The current role bindings, including bindings for the current activity or task, and any bindings for pertinent arguments
2. A bit-vector specifying the set of landmarks that the robot has personally visited
3. The bit vector for the `see-object(X)` predicate
4. An array representing the `location(X)` function, which give the two nearest landmarks, if known, for any role X

All of these are low-level outputs of the various pools, except for the current role bindings, which has to be stored on a separate latch on the user console. When the team is performing the Town Crier task, the latter two communication structures, i.e. `see-object(X)` and `location(X)`, are not utilized for reasoning.

2.12. Inference rules

One of the advantages of role-passing is that it allows relatively compact specification of control tasks through inference rules. The inference rules for the find object task are:

1. If `see-object(X)` is true, then `goto(X)`.
2. If `location(X)` is known, and `see-object(X)` is false, then `goto(location(X))`.
3. If `location(X)` is unknown, and `see-object(X)` is false, then `goto(next-unsearched-location())`.

The inference rules for the town-crier task are:

a. If `at-landmark(X)` and `not-announced-at(X)`, then `speak-string()`.
b. If `true`, then `goto(next-unsearched-location())`.

The inference rules for the Capture Evading Target task are somewhat more complicated:

a. If `not(observed(X))`, then `goto(next-unsearched-location())`.
b. If `see-object(X)` or `location(X)` is known, then `set-observed(X)`.
c. If `not(all-see(X))` and `see-object(X)`, then `set-current-role(sentry)`.
d. If `current-role()` is *sentry*, then `stop-moving()` and `announce-taunt()`.
e. If `not(see-object(X))` and `location(X)` is known, then `set-current-role(stalker)`.
f. If `current-role()` is *stalker*, then `goto(trapping-path(location(X)))`.
g. If `all-see(X)`, then `goto(X)` and `announce-capture()`.

The seven rules above are responsible for moving the robot between three states: search, stalk and trap. The robots start out in search mode (Rule 1) looking for the intruder. When the intruder is seen, the robots transition out of search mode (Rule 2). The robot

that sees the intruder goes into sentry mode (Rules 3 & 4), while the other robot becomes the stalker (Rules 5 & 6). The stalker plans a path to the location of the intruder that takes into account the location of the sentry. That is, with the sentry on one end of the intruder's corridor, the stalker enters that corridor from the other end, trapping the intruder in between the two robots. Finally, when both robots have the intruder in sight, they move in for the kill (Rule 7).

The function `next-unsearched-location()` returns the current location if there are no new locations to travel to. `Goto()` is a behavior that drives to a specified object, as specified by a role. However, its exact behavior depends on the sensory system in which the object is bound. If the argument is bound to a location, then the robot will navigate to that landmark. If the argument is bound to a color in the color pool, then the robot approaches the largest object matching that color in its view. `Goto()` activates the four behaviors described below as necessary to accomplish its current task.

2.13. Behaviors

There are four motor behaviors that drive the robot:

1. *Approach* drives to an object specified by role. It attempts to keep the object in the middle of its visual image.
2. *Turn-to* swivels the robot to face a new direction. It is used when the robot arrives at a landmark and needs to turn in a new direction to reach another landmark.
3. *Unwedge* activates when the robot becomes stuck in some corner unexpectedly. It swivels the robot in the direction in which it thinks has the greatest open space so the robot can continue moving.
4. *Follow-corridor* navigates the hallways. It tries to remain centered in the middle of the corridor to facilitate easy recognition of environmental features.

The behaviors are arbitrated strictly through a priority stack. Behaviors that are higher on the stack have higher priority, and, if active, will be chosen to run over those of lower priority. Since HIVEMind always ensures that all team members are up-to-date on the current situation, each robot always knows the appropriate behavior to activate for the current situation and no conflict between team members arises.

3. Results and Discussion

We have tested the system with a three-member team consisting of two robots and the command console. The team was tested in the west wing of the 3rd floor of the Computer Science Department building. The wing consists of a network of six corridors spanning an area approximately 6 × 20m with an aggregate path length of 50m. The network of corridors is represented by 12 landmarks in the topological map showing the locations of features such as corners and intersections. The robots drive at approximately 1m/s on straightaways, although stopping for ballistic turns at corners and intersections somewhat reduces their mean velocity. Sensing, inference and control decisions are each performed at 10Hz.

3.1. Search for a static object

In the Find-Object experiments, all team members were started from a central point at the extreme east end of the wing. The goal object, a green ball, was placed out of view, 15-20m from the starting point. The object was always at least two corridors and three landmarks away from the starting point. When the command "find green" was entered on the command console, the robots begin a systematic search of the wing for the goal object. Unlike stochastic search techniques such as foraging, the systematic search guarantees that each landmark is searched at most once and that all landmarks are guaranteed to be searched, if necessary. Using a greedy algorithm for landmark selection, the team was consistently able to find the landmark within 30 seconds provided that there were no catastrophic failures of the place recognition system. On typical runs, the team found the object in approximately 20 seconds. A robot team member in the Find Object task is shown in Figure 9.

Figure 9. Team member in Find Object task

3.2. Systematic traversal of space

For the Town Crier task, team members were again started from a central point at the extreme east end of the wing. The objective was for the robots to go through each landmark at least once, making the announcement at each landmark that the robots passed through. If a robot had already spoken at a particular landmark, then no further announcement should be made there, since we do not wish to inundate any nearby offices with multiple announcements. Again, barring any catastrophic failures of the place recognition system, the team was able to complete the task successfully.

3.3. Search for an Evading Target

Finally, for the Capture Evading Target experiment, the robots were started from the same position as the Find Object experiments. As in the static target case, the localization code formed the greatest impediment. The inference architecture performed flawlessly once again; the relevant inferences and sensory data were transparently shared between all team members (including the commander console) in an efficient manner. Figure 7 shows a screenshot of the status windows on the commander console. The highlighted corridor between the nodes labeled *rob* and *aaron* shows the location of the human intruder. The second (stalker) robot is near the *brian* node, performing a pincer move in order to come around and trap the human. Figure 10 (left) shows the intruder in front of the stalker robot. When the intruder attempts to escape to the other corridor, the stalker doesn't follow, but instead lets the other robot trap the intruder. In the end, the intruder was trapped in a corner, as shown in Figure 10 (right).

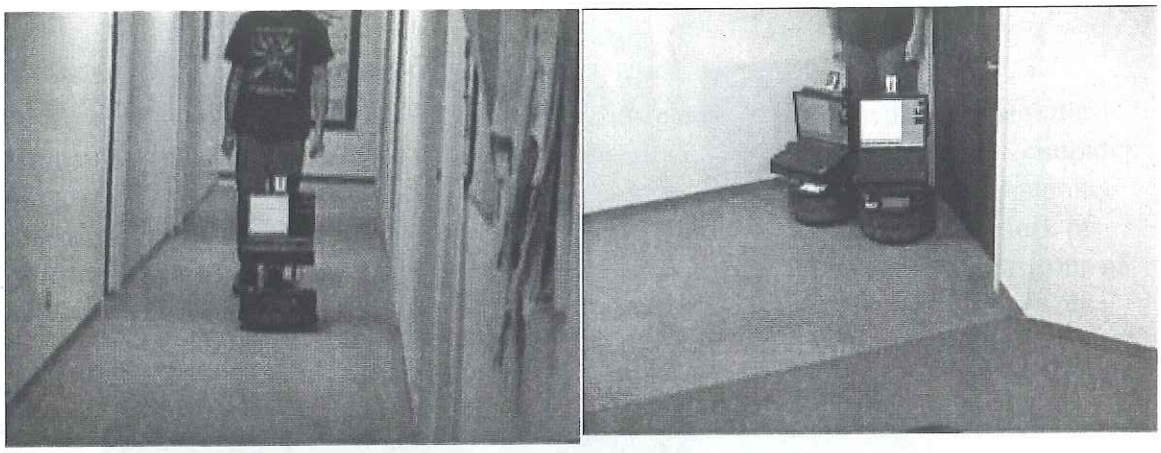

Figure 10. Team capturing an evading target

3.4. Discussion

The place recognition system is the weak point of the current implementation. Minor errors are common and occasional catastrophic failures can cause one of the team members to think that it has traversed its intended destination when in fact it has not. While we are working on improving the place recognition system, it should be stressed that the actual control and coordination architecture worked without error.

It may seem inefficient for each robot to have its own separate copy of the inference network. However, to have a single robot perform each inference and share the results would require much more complicated coordination protocols [21] analogous to the multi-phase commit protocols used in distributed database systems. Since communication bandwidth is a scarce resource and inference in our system is essentially free, it is more efficient for HIVEMind robots to perform redundant computation.

Limitations

Although HIVEMind has performed very well in our experiments, it does rely on a *convergence property* for the shared data. This requires that all team members will converge on identical world models in finite time, provided of course that the world doesn't change in the intervening period. This is guaranteed by the timestamping protocols used on packets provided that those packets contain *state information* not *event information.* That is, provided that the true state of the world can be inferred from the most recently transmitted packets without having to have also received any past packets. The problem is illustrated by the transition diagram shown in Figure 11.

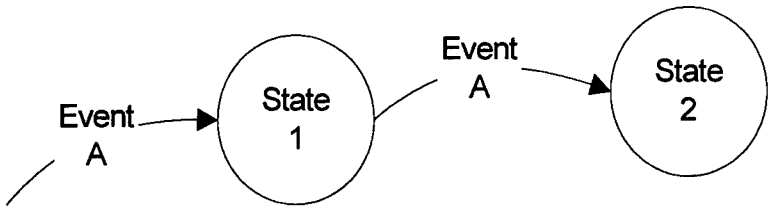

Figure 11. Problematic state transitions

If the robots communicate state information (e.g. "I'm in state A"), the situation is non-problematic. If, however, the robots share only event information, then packet loss can lead to desynchronization/decohesion of the team when the loss of an A report causes some team members to believe the system is still in state 1 instead of state 2. The problem can be solved by replacing UDP with some kind of reliable multicast protocol such as Scalable Reliable Multicast (SRM) [24].
However, would require robots to wait for their team members to acknowledge old communications before sending new communications, leading the team to fail entirely should one member fall temporarily out of communication. Another approach would be to transmit the last few events in each packet rather than just a single event. The system could be made arbitrarily reliable at the cost of increased communication bandwidth by increasing the size of the event buffer. In our work, however, we have chosen to solve the problem by transmitting the state information directly rather than by transmitting update events.

Another issue is the limited physical range and geographical coverage. Wireless networks have finite range. If the task area is large enough, team members will fall out of communication range. These issues could potentially be solved using tree-based algorithms such as prefix scans. However, we have not yet attempted this.

4. Conclusions

The HIVEMind architecture is an attempt to extend parallel reactive inference to a multi-robot environment. It allows behavior-based systems to abstract over both objects and sensors, while remaining efficient enough in both inference speed and bandwidth consumption to be usable on physical robot teams. HIVEMind provides multi-robot system designers with more powerful representations than behavior-based systems, and has a simple, efficient model for group coordination. We believe that the right set of representational choices can allow the kinds of inference presently implemented on robots to be cleanly grounded in sensor data and reactively updated by a parallel inference network. By continually sharing perceptual knowledge between robots, coordination can be achieved for little or no additional cost beyond the communication bandwidth required to share the data. The effect is a kind of "group mind" in which robots can treat one another as auxiliary sensors and effectors. We have implemented and validated the architecture on three tasks and achieved very good results.

References

1. I. Horswill. *Grounding Mundane Inference in Perception.* In Autonomous Robots, 5, pp. 63-77, 1998.
2. R.C. Arkin. *Behavior-Based Robotics.* MIT Press. Cambridge, MA. 1998.
3. P. Maes. *Situated Agents Can Have Goals.* Robotics and Autonomous Systems, Vol. 6, pp. 49-70.
4. Takaaki Hasegawa, Y. I. Nakano, and T. Kato. "A Collaborative Dialog Model Based on Interaction Between Reactivity and Deliberation." In *Proceedings of the First International Conference on Autonomous Agents.*, Marina Del Rey, CA, 1997.
5. C. Dwork, P. Kanellakis, and J. Mitchell. "On the Sequential Nature of Unification." *Journal of Logic Programming* 1:1, pp. 35-50.
6. R. Peter Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. "Experiences with an Architecture for Intelligent Reactive Agents." *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1997). Special Issue on Software Architectures for Physical Agents, H Hexmoor, I. Horswill, and D. Kortenkamp, eds.
7. Millind Tambe. Towards Flexible Teamwork (extended abstract). In *Socially Intelligent Agents*, AAAI technical report FS-97-02. AAAI Press, Meno Park, CA, 1997.
8. Maja Mataric. "Behavior-based Control: Examples from Navigation, Learning, and Group Behavior." *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1997). Special Issue on Software Architectures for Physical Agents, H Hexmoor, I. Horswill, and D. Kortenkamp, eds.
9. L.E. Parker *ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation*, IEEE Transactions on Robotics and Automation, Vol. 14, No. 2, April 1998.
10. Michael Sahota. "Reactive Deliberation: An Architecture for Real-time Intelligent Control in Dynamic Environments." In *Proceedings of the Eleventh Nactional Conference on Artificial Intellgience.* MIT Press, 1994.
11. Rodney Brooks. "A Robust Layered Control System For a Mobile Robot." *IEEE Journal of Robotics and Automation* 2:1 (1986) pp. 14-23.
12. Stan Rosenschein and L. Kaelbling. "A Situated View of Representation and Control." *Artificial Intelligence* 73(1-2) (1995) . Special double-volume on Computational Theories of Interaction and Agency, Rosenschein and Agre, eds.
13. Philip Agre and D. Chapman. "Pengi: An Implementation of a Theory of Activity." In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA. 1987.
14. Nils Nilsson. "Teleo-reactive Programs for Agent Control." *Journal of Artificial Intelligence Research*, 1994.
15. Lokendra Shastri and V. Ajjanagadde. "From Simple Associations to Systematic Reasoning: A Connectionist Encoding of Rules, Variables, and Dynamic Bindings Using Temporal Synchrony." *Behavioral and Brain Sciences*, 16:3 p. 417-494.

16. Marvin Minsky. "Plain Talk on Neurodevelopmental Epistemology." In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA. 1977.
17. Russell, S. and P. Norvig (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.
18. David Kortenkamp, personal communication.
19. N. Jennings. "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions." *Artificial Intelligence* 75 (2). 1995.
20. Charles Rich and C. Sidner. "Collagen: When Agents Collaborate with People." In *Proceedings of the First International Conference on Autonomous Agents (Agents 97)*, Marina Del Rey, ACM Press, 1997.
21. P. R. Cohen and H. J. Levesque. "Teamwork." *Nous* 35. 1991.
22. "Collaborative Plans for Complex Group Actions." *Artificial Intelligence* 86, 1996.
23. I. Horswill, R. Zubek, A. Khoo, C. Le, and S. Nicholson(2000) *The Cerebus Project.* In the AAAI Fall Symposium on Parallel Cognition and Embodied Agents, 2000.
24. D. Eckhardt, P. Steenkiste. *Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network.* In Proceedings of the ACM SIGCOMM '96, pp. 243-254, August 1996.
25. Horswill, I. (2000). "Functional Programming of Behavior-Based Systems," *Autonomous Robots* 9, 83-93. Kluwer Academic Publishers, the Netherlands.