Implementing Efficient Joint Beliefs on Multi-Robot Teams

Aaron Khoo

Northwestern University Computer Science Department
1890 Maple Avenue
Evanston, IL 60201
khoo@cs.northwestern.edu

Abstract

Traditional symbolic reasoning systems are typically built on a transaction model of computation, which complicates the process of synchronizing their world models with changes in a dynamic environment. This problem is exacerbated in the multi-robot case, where there are now \( n \) world models to keep in sync. Synchronizing these distributed world models is analogous to database replication across a set of mobile nodes. My thesis will show that replicating beliefs across a set of mobile nodes is a highly problematic issue. Instead, I will propose an inference grounding and coordination mechanism for multi-robot teams that is based on knowledge-base broadcast and tagged behavior-based systems. This new approach supports a large subset of classical AI techniques while providing a novel representation that allows team members to share information efficiently. Finally, I will illustrate this approach on three tasks involving systematic spatial search.

Introduction

Autonomous robots that reside in complex, dynamic environments must anchor the abstract representations they use to actual physical objects. Since the world around the robot continually changes, its sensory systems must track those changes. In turn, the robot’s control systems must be ready to alter plans and actions to suit its changing model of the world. Traditional symbolic reasoning systems are typically built on a transaction-oriented model of computation. Knowledge about the world, or the “world model”, is stored in a database of assertions in some logical language, indexed perhaps by predicate name [26]. Populating this database from a highly dynamic environment is a difficult and non-trivial problem [16]. In a cooperative multi-robot team, the group members must keep a consistent set of joint beliefs in order to coordinate their behavior in a coherent manner. Maintaining these joint beliefs is equivalent to performing world model synchronization across multiple distributed platforms. In this paper, I make the argument that world model synchronization is in fact analogous to database replication, and this has serious consequences for maintaining consistency in a multi-robot team. Furthermore, I propose an alternative coordination approach based on periodic knowledge-base broadcast that alleviate these issues, and describe three implementations based on this new approach. Finally, I conclude by proposing a tentative timeline for completing the remaining work for this thesis.

World Model Synchronization

Overview

When changes in the environment occur often, the world model must also be updated frequently, or the reasoning system will operate on stale data. In addition, assertions in the world model database can be dependent on other assertions. For example, the assertion that an area is safe could depend on the assertion that the robot does not currently observe any predators in the area. If the latter assertion is withdrawn, then the former must be too. Hence, each update from the perceptual systems can trigger a cascade of further transactions, resulting in additional load on the system. In principle, modifying such a system to track changes in the environment would require recording dependencies between stored assertions and their justifications such that when the perceptual system added or retracted an assertion, the reasoning system could enumerate and update the set of existing assertions affected by the change. This is a sufficiently complicated process that we know of no implemented physical robots that do it.

Keeping the knowledge-base synchronized with the external environment becomes even more difficult in cooperative activity. Rather than one robot with a single knowledge-base, we now have \( n \) robots with \( n \) knowledge-bases to keep consistent both with the world and with one another. Failure to maintain a consistent set of joint beliefs between team members could lead to system delusion [14], i.e. the databases are now inconsistent, and there is no obvious way to repair them, resulting in failure to coordinate activity.

Joint Intentions and Replicated Databases

Researchers in the multi-agent community have developed coordination protocols for controlling teams of agents. The best known of these is the joint intentions framework [6],
Joint Intentions specifies a formal theory of team commitment towards a specific goal, and the conditions under which that commitment is enacted or dissolved. The joint intentions are dependent on mutual beliefs, which are infinite conjunctions of beliefs about other agents’ beliefs about other agents’ beliefs (and so on to any depth) about some proposition. Instead of considering mutual beliefs, I propose to simply consider joint beliefs, i.e. the mutual agreement of two agents on the truth value of propositions. A joint belief on a proposition between two agents creates an obligation in each agent to inform the other if there is a change in the belief state or truth value of that proposition. That is, joint beliefs is concerned with the act of synchronizing the truth value of a proposition between agents on the same team. For example, two agents could have a joint belief on the proposition \( (\text{see-enemy}) \). Therefore, if one agent observes an enemy, it is obligated to communicate this to its partner through some mechanism. In a sense, joint beliefs are the simplest case of mutual beliefs; I do not consider the deeper belief structures (beliefs about beliefs about beliefs ad infinitum) of mutual beliefs.

Figure 1: Lazy Replication (left) vs. Eager Replication (right)

We are interested in performing an analysis of the mechanism that implements joint beliefs. We begin by observing that if the world model used in the symbolic reasoner is transaction-based, then the act of maintaining joint beliefs across different team members can be modeled as a set of replicated databases. Since the robots are distributed geographically, team members may observe events their teammates are not privy to. If relevant data is not properly shared (i.e. replicated) across all participating world models, team members will not be able to properly coordinate.

Lazy Replication There are two general models of replication: lazy and eager (see figure 1). Lazy replication algorithms asynchronously propagate replica updates to other nodes after the updating transaction commits. However, these algorithms suffer from the problem of stale data. When two transactions read and write data concurrently, one transaction’s updates should be serialized after the other’s. Failure to do so can lead to conflicts, which are defined as inconsistencies between the contents of two databases, e.g. two different values for the same database object. Conflicts occur when two different databases attempt to update the same object, or race to install their updates at other databases. Whenever conflicts occur, the replication mechanism must detect this and somehow reconcile the two transactions so that their updates are not lost.

To ground this in a specific example, suppose we have a team of robots assigned to find and attack a mobile target. At some point, one of the robots spots a strong hostile formation coming towards the team. This is asserted in the robot’s own database, and a further assertion of (current-goal (retreat)) is inserted into the knowledge-base. The information about the hostile formation is passed to other team members, but this information takes time to propagate. In the meantime, another team member spots the original target. This robot asserts the knowledge in its world model, and further asserts (current-goal (attack-target)). Then it begins transmission of this information to other team members.

Now the team members face a set of conflicting goals. If the assertion (current-goal (attack-target)) was inserted first, when information about the hostile formation came in, it should be withdrawn and replaced with (current-goal (retreat)). In the reverse case (i.e. the hostile formation is known before the target position), then the retreating goal should be maintained over the attacking one. However, as I pointed out previously, enumerating and updating dependent assertions can be a difficult and complicated affair. This conflict needs to be resolved appropriately for the team to perform correctly. Otherwise, different team members will have incompatible goals, and some team members might even have conflicting goals asserted. Furthermore, this example highlights the fact that conflicts do not have to occur at the level information is exchanged. There is no conflict between the assertions that both the target and a hostile formation have been observed. Rather, the conflict occurs on assertions that were dependent on them.
An analysis of asynchronous peer-to-peer replicated databases by Gray et al. [11] reveals a serious issue. Under the following assumptions --

- The databases are updated through lazy group replication, i.e. the originating database updates its entries, and then propagates the update to other replicas asynchronously.
- Each node updates any other database location with equal probability.
- All nodes impose an equal load on the system.
- There are a fixed number of objects per transaction.

Gray et al. were able to show that the conflict rate per second is:

$$O\left(\frac{r^2 a^3 t n^3}{s}\right)$$

where

- $r$ is the number of transactions per second initiated by each node.
- $a$ is the number of locations updated per transaction.
- $t$ is the time required to complete an update.
- $s$ is the number of distinct entries in the database.
- $n$ is the number of nodes (which, in our case, is equivalent to the number of robots) in the system.

The critical point here is that the number of conflicts encountered by the system increases with the cube of the number of nodes or robots. As Gray et al. point out, “having the reconciliation rate rise by a factor of a thousand when the system scales up by a factor of ten is frightening”.

Finally, note that message propagation times are not presently part of the conflict model as presented above. If message delays were added to the model, then each transaction would last longer, hold more resources and generate more conflicts. Moreover, mobile robots necessarily communicate via wireless links, which are well-known to have higher error rates [10, 32], and hence higher message delays, than their wired counterparts. This analysis suggests that we could potentially face serious scalability issues for any physical multi-robot system with a database-driven knowledge model. The work necessary to reconcile the conflicts that could arise as team members tried to communicate knowledge to other members could eventually overwhelm the robots, or leave them badly out of synch.

**Eager Replication** The key issue here is that eager replication schemes have trouble handling disconnected nodes. Joint Intentions requires that a decision be ratified by all team members. In real-world scenarios, communication links are not always guaranteed. Links between teammates could easily go down, and then be restored some time later. Suppose the network that the robots rely on for communication is unexpectedly partitioned. After an appropriate amount of time, a timeout is generated and the partitioned “subteams” continue with the mission assuming their missing teammates are “dead”. Each subteam makes decisions based on their own knowledge independent of the other subteams. Each new decision causes new assertions to be inserted into the knowledge-base for each robot. As more time passes and more independent decisions made, the world models for each subteam diverges more and more. When communication between the subteams is finally restored, the subteams are now hopelessly out of synch with one another, and there is no obvious way to re-synchronize the knowledge-bases.

Finally, the Joint Intentions specification views communication as a black box that passes messages between team members reliably and instantaneously. A naïve implementation of the communication protocol would have reliable connections between team members. Under this arrangement, each message to be passed to all team members would generate $O(n)$ packets because of the acknowledgement messages, where $n$ is the number of robots in the team. Suppose that on each processing cycle, some percentage $p$ of the team transmits a message, i.e. $pn$ robots transmit a message per cycle. Intuitively, this means that the number of packets required to synchronize the team is $O(n^2)$. This would be somewhat expensive as we scale to larger teams.

**Figure 2: Typical Tiered Architecture**

**Related Work**

**Symbolic reasoners in physical robots** There has been some recent progress towards the development of a formal framework for anchoring symbols in a physical environment [8, 9]. However, most implemented physical robots take the approach of equipping the symbolic system with a set of domain-dependent epistemic actions that fire task-specific perceptual operators to update specific parts of the knowledge-base. The programmer designing the
knowledge-base is responsible for ensuring that the proper updates are done, i.e. the right epistemic actions are fired at the appropriate times. This alleviates some of the difficulties of getting information into the knowledge-base in a timely manner. However, any mistakes by the programmer will lead to inconsistencies between the knowledge-base used by the symbolic system and the external environment. Tiered architectures, such as [2,4,7], that combine symbolic and behavior-based systems inherit these model coherence issues, because their symbolic layer still relies on a database-driven world model for its reasoning process.

Such model coherence issues are exacerbated in a cooperative environment where multiple knowledge-bases have to be synchronized. The programmer is now responsible for maintaining coherence across multiple distributed platforms connected via tenuous wireless links. We feel that this has led to a paucity of physical multi-robot systems utilizing symbolic reasoners. The only physical multi-robot team that utilizes both a symbolic reasoner (in a tiered architecture) and active communication that we know of is the DIRA system [27].

Figure 3: Communication via virtual wires

Behavior-based approaches Most existing multi-robot controllers implemented on physical systems focus on extending traditional behavior-based techniques [1] to a team environment (for example see [3,12,25]). Behavior-based systems allow rapid response to changes in the environment due to tight sensor-actuator integration. Many of these behavior-based systems also obey circuit semantics, which means their control programs are generally implemented as feed-forward circuits. This simplifies the communication structure necessary to maintain coordination between team members. Essentially communication in these behavior-based multi-robot controllers is reduced to virtual wires connecting the appropriate circuitry on one team member to another's. The wires carry relevant information from a robot to its counterparts. Conversely, each robot views its teammates simply as additional sensory input, and integrates the incoming information as appropriate. Conveniently, virtual wires can be simulated on physical robots using a broadcast communication mechanism such as User Datagram Protocol (UDP).

However, this convenience is not without cost. The strengths of the behavior-based approach are also its weakness. Circuit semantics impose a propositional representation on the reasoning system, i.e. representations without predicate/argument structure. Propositional representation makes most reasoning and planning tasks both difficult and clumsy since they require redundant copies of the system for each possible argument to a predicate or action [24]. Since most multi-robot controllers are extensions of behavior-based techniques, they inherit the same issues from the basic underlying architecture.

Other robotic techniques Some systems have attempted to solve the synchronization problem through techniques other than active communication. [31] proposes an approach that achieves coordination through teammate observation and plan inference. [21] utilizes stochastic techniques that allow the individual robots to achieve a global goal through the use of simple non-interference behaviors. Some systems in the robocup small-sized robot league utilize a traditional symbolic reasoner that relies on a central shared world model [30]. In this case, reasoning is performed at a central server location where the master knowledge-base is located, and then actions are transmitted to the individual robots. Little, if any, reasoning is done on the client side.

In this paper, we only consider team models where members are fully autonomous entities with independent decision making ability, i.e. the robots are not reliant upon a central reasoner. Each robot is responsible for deciding its own course of action. Furthermore, we are focusing on domains where passive communication or stochastic techniques are insufficient. That is, team members will
Our efforts in this direction have resulted in HIVEMind (Highly Interconnected Very Efficient Mind), a multi-robot control architecture that supports very efficient sharing of symbolic information between team members. The HIVEMind architecture is built on role-passing [16], a type of tagged behavior-based system [19]. Role-passing provides the developer with a limited set of domain-independent indexical variables (called roles) such as agent, patient, source, destination, etc. When a role is bound to an object, a tracker is dynamically allocated to it and tagged with the name of the role. Since the number of roles is relatively small, we can represent the extensions of unary predicates as bit-vectors, with one bit representing each role. This representation allows inference to be performed using bit-parallel operations in a feed-forward network.

Alternatively, for commodity serial hardware, we can represent a unary predicate extension using a single machine word. Inference rules can then be compiled directly into straight-line machine code consisting only of load, store, and bit-mask instructions [16]. While more limited than a full logic-programming system, it does allow us to express much of the kinds of inference used on physical robots today. The inference rules can be 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. In short, role-passing affords us the ability to implement traditional inference rules using circuit semantics.

In addition to allowing very fast inference, this representation allows for very compact storage of a robot’s current set of inferences. Unary predicates are stored in one machine word. Function values are represented using small arrays indexed by role. This compactness, combined with the circuit semantic nature of role-passing, allows us to take full advantage of simplified communication mechanism described previously, i.e. virtual wires connecting team members. In fact, for the kinds of tasks currently implemented by multi-robot teams, the representation we use is sufficiently compact to allow all function and predicate values of a robot to fit into a single UDP packet. Robots can therefore share information by periodically broadcasting their entire knowledge-base, or at least all those predicates and functions that might be relevant to other team members.

**Characteristics of the HIVEMind**

**Overview** Knowledge-base broadcast is a simple communication and coordination model that provides each robot with transparent access to every other robot’s state, establishing 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 are immediately informed and respond within one update cycle without the need for negotiation protocols. Moreover, since HIVEMind systems are based on role-passing, multi-robot controllers implemented using this architecture have greater representational power and flexibility than pure behavior-based systems with propositional representations. That is, our communication is not based on passing propositional values such as see-blue-object or see-red-object, but rather predicates such as see-object(X). Furthermore, since all relevant team knowledge is continuously being rebroadcast, each member’s knowledge-base converges to the same state within one update cycle of joining the HIVEMind. This means that team members can be brought online and integrated into the HIVEMind very easily, allowing us to add or subtract team members dynamically. In addition, should communication fail for some time, the team would very rapidly return to a common state when it is restored.

**Negotiation** The design of the HIVEMind architecture negates the need for negotiation amongst team members. The act of negotiation is required under three situations; I will briefly show that none of these three issues occur under the current HIVEMind model of cooperative activity.

First, negotiation is required if the cooperating agents are self-interested and do not necessarily share common goals. Hence, negotiation is required to ensure that all sides obtain a satisfactory resolution to any issues. Since the teams that HIVEMind is designed for are fully cooperative, this condition is not applicable.

Even if the agents are fully cooperative, team members may have different views of the same situation. That is, some team members might lack information that
necessitates a particular course of action. For example, suppose a member of a helicopter squad observes approaching tanks. This observation requires that the team delay, and perhaps even abandon its next objective. To do so, the team must come to a common understanding of the situation, with the agent observing the key event responsible for conveying that observation to all team members. The Joint Intentions framework [6] is specifically designed to account for situations like these. Under the HIVEMind model, however, pertinent information is continually being rebroadcast at a periodic rate. Therefore, all team members share a common situational awareness, and there is no necessity for additional mechanisms to ensure that all relevant data is communicated.

Finally, it is conceivable that even with access to complete information, team members differ on the course of action to take. Specifically, each member has its own internal plan, and at least some of the plans differ. Again, negotiation is necessary to ensure that the team takes coherent action. Otherwise, it is conceivable that the team members could execute plans which are redundant, or even contradictory in nature. For the moment, let’s consider the situation where two robots have different plans. There are two possibilities: one plan is better (by some metric) than the other, or the plans are equally good. In the first case, given that HIVEMind teams always have a common situational awareness, it should be trivial to determine which plan is better given the team’s current knowledge. In the second, we simply need to devise an arbitrary tie-breaker to determine which plan “wins”; in the HIVEMind case, since each robot has a unique id number, the lowest id number always wins in the case of a tie. It should be easy to see how the two robot case generalizes to an $n$ robot team.

![Figure 4: Heterogeneous goals in HIVEMind](image)

**Heterogeneity** Following on the above discussion, the HIVEMind architecture’s negotiation-free approach might appear to imply that a uniform inference network across all team members is required. This in turn could imply homogeneity in the team’s physical forms, hence limiting the team’s usefulness. However, I will show that the inference networks on the robots do not have to be completely identical.

Suppose we had a team of heterogeneous robots where a subset of the team had special responsibilities; as long as all current goals are parallelizable, there are no contention issues. The team members can be as opportunistic as possible. If conflicting goals arise, or if some goal can only be achieved by a subset of team members, then our goal tree will have the form in Figure 4.

Figure 4 shows the subgoal hierarchy for two heterogeneous robots. Subgoal1 and Subgoal2 are serial goals, i.e. Subgoal1 has to be achieved before Subgoal2. In this case, when Robot1 is busy achieving Subgoal1, Robot2 will remain idle executing *no-op*. When Subgoal1 has been accomplished, Robot2 will instantly know since the relevant information is continually being shared between the two robots. Hence, both robots will immediately move on to the subsequent subgoals. To take a concrete example from [29], consider a team consisting of multiple helicopters with a scout in the team. While the scout performs reconnaissance, the other helicopters lie in wait. Only when the scout is finished do the others move up to join it.

**Implementing the HIVEMind**

Figure 5 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 which drive the motor behaviors.

![Figure 5: Abstract View of Hivemind](image)

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 \( \beta \) 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,

- 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.
- 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 6 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).

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 RF 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.

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 [6] 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.

**Possible Issues** One possible problem is the assumption of a convergence property for the shared data. That is, we assume that the data is timestamped, and any data received that is earlier or equal to the latest timestamp is discarded. This accounts for the possibility of the UDP packets getting dropped, arriving out of sequence or being duplicated. However, for some applications, this restriction will be unacceptable. Specifically, if the inference rules contain transitions of the following sort:

Upon the first observation of Event A, the inference rules transition into State 1. The second observation of Event A
causes a transition into State 2. The problem here is that UDP is not reliable. Hence, one of the following situations could occur:

- Both Event A's could be transmitted successfully.
- Only one of the Event A's is transmitted successfully, trapping the robot in State 1.
- Both Event A's are dropped, meaning that the transition into State 1 is never made.

![Figure 7: Problematic Transitions](image)

UDP messages could also potentially be duplicated at the destination. However, this can be easily fixed with a simple timestamp and dropping any old messages.

There are remedies for this unreliability, of course. One simple approach would be to use a reliable protocol such as TCP. Such alternative communication mechanisms will be discussed in the next section. For the SEEKers task described later however, no such transitions exist. So using the simpler UDP broadcast protocol is acceptable.

One more possible approach is to transmit the current state of the agent rather than the events. For example, instead of transmitting *see-rock-fall*, an agent could continuously transmit *rock-fallen*.

Another issue is the limited physical range and geographical coverage. The wireless devices do not require the existence of a base station, i.e. they are capable of peer-to-peer transmission. However, they do not have an infinite range. If the task area is large enough, it is possible that some team members will be out of each other's transmission range. Note that the current HIVEMind implementation assumes that all team members will be within transmission range of all other members.

**Alternative Implementations** In this section, I will briefly explore some alternative implementations for the virtual shared memory structure that the HIVEMind architecture relies on for communication. These alternative implementations attempt to address some of the issues for the current UDP-based approach.

One concern, as discussed in the previous section, is the unreliability of the UDP protocol. One simple solution in the case of figure 7 is simply to transmit the current state rather than the triggering events.

Another possible solution to this is the use of a reliable protocol such as TCP for the transmission of our data. This would solve the issues of dropped or duplicated messages. However, as I pointed out previously, using a reliable protocol balloons the number of packets required to transmit a message. Since our messages are intended to be very lightweight, this overhead could become quite significant and have an impact on transmission rates.

Another approach is to extend the UDP messages. Instead of sending just the current values, the message will contain a vector of the last *k* values of each predicate and function, where *k* is a tunable value set by the user. The messages would still be timestamped so old or duplicated messages are ignored. However, we would be ensured that all events would be received and processed correctly. There are obvious disadvantages to this approach however. First, less information could be packed into each message since now we need to send *k* values per predicate or function. Second, the time needed to process the messages has also increased, since a more complex algorithm is required to analyze incoming packets and ensure all events are processed correctly.

One more approach that bears future investigation is the use of reliable multicasting techniques such as SRM (Scalable Reliable Multicasting) [11].

**Implementation**

**SEEKers**

**Overview** We have implemented the HIVEMind system on a robot team that performs three tasks:

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...
The team automatically loses if the target ever moves to an escape point and leaves the search area. In Figure 6, there is only one exit point – the Gauntlet. So reaching Andrew Ortony’s office would be considered victory for the target.

The robots obviously operate at a physical and mental disadvantage for task 3. So, to make things even, I defined some basic rules of fairness:

1. No jumping over the robots. Our robots cannot help being short. Furthermore, one could easily imagine a larger robot performing this task so that the human could not dodge it in this manner.
2. The human must stay in the corridor at all times. Specifically the human is not allowed to climb objects like tables to avoid being seen. Again, our robots are short and their cameras only point downwards.
3. The human is not allowed to damage the robot. Throwing things at the robot to disable it is not fair unless the robot is allowed to fight back with tasers.

At present, I am assuming that the robots will have a topological map of the area they will be operating in at start time. One possible future extension would be to relax this assumption and have the robots dynamically map the area as they searched. The search area is assumed to be an indoor office environment. In particular, the test area shall be the west wing of 1890 Maple Ave. Some of the labels in figure 8 are no longer accurate, since office assignments have changed over time.

Hardware 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. Vision is provided by a ProVideo CCD camera, connected by a Nogatech USB video capture adaptor cable to a laptop. The laptops are Dell Latitudes with Pentium II 450Mhz processors, 384Mb of RAM and 11Gb hard drives. They run Windows98, and communicate with the base through a serial cable. Remote communication is provided by Lucent Orinoco Silver wireless Ethernet cards that feature an 11Mbps data transfer rate under the IEEE 802.11b standard.

Software The main development work will be done in Generic Robot Language (GRL). GRL is an architecture neutral language embedded within Scheme [17]. Programs written in GRL ultimately look like networks of signals which are conceptually computed in parallel and updated continuously. These signals are compiled into a single while-loop containing straight line code. The output code can be C, C++, BASIC or Scheme. For my purposes, the GRL code will be compiled into Scheme. The AMRG group has also written a host of useful functions located in GRL libraries, including popular behavior-based architectures such as motor-schemas and subsumption, a graphical package called iview, probabilistic and fuzzy logic routines, remote communication utilities, etc. In the course of development, I will naturally use a number of these libraries.

I will resort to native Scheme code as necessary. Hooking native Scheme to GRL is a fairly straightforward endeavor, and it is occasionally simpler to implement certain routines using the full functional power of Scheme rather than GRL. When real speed is required, such as in the vision processing procedures, the preferred language is C/C++. The C/C++ routines are compiled into DLLs, which are then linked to Scheme via foreign function calls.

Figure 8: Map of Computer Science West Wing

In all tasks, a human user is responsible for indicating the current task to perform and supplying any required parameters for that task, e.g. the properties of the object to be found in the first task. The human interacts with the team through a user console, which appears as an additional, albeit non-performing, member of the team. When user input is entered into the console, that information is passed through the virtual wires to all team members. We have tested all tasks with a two robot team.
**User console** The Command Console for the HIVEMind team is based on the Cerebus project [16]. 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 a list form and transmitted on a virtual wire to all members of the team. The console appears as another robot to other team members, albeit one that is not doing any physical work. The user console also provides status information in the form of display windows based on the broadcast knowledge it is receiving from other team members. 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”.

**Perceptual Systems** The sensory and memory systems are divided into “pools”, which are useful abstractions for grouping perceptual systems or descriptions of objects. Note that we do not make any unique claims about pools; they are simply convenient abstractions for implementing role-passing. The pools drive the inference rule network, which in turn drives the low-level behaviors that actually control the robot. Figure 9 shows a high-level view of the system.

![Figure 9: Control flow from sensors to behaviors in a single robot](image)

The action pool stores a set of reified user-defined plans that can be bound to roles at runtime. These plans can then be run by calling the role to which they are bound. For example, the find plan is bound to the role activity when the user enters “find green ball” at the console. The binding is passed via virtual wire to the individual team members. So, when the control system calls activity, it would run the find plan. There are currently three plans in the action pool: find, announce and sentry.

The color pool stores color coordinates of different objects in a format suitable for use by the visual tracking system. The color of a desired object can be specified by binding a given color description in the pool to the role of the object. For example, when the user directs the team to seek a green ball, the term green is bound to an appropriate role. The bindings are then automatically passed over the network to the robots. The color pool presently contains descriptions for red, green, and blue objects, and is only used for the find object task.

The tracker pool consists of a set of trackers that utilize a variant of k-means clustering for tracking blobs of color in the robots visual image. These trackers can be allocated and bound to a role. 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 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.

**Communication** The following information is communicated during performance of all tasks:

- The current role bindings, including bindings for the current activity or task, and any bindings for pertinent arguments
- A bit-vector specifying the set of landmarks that the robot has personally visited
- The bit vector for the see-object(X) predicate
- 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.

**Inference Rules** The inference rules for both tasks are fairly simple. This is partly due to the continual recomputation of inferences, which alleviates the need for some error detection and recovery logic that would otherwise be necessary. The inference rules for the find object task are:

1. If see-object(X) is true, then goto(X).
Find Object Figures

Figure 10: Two robots leaving on their search task

Capture Evading Target Figures

Figure 11: One member of the team finding the ball

Figure 12: Screenshot of console status window

Figure 13: Sentry robot taunting human intruder

Figure 14: “We have you now, human scum!”
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:
1. If at-landmark(X) and not-announced-at(X), then speak-string().
2. If true, then goto(next-unsearched-location()).

The inference rules for the Capture Evading Target task are somewhat more complicated:
1. If not(observed(X)), then goto(next-unsearched-location()).
2. If see-object(X) or location(X) is known, then set-observed(X).
3. If not(all-see(X)) and see-object(X), then set-current-role(sentry).
4. If current-role() is sentry, then stop-moving() and announce-taunt().
5. If not(see-object(X)) and location(X) is known, then set-current-role(stalker).
6. If current-role() is stalker, then goto(trapping-path(location(X))).
7. 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 polymorphic action keyed by the type of the argument passed to it. 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.

Behaviors There are four motor behaviors that drive the robot:
- **Approach** drives to an object specified by role. It attempts to keep the object in the middle of its visual image.
- **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.
- **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.
- **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.

Results 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 6mx20m 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.

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.

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.

Finally, for the Capture Evading Target experiment, the robots were started from the same position as the Find Object experiments. These experiments were run with the aid of some fellow graduate students, as I performed the role of the evading target. They ran the commander console and the camcorders during the experiment. 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 12 shows a screenshot of the status windows on the commander console. The highlighted corridor in 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 13 shows me in front of the stalker robot, which is taunting me with phrases like “You can run, but you can’t hide”. When I attempt to escape to the other corridor, the stalker doesn’t follow me, but lets the other robot trap me instead. Then, finally, the hapless human is trapped near Ming’s lab, as shown in Figure 14.

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.

**Thesis Plan**

This thesis is intended to show that traditional symbolic reasoners with transaction-based world models will face serious difficulties in a distributed multi-robot team; as an alternative, it proposes an approach that ensures synchronization while avoiding the model coherence issues that symbolic systems encounter. This new approach, called HIVEMind, has been successfully implemented on a physical robot team that performs three different tasks.

Remaining work on this thesis:

- A closer analysis of joint intentions/mutual beliefs vs. joint beliefs.
  The argument that joint beliefs is a simpler case of mutual beliefs is not well-formed at this moment. We are currently working on a paper that develops the connection between the two, as well as the database replication argument.
  **Proposed completion date:** November 30, 2002

- Analysis of HIVEMind failure modes
  - HIVEMind is essentially using timestamp locking in database parlance. I am currently surveying the database literature in this area to better understand the pitfalls in the use of this approach.
  - Role assignment and race conditions.
    There are possible dangers during role assignments in HIVEMind. For example, two agents could oscillate role assignments and get into a race condition where no useful work is done.
  - Passing state instead of events.
    I proposed passing state instead of events in order to alleviate the problems of unreliable communication. However, it is not completely clear what the issues are when we resort to passing the state rather than events.
  **Proposed completion date:** January 31, 2002

- Finish writing dissertation.
  **Proposed completion date:** April 15, 2003

**References**


