

NORTHWESTERN UNIVERSITY

Implementing Efficient Joint Beliefs on
Multi-Robot Teams

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

by

Aaron Boo Boon Khoo

EVANSTON, ILLINOIS

June 2003

Copyright © 2003, Aaron Boo Boon Khoo

All Rights Reserved

ABSTRACT

Implementing Efficient Joint Beliefs on Multi-Robot Teams

Aaron Boo Boon Khoo

In this report, I investigate the problem of coordinating multiple robots in a cooperative team activity. Many different approaches have been suggested for team coordination, ranging from swarm techniques to symbolic methods. I will propose a class of coordination protocols called broadcast-an-aggregate mechanisms and will describe a specific instantiation of this approach on a multi-robot control architecture called HIVEMind (Highly Interconnected Very Efficient Mind) which operates in the following way:

- Each robot periodically broadcasts all its team-relevant data to its teammates.

- Team members base control decisions on shared situational awareness formed by aggregating all broadcast data from the team.

HIVEMind allows near-instantaneous sharing of data between all team members. This lets the team respond in real-time to contingencies sensed by individual robots. Furthermore, HIVEMind robots can use role-passing inference to support limited real-time symbolic reasoning over synchronized knowledgebases. This allows the use of more structured representations than conventional behavior-based systems, which in turn allows the team to be tasked and monitored in natural ways. All this is accomplished while using surprisingly little bandwidth. I will show that given the assumptions:

- Relevant aspects of the environment change relatively quickly.
- All team members must be informed of these changes in bounded time.
- All team members must be able to detect if they are failing to receiving data in a timely manner from their teammates.

the HIVEMind data sharing method is optimal in the sense that any communication scheme would have to transmit at least as many packets as it would. I also will describe an implementation of HIVEMind used to coordinate a team of robots in variety of tasks involving systematic search of physical space.

Acknowledgements

It is good to have an end to journey towards; but it is the journey that matters, in the end.

-- Ursula K. LeGuin

As a person, I have always been more about the destination than the journey. However, as the end of this particular journey draws near, I realize that my experience would have been that much poorer but for the wonderful people that I have met along the way.

The first person I really need to thank is my advisor, Ian Horswill, who gave me the freedom to find my own voice and bearing, while making sure I never wandered too far into the wilderness. Without his support, advice, encouragement and gentle prodding in the right direction, I would not be here today. I must also thank my committee members – Ken Forbus, Peter Dinda and Tucker Balch – for their time, patience and advice.

The people you interact most on an everyday basis in graduate school are, of course, your fellow students. They form a vital part of your education that cannot be quantified but surely cannot be dismissed. Dac Le was an officemate for three long years and acted as an invaluable sounding board when I needed someone to talk to while putting up patiently with my occasional rants and raves. Robert Zubek continues to be a close friend and collaborator extraordinaire who provides equal doses of

inspiration, insight and skepticism; our work together has yielded many interesting ideas, and hopefully will keep on doing so far into the future. Robin Hunicke's sense of humor and take on life keep me in good spirits when frustration threatens to seep in. Ayman Shamma both a good scientist and artist, and an awesome racquetball partner to boot. Praveen Paritosh has contributed good insights and provoked many interesting debates about various aspects of artificial intelligence. Michael Knop was a valued partner-in-crime in the online gaming world. The Flexbot developers, particularly Greg Dunham, Nick Trienens and Sanjay Sood, are further proof (as if more is needed) that Northwestern undergraduates are smart, motivated students. They have earned my eternal gratitude for helping to create an avenue for research that is not only interesting but fun. Vern Chapman, the "anti-Aaron", no matter the differences in our programming philosophies, you are still a great developer and fellow Chicago Bulls fan. Pinku Surana, Jason Skicewicz , Magy Seif El-Nasr, Cuong Pham and Shawn Nicholson are as good friends as one could possibly wish for.

Finally, my love and gratitude go out to my family. My wife Juliana has been nothing but supportive throughout this long journey and has always been there for me when I needed someone to lean on. My daughter Adalea was born only a week and a half before my thesis defense (ensuring I had two significant milestones in my life in short order) and continues to be our pride and joy. I love you both very much.

Table of Contents

Acknowledgements.....	v
Table of Contents.....	vii
Table of Figures.....	xiii
Table of Tables.....	xvi
Chapter 1 Introduction.....	1
1.1 Coordinating robots.....	3
1.1.1 Broadcast-and-aggregate.....	3
1.1.2 Periodic data broadcasting.....	4
1.2 Example.....	6
1.3 Control and Coordination.....	7
1.3.1 Behavior-based systems.....	8
1.3.2 Symbolic Systems.....	8
1.3.3 Hybrid Systems.....	9
1.4 Summary of Results.....	11
1.4.1 Broadcast-and-Aggregate Coordination Method.....	11
1.4.2 Structured representations in multi-robot teams.....	12
1.4.3 HIVEMind architecture and implementation.....	13
1.5 Structure of the Report.....	14

Chapter 2 Multi-Robot Coordination through Broadcast-and-Aggregation.....	17
2.1 Broadcast-and-Aggregate Coordination	18
2.1.1 Aggregating Shared Data.....	18
2.1.2 Example of Broadcast-and-Aggregate coordination.....	20
2.2 Data sharing in distributed robot teams	21
2.2.1 A model for data sharing.....	21
2.2.2 Mechanisms for sharing signals.....	24
2.2.3 Failure awareness.....	26
2.2.4 The feasibility of periodic broadcast.....	28
2.2.5 Simulation results for periodic broadcasting	30
2.3 Extensions to the data sharing discussion.....	34
2.3.1 Communication over a large geographic area	35
2.3.2 Sharing large amounts of data.....	37
2.4 Limitations of the communication model	38
2.4.1 Transaction processing.....	40
2.4.2 Reliable communication protocols	41
2.4.3 Sharing state.....	41
2.5 Implementing data sharing on physical robots	42
2.5.1 Event-driven communication using reliable protocols	42
2.5.2 Periodic broadcasting using unreliable protocols	43
Chapter 3 Behavior-based systems	45

3.1 Overview of Behavior-based Systems	45
3.2 Behavior-based Multi-Robot Systems	47
3.3 Limitations of Behavior-based systems	53
Chapter 4 Symbolic reasoning on robots	56
4.1 Model Coherency and Inference Tracking	56
4.1.1 Implications for Multi-Robot Systems.....	59
4.2 Alternatives to Traditional Symbolic Systems	61
4.2.1 Tiered architectures.....	62
4.2.2 Mapping to circuit semantic systems.....	65
4.2.3 Deictic Representation.....	66
4.3 Role-passing.....	67
4.3.1 Compact Storage.....	69
4.3.2 Efficient Inference	70
Chapter 5 HIVEMind.....	73
5.4 Control	73
5.1 Coordination	74
5.1.1 Unique Identification Numbers.....	75
5.1.2 Maximum Team Size.....	77
5.2 Aggregating Shared Data to form Joint Beliefs.....	77
5.2.1 Multi-robot task allocation (MRTA)	80
5.2.2 Clearing slots in the storage vectors	83

5.3 Configuring a HIVEMind team member	83
5.4 Characteristics of HIVEMind	86
5.4.1 Communication costs.....	86
5.4.2 Failure awareness.....	87
5.4.3 Negotiation.....	88
5.4.4 Heterogeneity.....	91
5.5 Handling oscillation.....	91
Chapter 6 The Experimental System and Environment.....	93
6.1 Task and environment.....	93
6.1.1 Town Crier.....	94
6.1.2 Find Static Object	95
6.1.3 Capture Evading Target.....	96
6.2 Hardware.....	99
6.2.1 Mechanical Design.....	100
6.2.2 Sensors	100
6.3 The GRL Programming Language.....	102
6.3.1 Primitive signals.....	102
6.3.2 Compile-time abstractions	103
6.3.3 Behaviors	104
6.3.4 Sequencing.....	106
6.3.5 Pools.....	107

6.3.6 The GRL Compiler	109
6.3.7 Calling programs external to GRL.....	110
Chapter 7 Design of A Robot Using HIVEMind.....	111
7.1 Color Pool.....	112
7.2 Tracker Pool.....	113
7.3 Place Pool.....	116
7.3.1 Map Format.....	116
7.3.2 Probabilistic Localization	119
7.3.3 Landmark-to-Landmark Navigation	122
7.3.4 Navigating around blocked landmarks	124
7.3.5 Searching the Map	127
7.4 Action Pool	129
7.4.1 Town-Crier.....	130
7.4.2 Find-Object	131
7.4.3 Sentry	132
7.4.4 Goto.....	136
7.5 Behaviors	137
7.5.1 Approach.....	138
7.5.2 Turn-to	139
7.5.3 Unwedge	140
7.5.4 Follow-corridor	140

7.6 Command Console.....	143
7.6.1 Command Interface.....	143
7.6.2 Status Display	144
7.7 Communication.....	145
Chapter 8 Tasks implemented using HIVEMind.....	150
8.1 Town Crier	152
8.2 Find Static Object	152
8.2.1 Transcript of a Find Static Object test run.....	153
8.3 Capture Evading Target	156
8.3.1 Transcript of a Capture Evading Target test run.....	156
Chapter 9 Other Multi-Robot Control Strategies.....	163
9.1 Swarm Cooperation	163
9.2 Coordination through observation	165
9.3 Centralized world model.....	167
Chapter 10 Conclusions and Future Work.....	169
10.1 Discussion on coordination.....	169
10.2 Future Work	171
Bibliography	175

Table of Figures

Figure 1.1: One of the Unmanned Ground Vehicles spots an enemy convoy	5
Figure 1.2: Surrounding the enemy convoy as a team.....	5
Figure 2.1: Implementing <i>team-see-blue-object?</i> using broadcast-and-aggregate	20
Figure 2.2: The communication model.....	22
Figure 2.3: Sharing a piecewise continuous signal.....	22
Figure 2.4: Small robot teams, 0.9 message arrival probability	31
Figure 2.5: Small robot teams, 0.5 message arrival probability	31
Figure 2.6: Small robot teams, 0.1 message arrival probability	32
Figure 2.7: Large robot teams, 0.9 message arrival probability	32
Figure 2.8: Large robot teams, 0.5 message arrival probability	33
Figure 2.9: Large robot teams, 0.1 message arrival probability	33
Figure 2.10: Cluster protocol for communication.....	36
Figure 2.11: Observation of events leading to state transitions	39
Figure 3.1: Typical Behavior-based Robot Control System.....	46
Figure 3.2: The Alliance Cooperative Robot Architecture (from [Parker 1998])	48
Figure 3.3: The Pack Controller [Goldberg and Mataric 2000].....	50
Figure 3.4: Propositional explosion in the inference network	54
Figure 4.1: Typical Tiered Architecture	62

Figure 4.2: DIRA architecture and interaction paths [Simmons et al. 2000].	63
Figure 4.3: Unary predicates represented as bit vectors	69
Figure 4.4: Functional outputs represented as numeric vectors.	70
Figure 4.5: Efficient inference using bit-wise logic operations on unary predicates.	71
Figure 5.1: The HIVEMind Architecture in practice.	76
Figure 5.2: Determining which robot is responsible for going to the X position	81
Figure 6.1: The topological map used by the robot team for navigation	94
Figure 6.2: Ochlo and Alektoro shown in side and front profiles respectively	99
Figure 6.3: ProVideo Camera attached to the Nogatech framegrabber	101
Figure 7.1: High-level layout of the experimental system.	111
Figure 7.2: Camera image containing target object and silhouette of pixels in the representative cluster.	115
Figure 7.3: Possible corridor outlets from a landmark node.	117
Figure 7.4: Breadth-first walk using landmark 1 as the root node.	123
Figure 7.5: Robot1 spots the intruder as it goes from A to B.	125
Figure 7.6: The normal path that Robot2 would take to B is blocked by Robot1	125
Figure 7.7: Robot2 takes an alternative path to B in order to trap the intruder	126
Figure 7.8: Visual representation of lookup vector	126
Figure 7.9: Rule 1 – The robots search for the intruder.	133
Figure 7.10: Rule 2 – The intruder has been spotted	134
Figure 7.11: Rules 3&4 – Robot1 halts and taunts the intruder in <i>Sentry</i> mode	134

Figure 7.12: Rules 5&6 – Robot 2 (the <i>Stalker</i>) moves around to trap the intruder.....	135
Figure 7.13: Rule 7 – Both robots move to trap the intruder while taunting.....	136
Figure 7.14: Priority of Behaviors	138
Figure 7.15: Priority of Follow-Corridor Behaviors.....	140
Figure 7.16: Original image and depth map computed from Polly algorithm. Used with permission from [Horswill 1994].....	142
Figure 7.17: Line projections along the left and right walls of the corridor.....	143
Figure 7.18: Map display on the command console	145

Table of Tables

Table 7-1: Typical initial intensity-normalized YUV values	113
Table 7-2: Percepts provided by the tracker pool	114

Chapter 1

Introduction

In this report, I investigate the problem of coordinating multiple robots in a cooperative team activity. Coordinating a group of robots can be a very difficult problem. Getting one robot to carry out a task properly is hard enough; getting multiple robots to perform together in the real world can be like herding cats. Many different approaches have been suggested for team coordination, ranging from swarm techniques to symbolic methods.

I will focus on the particular case of multi-robot teams that perform tasks where only a part of the team can observe changes in the environment, but all robots on the team should be informed of these changes. An example of this is a team that is assigned to capture an intruder in a building; the interior walls separate the robots from each other, but to search and trap the intruder effectively the team needs to work closely together. Coordination for these types of teams will depend on sharing data between teammates in a timely and efficient manner.

I will propose a multi-robot control architecture called HIVEMind (Highly Interconnected Very Efficient Mind) that is based on a simple and effective coordination model:

- Each team member periodically broadcasts all its data to the team.
- Team members base control decisions on shared situational awareness formed by aggregating all broadcast data from the team.

HIVEMind allows near-instantaneous sharing of data between all team members; in essence, the robots are able to function as a sort of ‘group mind’ where all team members share a common situational awareness. This lets the team respond in real-time to contingencies sensed by individual robots without the need for complex negotiation protocols. In addition, HIVEMind robots are able to use role-passing inference to support a limited form of symbolic reasoning over the synchronized world representations. This allows the use of more structured representations than conventional behavior-based systems, which in turn allows the team to be tasked and monitored in natural ways. All this is accomplished while using surprisingly little bandwidth. I will show that given the assumptions:

1. Relevant aspects of the environment change relatively quickly.
2. All team members must be informed of these changes in bounded time.
3. All team members must be able to detect if they are failing to receiving data in a timely manner from their teammates.

the HIVEMind approach to data sharing is optimal in the sense that any other communication scheme must use as many packets as it. The HIVEMind architecture has been implemented on a team of physical robots that perform a variety of tasks involving systematic search of physical space.

1.1 Coordinating robots

1.1.1 Broadcast-and-aggregate

Multi-robot teams often operate in dynamic, unpredictable and often unfriendly environments. In many cases, the robots on the team will not have the luxury of sharing a common physical space, i.e. they may be separated either by distance or physical objects. Each autonomous robot on a cooperating team therefore has its own limited view of the world. One way of coordinating such a team is to have the robots base their control decisions on a unified world model, i.e. the disparate world views from each robot are combined to form a single synchronized representation of the world. This unified world model would be identical across all robots and therefore form a shared situational awareness for the team as a whole. Each team member could then autonomously perform action selection based on this shared situational awareness. I call this approach the *broadcast-and-aggregate* coordination method. Robots using this coordination method share their team-relevant data with teammates and then the shared data is aggregated to form the unified world model on each local robot. The latter step (aggregation) is relatively simple once the data has been shared; finding an efficient and reliable mechanism for sharing data amongst distributed physical robots is the greater challenge.

1.1.2 Periodic data broadcasting

I will argue that a good way of sharing data is by having each robot on the team periodically broadcast all its team-relevant data to the entire team. I will show that, under certain reasonable assumptions, this *periodic data broadcasting* technique is an optimal way of sharing data between team members. Furthermore, while this method appears costly, it is actually quite feasible given the present condition of wireless network bandwidth and multi-robot teams. The available bandwidth on modern commercial RF networks is fairly large in comparison to the amount of shared data on most implemented multi-robot systems (e.g. see [Balch and Arkin 1995], [Parker 1998], [Goldberg and Mataric 2000]). For example, a hundred robots communicating 1000 bits of data would only utilize approximately 1% of the theoretical bandwidth available on a commercial 802.11b wireless network. While the complexity of tasks achievable by multi-robot teams will no doubt increase in the future (resulting in a corresponding increase in the amount of communicated data), the constant factors are still in our favor. That is, an increase in either amount of communicated data or size of the robot team would result in a linear increase in bandwidth usage, but the quantity of available bandwidth is so much larger by comparison that it simply will not matter.

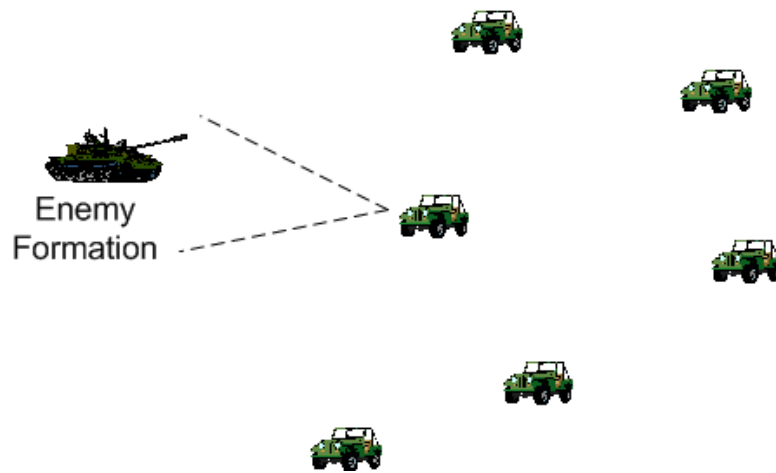


Figure 1.1: One of the Unmanned Ground Vehicles spots an enemy convoy

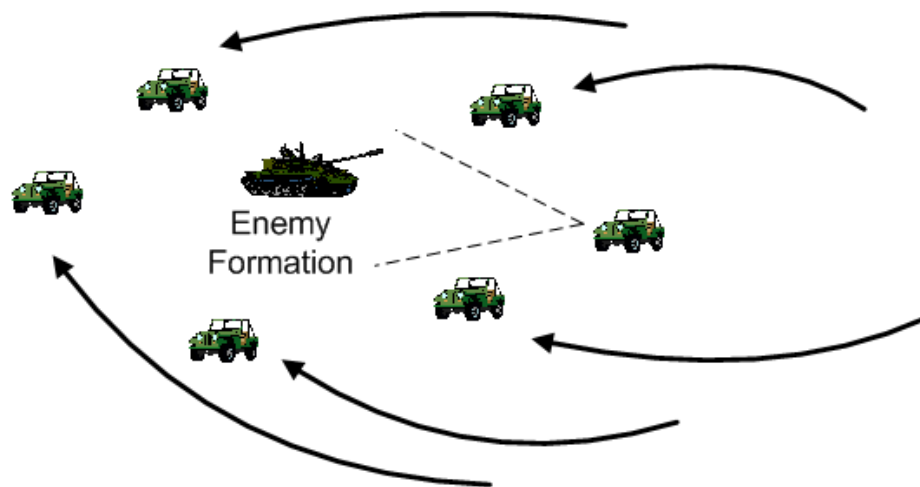


Figure 1.2: Surrounding the enemy convoy as a team

1.2 Example

Figure 1.1 shows a team of Unmanned Ground Vehicles (UGVs) on a search and capture mission. The UGVs must move through their assigned search space in a coordinated fashion. Moreover, the search is *systematic* – each team member attempts to avoid traversing an area that has already been searched by its teammates. If an enemy convoy is encountered, the UGVs must surround the convoy (Figure 1.2). We assume there are intervening objects like buildings that prevent robots from having a clear view of the entire region (otherwise there would be no need to search for the target). Consequently, the team must communicate to know what areas of space have already been searched and whether the target has been found.

We can implement this task using broadcast-and-aggregate as follows. Each robot shares the following information with its teammates:

- Its current location (assume that the UGVs have GPS devices).
- Whether or not it observes an enemy convoy.
- The location of said convoy if it exists.

Although numerous other protocols can be used for communication, the simplest way to achieve this task is to have each robot broadcast its shared data once per second. Assuming coordinates are represented as pairs of 32 bit values, this is only 129 bits of data, which will easily fit in a single network packet. Moreover, a 1Hz update rate should be more than sufficient for this task. A robot transmitting all 129 bits of data per

second would consume less than 0.001% of the available bandwidth on an 802.11b wireless network. So, ten UGVs, transmitting at 1Hz would not even make a noticeable impact on the network. Even without more complicated protocols, this approach could support teams of well over a hundred robots for this task.

Having broadcast its data to the team, each robot then aggregates the data coming from the team to form a unified representation of the situation, e.g. the locations of all team members and the mean estimated position of the convoy. This acts as a synchronized virtual sensor that provides real-time update of the situation. The aggregate model can then be used in conjunction with local sensor data to drive any of a range of control systems.

1.3 Control and Coordination

Not surprisingly, the choice of control systems used on a robot has a significant bearing on its internal representation of data. In general, there are two broad classes of autonomous robot control systems: parallel-reactive (behavior-based) systems and symbolic reasoning systems such as planners or reactive planners. The two types of systems have complementary strengths and researchers have combined them successfully using different techniques.

1.3.1 Behavior-based systems

Behavior-based systems use tight sensor-actuator integration to provide rapid response to changes in the environment. These systems divide sensing, modeling, and control between many parallel task-achieving modules called “behaviors”. Most behavior-based systems are equivalent to feed-forward circuits, i.e. parallel networks of finite-state or zero-state computational elements linked by a series of fixed connections. In essence, these connections are wires connecting circuit nodes in the control system and carry data between those nodes. The downside is that behavior-based systems suffer from very limited representations, rendering most reasoning tasks clumsy and difficult to implement. Most current physical multi-robot systems are behavior-based in nature and hence inherit the problems of this approach.

1.3.2 Symbolic Systems

Symbolic (or deliberative) reasoning systems traditionally operate by performing search algorithms over a set of mutable graph structures in memory. They are able to use far more flexible and expressive representations. However, they are much more difficult to couple to sensors than behavior-based systems since the memory structures must somehow be updated and the search algorithms rerun as sensor data changes. Typically, these memory structures consist of beliefs (assertions in some appropriate logic) stored in a knowledge base, i.e. a database of beliefs. It is generally left to the programmer to add domain-specific rules specifying when to update what aspects of the knowledge

base, which can lead to missed events. Behavior-based systems, on the other hand, typically update all their sensory inputs on every decision cycle, hence ensuring that the control system always uses fresh data.

If ensuring consistency between a single robot's knowledge base and the environment is difficult, then insuring consistency between the knowledge bases of an entire team of robots is even more difficult, since we effectively have a replicated database problem. Some researchers in the multi-agent community have proposed coordination protocols that guarantee hard synchronization of knowledge bases for key items such as team goals (e.g. [Cohen and Levesque 1990]). These coordination protocols enforce synchronization by blocking agents at key points during reasoning until all knowledgebases have been updated with identical joint beliefs, a process that is akin to the commit protocols used in database replication.

1.3.3 Hybrid Systems

Recognizing the complementary strengths of behavior-based and symbolic systems, researchers have attempted to combine the two, usually by layering one or more symbolic systems on top of a behavior-based system. However, tiered systems generally do not resolve or really even address the knowledge base synchronization issues discussed above. Layering these two disparate techniques creates two different representations of beliefs on a single robot. The behavior-based layer retains its circuit-semantic representation, while the symbolic system still relies on a separate

knowledgebase of logical assertions. On a multi-robot team, synchronizing the joint beliefs in all layers of the control system is still a very challenging problem.

An alternative to layering symbolic and behavior-based systems is to find ways of extending behavior-based systems to support more flexible representations while retaining the tight sensory-motor coupling of circuit semantics.

The main weakness of behavior-based systems is that their representations are essentially propositional: they have difficulty expressing predicate/argument structure and other kinds of variable binding. “Deictic” or “indexical-functional” representation [Agre and Chapman 1987] is an attempt to mitigate this problem by using perceptual attention as a form of variable binding. Roughly, the idea is that if a sensory tracker is presently tracking one object, it cannot be tracking any other; hence the object it is tracking is “bound” to the sensory tracker. Role-passing [Horswill 1998] is a form of deictic representation that provides a finite set of indexical names (typically linguistic roles such as *agent*, *patient*, *source*, *destination*, etc) that can be dynamically bound to different objects through sensory attention; when a sensory tracker is assigned to an object in the world, the tracker is tagged with the role to be bound to the object.

The advantage of role passing is that it allows inference rules to be compiled into code that looks like a behavior-based system. The extension of a unary predicate, i.e. the set of objects for which the predicate is true, can be easily represented in role-passing using a bit-vector with one bit per role. We can then compile forward-chaining Horn clause inference into fast, feed-forward Boolean networks [Horswill 1998] whose

inputs are driven by the sensory systems and whose outputs drive the inputs of low-level behaviors. The network changes these outputs continuously as the sensory data changes, allowing behavior-based systems to represent predicate/argument structure while retaining the responsiveness of circuit semantics.

The HIVEMind architecture supports the use of role-passing on world models formed using broadcast-and-aggregate coordination. This allows robots to use flexible representations, real-time inference, and tightly synchronized knowledge bases. Moreover, the compactness of the bit-vector representations used in role passing further reduces communication bandwidth. It is, to our knowledge, the only implemented system for general, automatic knowledge base synchronization of robot teams.

1.4 Summary of Results

This report makes four contributions: the broadcast-and-aggregate method of coordination, a new way of incorporating structured representations into multi-robot teams, the HIVEMind architecture itself and a HIVEMind implementation running on a team of robots.

1.4.1 Broadcast-and-Aggregate Coordination Method

Under the *broadcast-and-aggregate* method, each robot makes its control decisions based on a world representation that is synchronized across all team members. The synchronized world model is formed by aggregating the individual world

representations of each robot. This technique continually maintains a shared situational awareness among the robots and allows team members to react appropriately in real-time to any contingencies. I will argue that it is a feasible approach given the current state of available network bandwidth and multi-robot systems, and furthermore, may be implemented using very simple data sharing methods.

1.4.2 Structured representations in multi-robot teams

Role-passing manages to marry symbolic reasoning and the performance characteristics of conventional behavior-based systems in a unified architecture. It is a form of deictic representation that provides the system designer with a finite set of domain independent indexical variables or *roles*. These roles can be dynamically bound to objects through the sensory system. For example, the role *target* could be first bound to a color blob tracker that is presently following a green target and then later to a pyro-electric sensor detecting the body heat of a target. The inference rules in the robot's control system can access the information provided through this role without having to know the origin of this data. This allows behavior-based robots to use more expressive representations and hence more general inference rules. We took advantage of this in the implementation of our robot team to reuse rules that were common to multiple tasks and to create a limited natural language interface for the human user that tasked the robots.

Another significant advantage of this approach from the context of multi-robot communication is that the representations used in role-passing are extremely compact.

For example, all unary predicates (e.g. *see(X)*) can be represented in just 32 bits. This further reduces the amount of bandwidth usage necessary for team coordination and therefore makes the broadcast-and-aggregate approach more attractive.

1.4.3 HIVEMind architecture and implementation

HIVEMind was designed using both the efficient broadcast-and-aggregate technique and the role-passing architecture. As such, it has a number of advantageous characteristics. The HIVEMind robots:

- are able to efficiently maintain a shared situational awareness; when a team member detects a contingency, other members are immediately informed and respond quickly without the need for negotiation protocols.
- have greater representational power and flexibility in their control systems than conventional behavior-based systems.
- can dynamically subtract or add members of the team at run-time. If a robot has a complete or communication failure, the rest of the team will detect it in bounded time, and carries on without that team member. On the other hand, if a new robot joins the team or an old teammate has its communication restored after some downtime, it will quickly be brought up to speed and be a fully functioning member of the team.
- can be heterogeneous, both in their software control and physical design.

The HIVEMind architecture was implemented on a team of physical robots that performed a set of coordinated systematic search tasks. The robots were controlled by a human commander through the use of a console that had a limited natural language interface and status displays that indicate the present state of the team. The robots were successful in cooperatively performing the complex tasks they were assigned while using very little of the available network bandwidth.

1.5 Structure of the Report

Chapter 2: Multi-Robot Coordination through Broadcast-and-Aggregation. This chapter discusses the broadcast-and-aggregation model of coordination in detail and describes techniques for implementing this approach. It also explores techniques for extending this approach as the number of robots and shared data increases.

Chapter 3: Behavior-based systems. This chapter starts with a brief overview of behavior-based systems. Then, I discuss in detail the virtual wires coordination model inherent in such systems. I also examine some existing multi-robot behavior-based systems in the context of the virtual wires model and describe how this technique can be implemented using broadcast-and-aggregation. Finally, I conclude with a discussion of the shortcomings of behavior-based techniques.

Chapter 4: Symbolic reasoning systems. A brief overview of symbolic systems, their advantages and disadvantages, as well as some techniques for implementing such systems on physically embodied robots.

Chapter 5: The HIVEMind architecture. This chapter begins with a discussion of deictic representation in general and role-passing in particular. Then, it describes the HIVEMind architecture, including a general overview of the approach and the assumptions of the architecture. The key advantages of this architecture, as well as its possible shortcomings, are also discussed.

Chapter 6: The Experimental System and Environment. This chapter describes the robotic team used in the HIVEMind experiments, the environment in which they ran, and the tasks which they performed.

Chapter 7: Design of a robot using HIVEMind. This chapter describes in detail all components of the system implemented on the robots, including the sensory processes, the color blob tracker, the localization system, the inference rules and the HIVEMind implementation.

Chapter 8: Tasks implemented using HIVEMind. This chapter discusses the results of the experiments performed with the HIVEMind robot team described in chapter 6.

Chapter 9: Other Multi-Robot Controllers. A brief discussion of multi-robot control systems that fall under other categories, e.g. swarm techniques.

Chapter 10: Discussion and Future Work. This chapter contains some discussion on aspects of the HIVEMind architecture and some directions for future work.

Chapter 2

Multi-Robot Coordination through Broadcast-and-Aggregation

This report focuses on robot teams that have the following characteristics:

- The robots operate in a dynamic environment where changes occur frequently.
- A given change may only be detectable by a few robots, or even just one.
- All robots nevertheless need to be aware of the changes in order to properly carry out the task.

In this chapter, I will propose a class of coordination mechanisms for such teams known as *broadcast-and-aggregate* mechanisms. These coordination mechanisms generate a unified world model across all robots on the team, enabling them to have a shared situational awareness that can be used to make effective control decisions. I will also explore different protocols for sharing data on physical robot teams and argue the simple method of *periodic data broadcasting* actually suffices for coordinating robots. My goal is to convince the reader that using periodic data broadcasting for communication in a broadcast-and-aggregate coordination method is both a feasible and efficient way of maintaining shared situational awareness between robots.

2.1 Broadcast-and-Aggregate Coordination

Cooperative robot teams often have to face the characteristics of physical space (obstacles or dispersion over a large area) and the limitations of their sensory systems; this results in each team member having their own partial view of the world, i.e. some changes in the environment are observed by only a part of the team.

These disparate world models pose some challenges for coordination, since each team member has a different representation of the world. One way of coordinating the robots is to establish a single, unified world representation across the team. I propose to accomplish this by using the *broadcast-and-aggregate* coordination mechanisms. This approach creates a synchronized world representation on every robot by aggregating the shared data from all team members. Each robot can then autonomously make appropriate control decisions based on that team-synchronized view of the environment.

2.1.1 Aggregating Shared Data

In a team with n members, each aggregate datum j_i is determined by combining the values of shared data $b_{i1}, b_{i2}, \dots, b_{in}$ from robots 1 through n using some aggregation function f_i :

$$j_i = f_i(b_{i1}, b_{i2}, \dots, b_{in})$$

If the shared data b_{ik} are propositional values, then the aggregation function f could be a Boolean operator, such as OR or AND. As an example, an OR operator could be interpreted as the team believing some proposition j_i if and only if at least one member

of the team had evidence for it. In more complicated cases, fuzzy logic or Bayesian inference could be used. Real-valued quantitative data is likely to require task-specific aggregation. Some examples:

- 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.

Ultimately, it is unlikely that there can be a general theory for aggregation functions. As researchers and developers create more multi-robot systems, we will understand how to better combine inputs from different robots and should be able to construct a toolkit containing the most commonly used aggregation techniques for others to reuse.

Since the world representation is synchronized across all team members, the aggregate values have to be known at all robots. This can be accomplished either by sharing the data with all teammates and applying the reductions locally, or sending the data to a central location and then transmitting the aggregate values back to the individual robots.

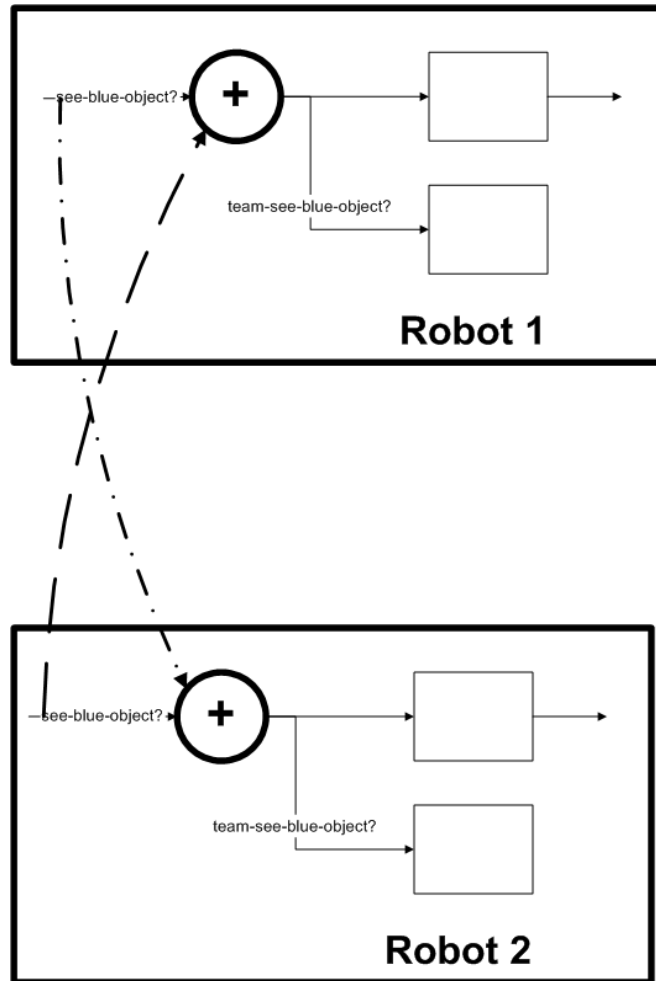


Figure 2.1: Implementing *team-see-blue-object?* using broadcast-and-aggregate

2.1.2 Example of Broadcast-and-Aggregate coordination

Figure 2.1 shows a simple example of two cooperative robots coordinating using the broadcast-and-aggregate method. In this example, a two robot team is looking for some blue colored target object. Both robots have a Boolean perceptual output from their sensory systems, *see-blue-object?*, whose current value is shared with their teammate.

This shared data is aggregated with the local percept value using an OR-operation to generate a new signal: *team-sees-blue-object?*. This new output is then passed to the rest of the control system for processing.

2.2 Data sharing in distributed robot teams

Before the team-relevant data can be aggregated, it has to be shared between teammates. In this report, I will assume that this sharing is achieved through explicit communication between robots. Many different techniques for data sharing have been used, ranging from simple broadcast mechanisms to reliable remote procedure call mechanisms. I will argue here that minimalist approaches based on broadcasting at regular intervals is, in fact, an efficient and simple way of sharing data between team members.

2.2.1 A model for data sharing

Before comparing mechanisms for data sharing, we first need to specify what we mean by “data sharing”. Here the relevant aspects of the task are that:

- Data needs to be shared with all team members
- That data is *time-varying*: team members must be continually updated as the data changes.

We will refer to any time-varying data item as a *signal* and represent it as a function from time to the current value of the signal.

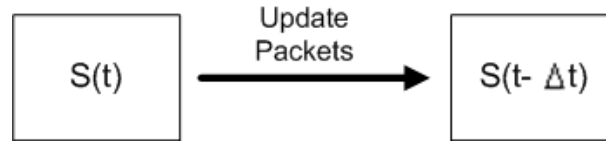


Figure 2.2: The communication model

As said I previously, sharing a signal necessarily involves transmitting its value to teammates (Figure 2.2). At least for the foreseeable future, communication is likely to take place over some sort of packet-switched network such as the IEEE 802.11 standard. Here the simplest case is where the signal is piecewise constant. There will then be discrete events where the signal changes discontinuously and so the robot possessing the signal will be required to transmit updates to its teammates (Figure 2.3). However, transmission takes time and those update packets can be lost, resulting in delay or even complete breakdown of sharing.

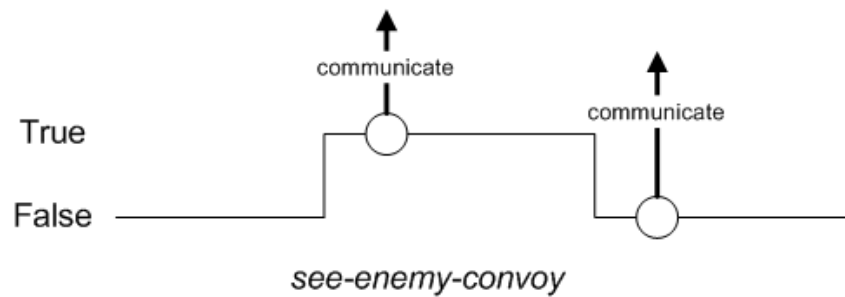


Figure 2.3: Sharing a piecewise continuous signal

Because of these complications, I will adopt a “PAC” definition of sharing: a robot will be said to successfully share its signal with a group if group members always have a *probably almost current* value for that signal:

Definition 1: A team R shares a signal S with lag ε and reliability p if with probability at least p , for all time points t and team members $r \in R$:

$$\exists \Delta t < \varepsilon \text{ s.t. } r \text{ knows } S(t-\Delta t) \text{ at time } t$$

Here the lag parameter ε is a measure of the degree of synchronization of the team and the probability parameter p is a measure of the reliability of that synchronization. It is important to remember that synchronization guarantees must always be probabilistic because of the unreliability of communication. They must also involve non-zero lag because of the finite bandwidth and latency of real communication channels as well as of packet loss.

This definition gives us an immediate lower bound on communication for sharing in the absence of prediction:

Proposition 1: Let R be a set of robots, each of which shares an unpredictable signal S_r , $r \in R$ with the group. If $c > \varepsilon$ is the mean period between changes for a given signal, then on average the robots must exchange at least $o(|R|/c)$ packets per unit time to share the signals.

Proof: Consider two arbitrary robots r and r' . Since S_r is unpredictable, then in order for the group to share signal S_r , r' must always receive an update from r within ε time units of a change in S_r . If $c > \varepsilon$, then on average, r must transmit at least $1/c$ packets per unit time, simply to keep r' synchronized with S_r . Since

there are $|R|$ robots and signals, the minimum average number of packets then required to keep r' in sync with all signals is $\frac{|R|-1}{c}$, or $o(|R|/c)$.

In principle, this bound can be overcome in the special case where the signals can be accurately predicted. In this case, all receivers can predict the value of the signal from past values and the sender (who can also perform the prediction) need only transmit updates when the prediction is incorrect. However, even in this case, prediction is only likely to save a constant factor, and so the asymptotic bound still applies.

2.2.2 Mechanisms for sharing signals

There is a very broad space of protocols that one can use for sharing data. One class of such protocols is *event-driven communication*, where communication is used only when necessary, e.g. when the values of the shared signal changes.

Another class of protocols takes a simpler approach: the value of the shared signal is broadcast to the entire team at regular intervals. I call this approach *periodic data broadcasting*.

Proposition 2: If $c > \varepsilon$ is the mean period between changes for signals $S_r, r \in R$, then each robot r periodically broadcasting the value of S_r at intervals of ε suffices for a team R to share the signals.

Proof: Since the transmitter broadcasts packets every ε time units, it is guaranteed to broadcast a packet within ε time units of a given change.

Therefore, if p is the probability of packet loss, then every receiver is guaranteed to have lag less than ϵ with probability $> p$.

This approach is equivalent to sampling a signal S_r at some given rate, which is appropriate given our model of shared data as a set of signals. To reconstruct the underlying signal at the receiving end, the sampling has to be performed at a fast enough rate, i.e. broadcast to the team at appropriate intervals. In this case, the lag parameter ϵ determines the periodicity of sampling or periodicity of broadcasts in order to share S .

Proposition 3: The periodic data broadcasting technique for sharing a signal is optimal up to a factor of $\frac{c}{\epsilon}$ in the number of packets transmitted per unit time.

Proof: A robot r needs a minimum of $1/c$ packets per unit time to share a signal S_r with robot r' . The periodic data broadcasting technique generates $1/\epsilon$ packets per unit time. Therefore, this technique is $\frac{1/\epsilon}{1/c}$ or $\frac{c}{\epsilon}$ times the optimal amount of packets necessary to share a signal.

Furthermore, if robots on a team have signals S_1, S_2, \dots, S_n , and their respective mean periods between changes $c_1, c_2, \dots, c_n > \epsilon$, then each robot can share the signals by periodically broadcasting a single packet with all the current values of S_1, S_2, \dots, S_n . This improves efficiency by a significant constant factor, since there is no longer any need to send individual packets for each signal.

2.2.3 Failure awareness

Again, packet loss requires that any sharing guarantees be only probabilistic. Sharing breakdown is inevitable. However, different protocols vary in the way they cope with breakdown.

Definition 2: Suppose a group of robots share a signal S that is transmitted by robot r . A robot r' is *Receive Failure Aware* (RFA) if it can determine whether its current lag for S is greater than ε . Robot r is *Transmit Failure Aware* (TFA) if it can determine whether another robot's current lag for S is greater than ε .

It is useful to have a robot team that is RFA and TFA when team members are sharing data continuously. After all, the first step to solving a problem is to be aware that it exists. Both RFA and TFA can be easily implemented in the context of the periodic data broadcast technique discussed in the previous section.

RFA can be easily implemented by having the robot maintain a *last-heard-from* timestamp for each of its teammates. Every time a packet is received from a teammate, this timestamp is updated to the current time. If no new packets arrive from one or more team members, the robot can notice that the timestamp(s) are now stale and take appropriate action. Note that RFA cannot be easily implemented by a purely event-driven communication model. Robots have no way of distinguishing between no events happening (which would generate no new packets) and a communication failure (which would prevent sent packets from getting through).

TFA can be implemented by attaching the *last-heard-from* timestamp value for all teammates in outgoing packets. This essentially acts as an acknowledgement message for senders. A robot can check the *last-heard-from* timestamp that its teammate is maintaining for it; if that value is not being updated, then that teammate is failing to receive the robot's transmissions.

Proposition 4: Any communication scheme that is RFA has to send at least as many packets as the periodic data broadcasting technique.

Proof: Two robots r and r' are sharing data using some protocol is that RFA, but transmits less packets than the periodic data broadcasting technique. Consider the situation where r has no new data to share and the situation where r transmits a packet that is lost; in both cases, the input to r' is exactly the same, i.e. r' receives no new transmissions from r . The deterministic failure detector on r' has to map this input to one of two possible outputs: there was or was not a packet reception failure. Either way, one of the situations will be mapped to the wrong output. Therefore, the input to the failure detector has to be changed for one of the two situations. The latter situation (the packet fails to arrive) cannot be changed, so we should modify the former situation instead: r' should now expect to receive a packet from r at least every ε time, whether or not new data exists. This way, there are two different inputs to the failure detector (a packet has or has not arrived) that can be clearly mapped to two different outputs (there has or has not been a reception failure). Note that this is identical to the

communication scheme specified by the periodic broadcasting technique, therefore periodic data broadcasting sends the minimum number of packets in order for a scheme to be RFA.

2.2.4 The feasibility of periodic broadcast

While periodic broadcast appears to be an effective way of sharing data, the approach raises obvious questions about the cost of communication. Specifically, would periodically sending all shared data overwhelm the bandwidth available for communication? As the discussion below shows, the answer is a resounding “No”.

Most implemented multi-robot systems only share ten kilobits of data or even less per robot. Some typical examples:

- [Balch and Arkin 1995] present a team of robots that perform foraging, consuming and grazing tasks. The robots could communicate in two active modes: *state* communication, where one bit of data indicated whether a robot was in a particular state and *goal* communication, where two bytes represented the location of a target.
- A robot using the Alliance architecture [Parker 1998] periodically broadcast a statement of their current action; its teammates could use this information to determine their own courses of action based on concepts of impatience and acquiescence. It is possible to represent up to 256 different actions in a single byte of data.

- [Goldberg and Mataric 2000] describe a pack of robots that cooperatively foraged for pucks. If a robot with higher precedence is delivering a puck, the others are supposed to wait until it completes before they proceed. A robot can inform its teammates that it is delivering a puck by transmitting its priority value, which can be represented in a few bits.

Of course, there are exceptions to this rule. Collaborative map-building robots that generate occupancy grid maps, for example, have to communicate their current version of the map to teammates; this can be fairly expensive, depending on the granularity and total size of the map. Specialized bulk protocols can be used for high bandwidth applications like collaborative mapping or transmitting a video feed to an Operator Control Unit (OCU).

Ignoring any overhead costs, 1000 robots transmitting 1000 bits of data at one hertz would use 1Mbs of network bandwidth. In comparison, the IEEE 802.11b specification for wireless networks provides up to 11Mbps of bandwidth¹. Therefore, the robots would only use approximately 10% of the available bandwidth. Even assuming available network bandwidth never increases in the future (an unlikely prospect), the total amount of shared data for the team could theoretically increase

¹ Of course, the full theoretical bandwidth of the network is generally not achievable in practice; however, the bandwidth achievable in practice still far outstrips what is needed for most existing multiple robot teams.

fivefold and there would still be sufficient bandwidth available. I believe it is safe to say that for the foreseeable future, the amount of available network bandwidth will far outstrip the amount of shared data for most implemented multi-robot systems.

2.2.5 Simulation results for periodic broadcasting

I ran simulations of a robot communicating a significant event to its teammates. The probability of a message arriving intact at a given destination/robot was varied between 0.1, 0.5 and 0.9, representing different network conditions. Each simulation was run 1000 times for different team sizes to determine the mean and standard deviation of the number of broadcasts needed for all team members to be informed of the event. The results are shown in Figures 2.4 through 2.9.

Figures 2.4 and 2.7 show that, when network conditions are good (90% of messages reach their intended recipients), the number of transmissions needed to update the entire team does not go up by much as the team size increases. On the other hand, when conditions are bad, as in Figures 2.6 and 2.9, then data takes a very long time to propagate through the team. However, it should be noted that this is a problem faced by all coordination protocols with explicit communication, i.e. noisy or congested mediums will be a difficulty for any team that uses communication to coordinate their activities.

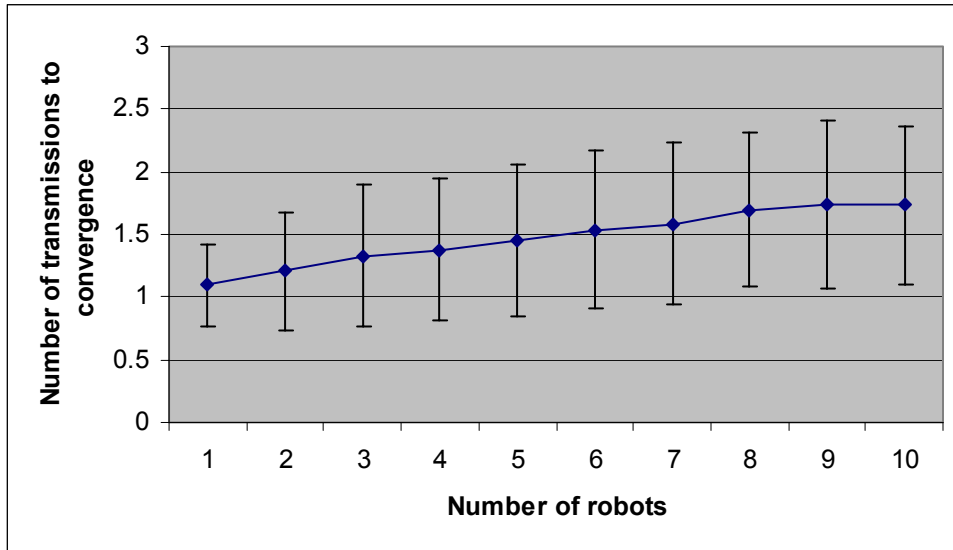


Figure 2.4: Small robot teams, 0.9 message arrival probability

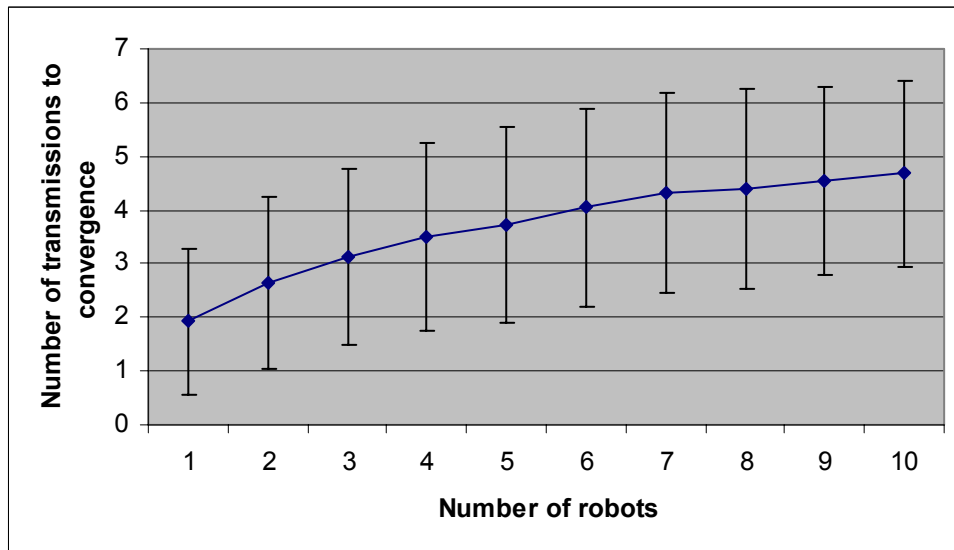


Figure 2.5: Small robot teams, 0.5 message arrival probability

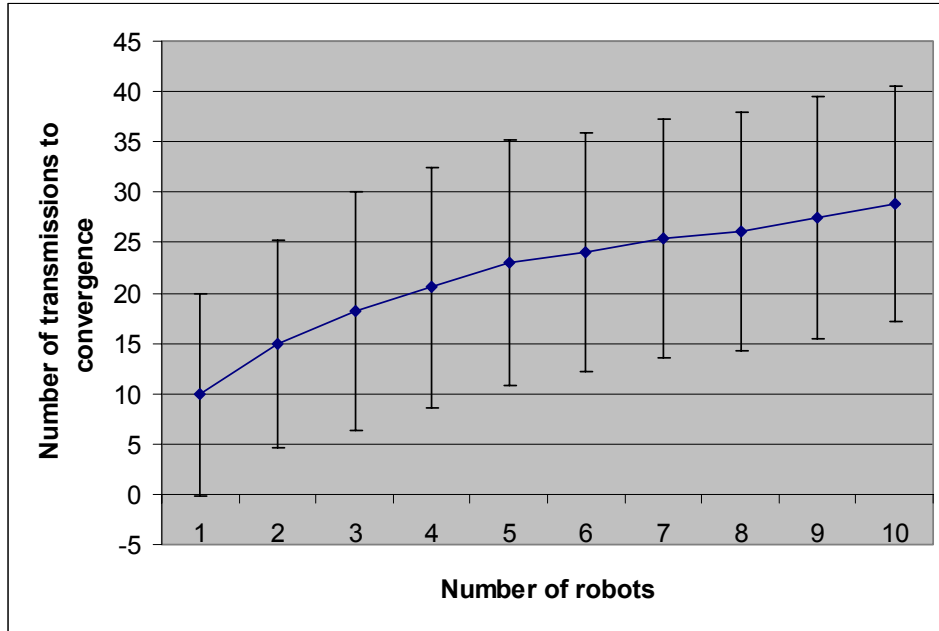


Figure 2.6: Small robot teams, 0.1 message arrival probability

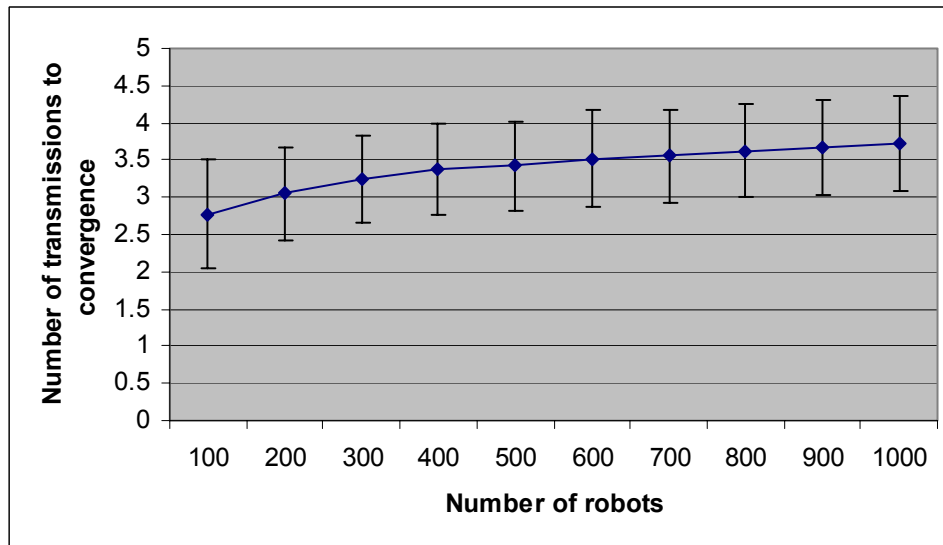


Figure 2.7: Large robot teams, 0.9 message arrival probability

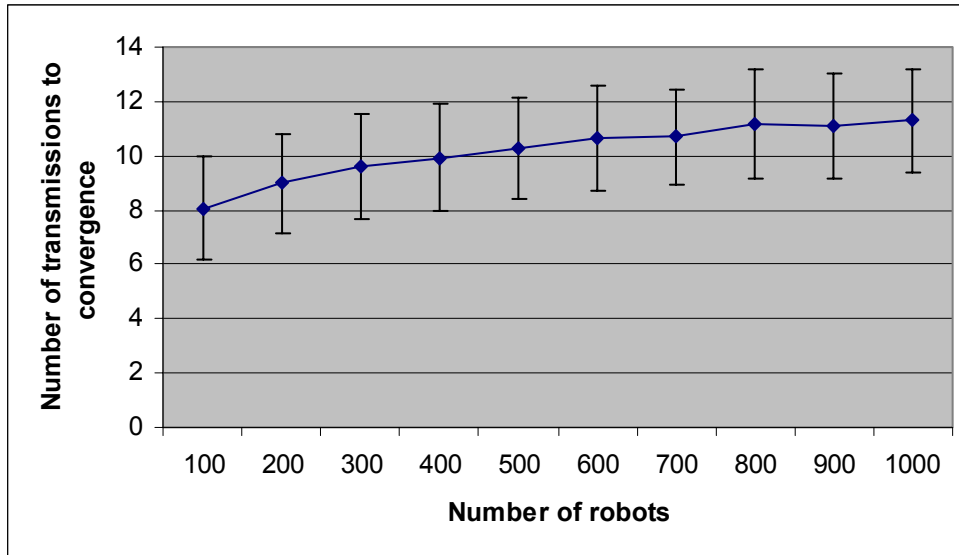


Figure 2.8: Large robot teams, 0.5 message arrival probability

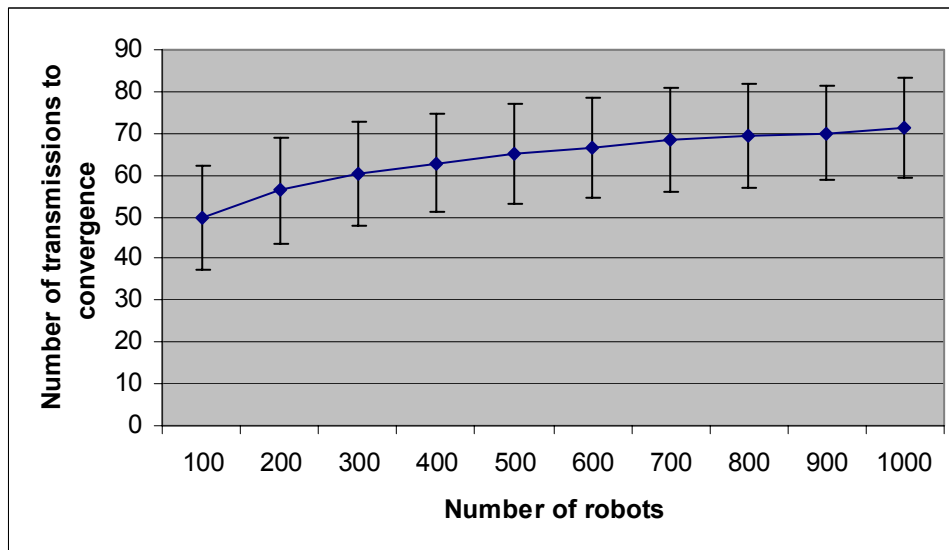


Figure 2.9: Large robot teams, 0.1 message arrival probability

The message delivery model used in this analysis assumes that messages are independent and have identical distributions. However, researchers have shown that loss characteristics for most wireless networks are not independent across packets (e.g. see [Eckhardt and Steenkiste 1996]), i.e. a lost message is generally followed by the loss of several subsequent transmissions. A more thorough analysis of data propagation within a team should use take into account these error models.

2.3 Extensions to the data sharing discussion

The discussion on data sharing has implicitly assumed two points:

- All robots are within transmission range of each other
- All shared data can be fit into a single communication packet

Obviously, these two assumptions will not always be true. As the number of robots on the team increase beyond a certain size, it might simply not be possible or even desirable to fit all robots on the team into a constricted physical space. The Lucent Orinoco Silver cards used by the robots in my experiments have an effective communication distance from around 160m at 11Mbps up to about 550m at 1Mbps in an open environment. The robots might well be spread out over an area larger than this, so some team members could be positioned beyond the maximum effective communication distance from each other. Furthermore, as the tasks performed by robots become more complex, the amount of shared data will also grow correspondingly. The broadcast-oriented UDP protocol currently used by the robots has a maximum data size

of 1024 bytes. While presently sufficient, it is possible to imagine tasks where the robots have to share more data than this limit.

2.3.1 Communication over a large geographic area

The Mobile Ad Hoc Networking (MANET) community is concerned with collections of mobile nodes that are dynamically located and where the interconnections between the nodes are capable of changing at a moment's notice. MANET researchers have created a number of routing protocols designed to establish communication routes between two nodes so that messages can be delivered in a timely manner [Royer and Toh 1999]. These routing protocols could be used to connect robots that are spread out over a large area.

Clustering protocols are one example of such a routing scheme. These protocols group nodes (or, in our case, robots) into clusters based on proximity, and perform hierarchical routing between these clusters. Many clustering protocols establish a leader for each cluster and utilize gateway nodes for inter-cluster communication ([Baker and Ephremides 1981], [Basagni 1999], [Chiang et al. 1997]). Other clustering protocols take a more distributed approach to cluster management (e.g. see [Lin and Gerla 1997]). [Streenstrup 2000] provides an overview of many different clustering protocols. In general, the robot team designer can select an appropriate routing protocol based on the needs of the application.

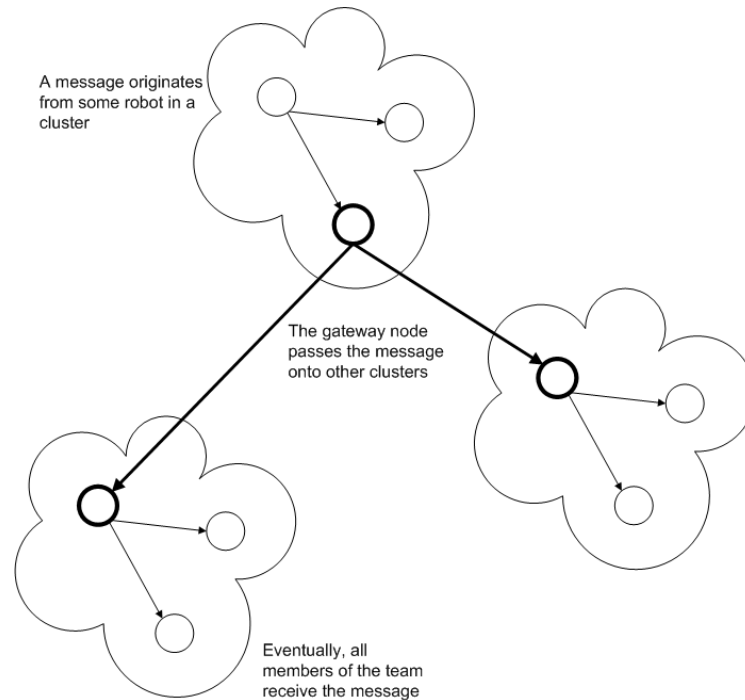


Figure 2.10: Cluster protocol for communication

The clustering protocol partitions the robot team into a reduction tree or prefix scan communication structure. In most cases, the aggregation functions used to generate the synchronized world representation are associative and commutative operations, e.g. the bitwise OR and AND functions used to combine propositional values, MIN, MAX, MEAN, etc. This allows the natural use of prefix scans as a communication structure for sharing data across a dispersed robot team. Sometimes, the aggregation function will not be an associative reduction operator, e.g. when the aggregation function is a voting mechanism. However, the prefix scan approach can still be used if the output of the aggregation function were changed appropriately. For example, instead of returning the top vote-getter, the aggregation function returns a vector containing the number of votes

for each option. This effectively changes the aggregation function into a vector summation operation, which is associative and commutative, and can be easily implemented using prefix scans.

The use of reduction trees slows down the rate at which the robots are synchronized. Within a cluster, assuming no communication failures, robots are synchronized within one communication cycle. It would require $o(\log n)$ time for all clusters to be synchronized, where n is the number of robots in the team.

2.3.2 Sharing large amounts of data

There are several different options for getting around the problem of data being too large to fit in a single communication packet. One simple approach is to split the shared data into multiple packets; each packet has an associated source robot ID and sequence number. This approach is implemented in the HIVEMind architecture discussed in Chapter 5, although it is a feature that has not been needed for the tasks implemented so far.

Another possibility is to simply let the network layer handle the problem through IP fragmentation or IP segmentation. The Internet Protocol (IP) has a Maximum Transfer Unit (MTU), which is the largest size of datagram that can be transferred using a data link connection. IP packets that are larger than the MTU size are cut up into smaller fragments by the network layer. The final receiver can then reassemble the parts using information in the IP packet header. Of course, the larger the

original packet, the greater the likelihood that the packet will reach its destination corrupted. This could necessitate the data being resent or result in information loss.

A more sophisticated approach is the use of sparse vectors. Rather than unreliable broadcast, this technique calls for multiple point-to-point connections with acknowledgement messages. The sender maintains a list of acknowledgements from receivers indicating what data they received and when it was received. Then, in the next communiqué, the sender only transmits the data that has changed since the acknowledgement from the receiver. This approach is effective when there is a lot of shared data, but most of it does not change very often.

2.4 Limitations of the communication model

The communication model described in Section 2.2.1 guarantees convergence for shared signals, i.e. if the system is allowed to run to quiescence (the signal is no longer changing), then all parties have the same value for the signal.

This convergence property, while convenient and useful, can lead to problems for some applications. This approach is analogous to timestamp locking in distributed databases [Gray and Reuter 1993] which guarantees convergence and identical end states on the different nodes of the system (in our case, data on robots) in a simple and efficient manner. Applications that only require convergence, such as Lotus Notes, the Internet name service and many email systems, use timestamp locking techniques to achieve synchronization.

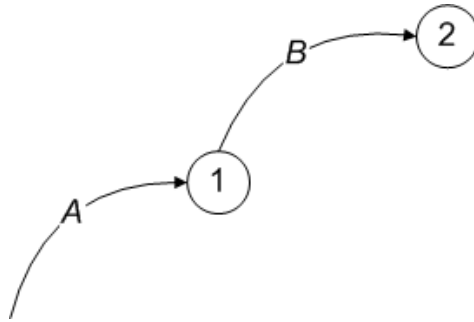


Figure 2.11: Observation of events leading to state transitions

However, since only the latest received values are stored, timestamp locking does not guarantee that transmitted data will be stored in any particular order or that they will even be stored at all. If a message arrives later than one that was sent after it, then that message is simply discarded. This phenomenon is known as the *lost update problem* and may result in problems if the application depends on communicated data being applied in some specific order. One example of the lost update problem is the following: Suppose that the robots share the values of $see(eventA)$ and $see(eventB)$. Event A should occur before Event B , e.g. Event A could be “see enemy at point A ” while Event B could be “see enemy at point B ”. Upon the observation of Event A by a member of the team, all robots should make a transition into State1. Then, when Event B is observed, the robots will transition into State2. Since the underlying protocol is unreliable, one of the following situations could occur:

- The observation of Events A and B could be sent and received in the proper order.

- Event A could be received properly, but Event B is dropped for at least some of the robots, leaving some of the robots trapped in State1.
- Event B is received before Event A , leading the latter to be discarded (since Event B was actually sent after Event A). The robots never even transition into State1.

2.4.1 Transaction processing

A stronger alternative model for data sharing is transaction processing protocols. Regular databases use an ACID (Atomicity Concurrency Isolation Durability) transaction model which guarantees that:

- Any updates (i.e. transmitted data) are stored at every node (robot) or not at all.
- Updates are stored in exactly the same order at every node.

Each update is treated as a transaction that is stored using a Two Phase Lock and Commit [Bernstein, Hadzilacos and Goodman 1987]. This is the strongest solution to the lost update problem; essentially, using this approach would be tantamount to treating each robot as a database and performing data replication across them. Unsurprisingly, this approach has much higher overhead costs than timestamp locking. Furthermore, [Gray et al. 1996] have illustrated numerous problems associated with database replication; eager replication techniques can lead to an abundance of deadlocks while lazy replication can lead to inconsistencies between databases.

2.4.2 Reliable communication protocols

An alternative would be to share data reliably from a networking point of view, hence ensuring all packets arrive and are in the right order. In this case, communication could be achieved through the use of reliable protocols such as the Transmission Control Protocol (TCP) or Scalable Reliable Multicast (SRM) [Floyd et al. 1997]. These protocols will account for dropped packets and ensure that messages from a particular source always arrive in order. However, it is still possible to lose messages in temporary communication blackouts that can cause the protocol to timeout and give up. Furthermore, unlike transaction processing, TCP will not guarantee that an update happens at all robots or not at all, i.e. there is no rollback functionality. Suppose communication fails temporarily on one or more team members and the rest of the team continues transmitting and receiving packets. When communication is restored, a subset of the team is now out of synch with the remainder for whom there was no break in communication. Under transaction processing, updates would not have been applied unless all team members received them; reliable transmission protocols, however, generally have no such provision.

2.4.3 Sharing state

Perhaps the solution lies in the form of the data that is being communicated. The data being sent in the example above represents fleeting events; hence, the propositions that represent them are true only for a short time. If the packet that carries the TRUE

instance of the proposition is dropped or delayed, then at least some robots on the team will never know that the events have occurred. So, instead of transmitting *state transitions*, a better technique would be to transmit the state *itself*. That is, rather than sending the beliefs “see enemy at point *A*” and “see enemy at point *B*”, the robots should communicate “seen enemy at point *A*” and “seen enemy at point *A* and *B*”. This approach is tolerant of dropped or delayed messages since in many cases only the most recent robot state matters; since the robots are continually rebroadcasting the current signal values, the proper state will eventually reach all team members even if some packets are dropped. Transmitting their current state rather than observed events will allow the robots to exploit unreliable protocols that use less network bandwidth and hence be more efficient.

2.5 Implementing data sharing on physical robots

In this section, I will discuss using specific protocols to implement the periodic data broadcasting technique. The focus again is to use methods that use as little bandwidth as possible while being able to share data among the robots in a timely manner.

2.5.1 Event-driven communication using reliable protocols

Many current physical robot teams use an event-driven communication mechanism over TCP links to share data. This is actually a fairly expensive way of sharing data. First, TCP has considerable overhead costs, especially in relation to the (generally) small

amounts of data being shared. Second, TCP is a point-to-point protocol, so each robot must maintain a separate TCP connection for each teammate on the team.

Proposition 6: Let R be a set of robots, each of which shares an unpredictable signal S_r , $r \in R$ with the group. If $c > \varepsilon$ is the mean period between changes for a given signal and the robots share data using the standard TCP protocol, then the robots must exchange at least $o(|R|^2/c)$ packets per unit time to share the signals.

Proof: A robot r has to maintain $|R|-1$ TCP connections for communication, i.e. each time S_r changes, it has to send at least $|R|-1$ packets (one for each teammate). Since $c > \varepsilon$, then on average, r must transmit at least $\frac{|R|-1}{c}$ packets per unit time to keep the rest of the team synchronized with S_r . Since there are $|R|$ robots and signals, the minimum average number of packets per unit time is therefore $\frac{|R|^2 - |R|}{c}$, or $o(|R|^2/c)$.

This is considerably more costly than the lower bound of $o(|R|/c)$ for sharing signals (by Proposition 1).

2.5.2 Periodic broadcasting using unreliable protocols

An alternative approach would be to use an unreliable broadcast protocol, e.g. non-blocking broadcast-oriented User Datagram Packets (UDP), to send the messages to teammates. The primary advantage of such protocols is that they require less overhead

than reliable protocols. The disadvantage, of course, is that packets could arrive out of order, be duplicated or even be completely dropped.

A partial remedy is to attach a timestamp to each outgoing message. When a packet is received by a robot, the timestamp is checked against the last timestamp stored for the source of that packet. If the current packet has a later timestamp, its contents replace the old values; otherwise the packet is simply discarded. Dropped packets are simply ignored and not resent. This suffices to implement data sharing since the definition of data sharing only requires team members to have accurate current data, so older values may simply be overridden or even dropped/ignored.

In contrast to the expensive event-driven model discussed in the previous section, an implementation of the periodic data broadcasting technique using an unreliable protocol like non-blocking broadcast-oriented UDP is $\frac{c}{\epsilon}$ -optimal in the number of packets per unit time (by Proposition 3). In fact, this approach is optimal among RFA systems (by Proposition 4). In addition, the overhead cost of a protocol like UDP is generally much lower than TCP, resulting in large constant factor savings in bandwidth per message sent. So, this approach is, in fact, a much more efficient data sharing mechanism. Finally, as I pointed out in the previous section, the main weakness in this approach (i.e. dropped packets) can be alleviated in most applications by sharing state rather than observation of specific events.

Chapter 3

Behavior-based systems

3.1 Overview of Behavior-based Systems

The control systems of behavior-based robots [Arkin 1998] consist of a set of behaviors that are tightly coupled to the sensory and actuator components of the robot. Some behavior-based systems are biologically inspired (e.g. [Arkin 1989b], [Beer, Chiel and Sterling 1990]), while others are built using a bottom-up, experimental approach (e.g. [Connell 1989], [Ferrell 1994]).

The term “behavior” is somewhat vaguely defined in the robotics literature. For example, a behavior can be a simple feedback loop that measures the difference between its goal and the current state of the world, and produces an output value that attempts to compensate for this difference. A behavior can also be a stimulus-response pairing or a finite-state machine. In addition, complicated combinations of simpler behaviors are sometimes called behaviors as well. For the current discussion, I take a behavior to mean an identifiable component or building block in the control system of the robot that directly implements some sort of sensory-motor loop.

All behaviors in the control system are ostensibly executed in parallel, and some arbitration mechanism is responsible for coordinating the output of these behaviors, as shown in Figure 3.1. There exists a large body of robotics literature describing

architectures for behavior-based robotics; by and large, each of these architectures has developed its own flavor of arbitration mechanism.

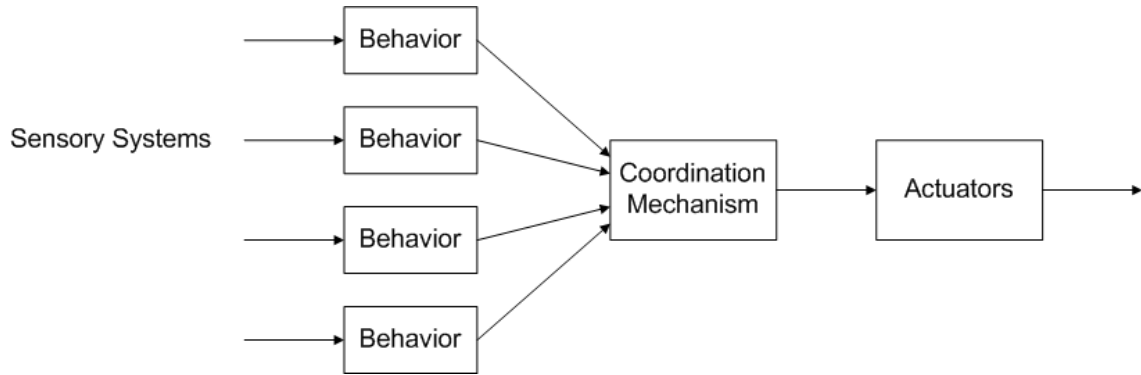


Figure 3.1: Typical Behavior-based Robot Control System

The subsumption architecture [Brooks 1986] is probably the best-known behavior-based robotic architecture. Arbitration is achieved in a competitive manner. The control behaviors are arranged in layers of ascending priority. Layers with higher priority are said to *subsume* the operation of lower layers by either overwriting their input or output data streams. The layers of control are developed incrementally; the robot programmer creates and debugs each layer before moving on to the next. In this way, the robot gains new skills in a sort of evolutionary approach.

Arkin's Motor Schemas [Arkin 1989a] uses a vector-based approach to behavior coordination. Each behavior generates a vector denoting the direction and speed it would like the robot to travel. The vectors are summed together, resulting in a composite output vector for the entire group. This approach is related to work in

potential fields (see [Khatib 1985] and [Krogh 1984]), and a detailed presentation of such techniques appears in [Latombe 1991].

The Distributed Architecture for Mobile Navigation (DAMN) [Rosenblatt 1995] uses a voting approach to arbitration; each behavior has a certain number of votes available for allocation to a number of available actions. The action with the most number of votes is chosen.

Most behavior-based systems are equivalent to parallel networks of finite-state or zero-state components that communicate over a set of fixed connections, i.e. circuit semantic systems [Nilsson 1994]. In general, the components or computational nodes of the system receive input from their incoming connections and recompute their outputs on each decision cycle, allowing the control system to track changes in the physical world as sensory data changes. The connections between the nodes carry data between them and the interpretation of this data is dependent on the node the connection is linked to.

3.2 Behavior-based Multi-Robot Systems

A number of multi-robot systems based on the behavior-based approach have been implemented. In this section, I will describe some typical behavior-based multi-robot systems and briefly discuss their coordination strategies.

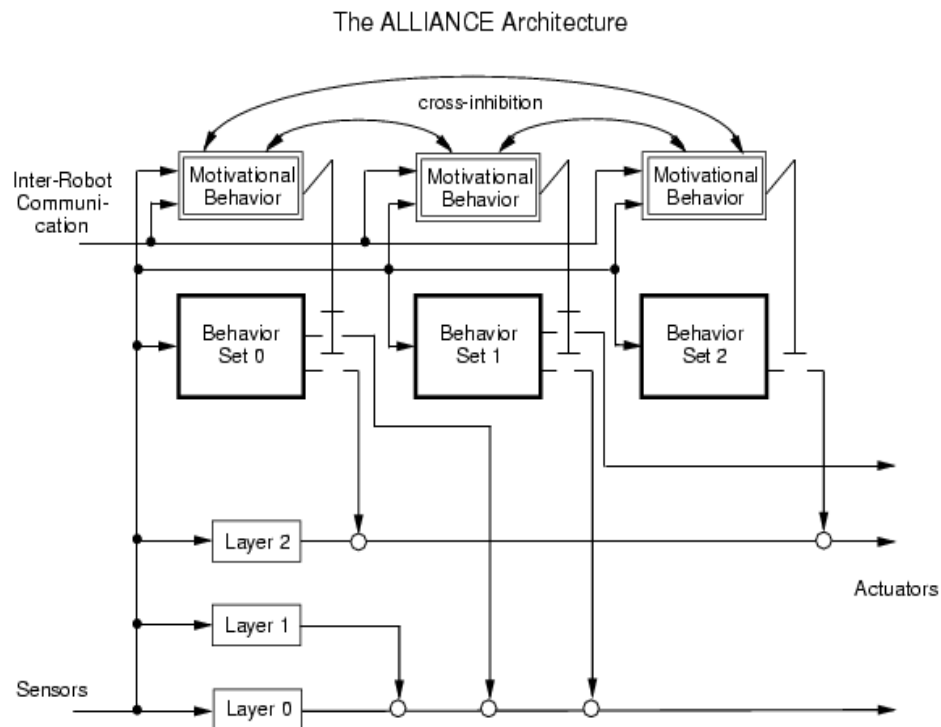


Figure 3.2: The Alliance Cooperative Robot Architecture (from [Parker 1998])

The Alliance cooperative robot architecture described in [Parker 1998] is a direct descendant of the subsumption approach described in the previous section. Alliance is specifically designed for robot teams that perform loosely coupled tasks. Individual robots are guided by subsumption-like control programs using both local sensory inputs and communicated information from its teammates. Team coordination is based on concepts of *impatience* and *acquiescence*. Each robot periodically broadcasts the task it is currently attempting to perform. This communication feeds directly into its teammates' *motivational behaviors* (see Figure 3.2) and lets them know how long it has attempted to perform a particular task. If a robot does not complete a task in a timely

fashion, its teammates could become impatient and try to take over that responsibility. Conversely, a robot that is failing to perform a task could step aside and allow one of its teammates to take over. Robots using the Alliance architecture have successfully performed such tasks as formation maintenance, simulated toxic waste clean-up [Parker 1994a], box pushing [Parker 1994b] and cooperative observation of multiple targets [Parker 1997]. A variation of the Alliance architecture, called L-Alliance [Parker 1996], incorporates a learning mechanism through the use of parameter tuning. L-Alliance lets members of a team to learn the efficacy of each robot in performing a task, hence allowing the team to improve its performance over time as robots find out which team member is best at executing particular tasks.

The Societal Agent Theory [MacKenzie 1997], inspired by Minsky's Society of Mind [Minsky 1986], establishes a single representational syntax for expressing both sensorimotor behaviors and teams of physical agents. That is, the approach makes no distinction between the construction of intra- or inter-agent behaviors. A team of robots may, in effect, be viewed as an assemblage itself. The ideas from this approach were implemented on a multi-robot design system called MissionLab [MacKenzie, Cameron and Arkin 1995]. Multi-robot coordination under this approach can be either state-based (assigning responsibilities to team members through state transitions) or continuous (i.e. combining real-valued functional output from multiple robots). The underlying robot control architecture, as well as the multi-robot coordination mechanism, is behavior-based.

Market-based architectures have been successfully used to coordinate a number of robot teams. Under this approach, each robot bids on tasks that are made available by a scheduler. The goal of each robot is to maximize its own profit by minimizing its individual costs. [Goldberg et al. 2003] describe a market-based architecture that has been applied to a Mars exploration scenario where the robots search for interesting rocks. MURDOCH [Gerkey and Mataric 2000] [Gerkey and Mataric 2002] is a variant of the contract net protocol [Davis and Smith 1983] built on a publish/subscribe communication model. This architecture has been implemented on a box pushing task as well as a loosely coupled multi-robot experiment where the robots bid on a series of tasks that arrive asynchronously over a period of time.

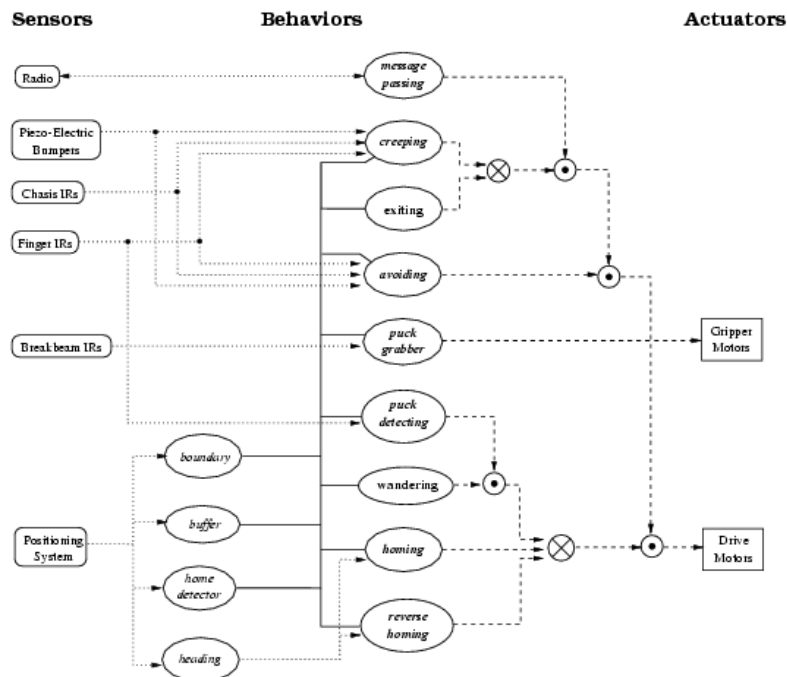


Figure 3.3: The Pack Controller [Goldberg and Mataric 2000]

A lot of work in behavior-based multi-robot systems has focused on foraging or collection tasks. The foraging task generally involves having the robots collect specific objects in the environment and returning them to a home base. [Goldberg and Mataric 2000] presents a series of experiments utilizing three different multi-robot controllers on robots performing collection tasks. Two of the controllers do not explicitly use communication; the last one (the Pack controller) uses the radio as an extra “sensor”. Figure 3.3 shows the reasoning circuit layout for the pack controller. The “wire” from the radio sensor carries data about the status of other robots. If a robot with higher precedence is delivering a puck, this input acts as an inhibitor to the rest of the robots, allowing only one robot at a time to deliver a collected puck to the home base. In [Balch and Arkin 1995], the authors evaluate the performance of robot teams that utilize three different communication modes: no communication, state transmission and goal communication. The robots are designed to perform foraging, consuming and grazing tasks, and are controlled using schema-based reactive control systems. They are able to show that no communication whatsoever is required for the successful completion of some tasks and it is possible to perform all the tasks examined while sharing very little data between robots. [Fontan and Mataric 1997] investigate a cooperative strategy where the environment is divided into specific territories for each robot on the team. The robots then hand off collected objects from area to area, with the last robot delivering the objects to the home base. This approach allows foraging without the need for explicit communication between the robots. [Balch 1999] investigates *multi-*

foraging, which requires robots to collect different objects and deliver them to different locations according to type. Three different strategies were investigated: homogeneous (all robots may deliver any object), specialize-by-color and territorial (the last robot is responsible for sorting and final delivery). The homogeneous strategy was found to be the optimal behavior for this task.

Robotic soccer is another popular domain for multi-robot research. The annual Robocup event [Kitano et al. 1997] has attracted many participants and is active arena for research in cooperative robot teams. [Roth, Vail and Veloso 2003] describe a Robocup team consisting of Sony Aibos that use a shared world model. The shared model is generated from data that is broadcast from each team member at 2Hz. This approach is very similar to the broadcast-and-aggregate method described in the previous chapter; however, due to the high latency, the shared world model is only used to make control decisions when a robot cannot rely on its own individual world model. [Vail and Veloso 2003] describes the use of this shared world model mechanism to implement role assignments via shared potential fields.

Formation maintenance is another common multi-robot task. The Alliance architecture previously mentioned has been used to implement robots in a line-abreast formation that navigate through waypoints to a final destination. [Balch and Arkin 1998] describe a set of motor schemas that implement four different formations based on military doctrine in a team of robots. Two different strategies for maintaining the

formations are discussed: unit-centered-referenced and leader-referenced. The former strategy performs better, but uses more bandwidth for communication.

3.3 Limitations of Behavior-based systems

For all the advantages and convenience of behavior-based systems, they do face some serious issues. The strengths of this approach are also its weaknesses. Since behavior-based systems obey circuit semantics, they are equivalent to parallel circuits and are therefore restricted to the class of computations expressible as such circuits. This restriction limits the representational power of behavior-based systems; all representations in the control system are based on wires transmitting scalar values and are therefore essentially limited to propositional representations. That is, behavior-based systems are limited to representations without predicate/argument structure, term expressions, or any data types requiring dynamic graph or tree constructs. While it is possible to support such structures in parallel networks by introducing a switching network, this approach is expensive and a potential bottleneck.

For example, suppose the sensory system on the behavior-based robot is capable of tracking objects of various colors. The sensory system can determine if the robot is facing a colored object (i.e. it provides the propositions *facing-blue-object?*, *facing-red-object?*, *facing-green-object?*, etc), and knows if it is currently located near the object (the propositions *near-blue-object?*, *near-red-object?*, *near-green-object?*, etc). Each of these propositions is represented as a wire that carries its truth-value to the rest of the

reasoning system. The robot may grab an object if it is both near the object, and sufficiently close to it, i.e.

$$\text{see-object}(X) \wedge \text{near-object}(X) \Rightarrow \text{can-grab-object}(X)$$

However, for circuit semantic systems, this logic has to be repeated for every color the sensory system can track. That is, to perform the equivalent of variable binding in predicate logic, the inference network that controls the *grab* proposition has to be cloned for every instance of color, as shown in Figure 3.4.

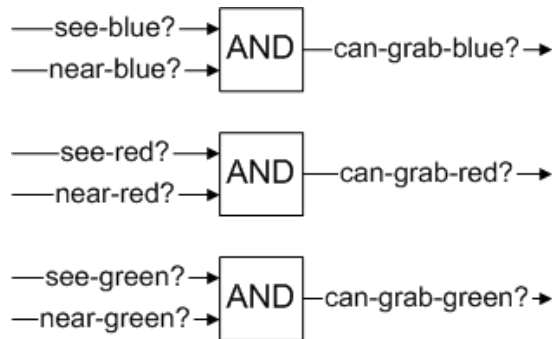


Figure 3.4: Propositional explosion in the inference network

The inability to cleanly represent these structures in behavior-based systems can result in large networks, making most reasoning and planning tasks both difficult and clumsy. The behavior-based solution to the blocks-world problem, for example, requires distinct nodes for each ground instance of each predicate and action [Maes 1990]; a behavior-based dialog system [Hasegawa, Nakano and Kato 1997] required separate behavior modules for each possible utterance of each possible speaker. Such reasoning networks are ultimately exponential in the order of the domain theory's highest-order predicate or

action. Since most multi-robot controllers are extensions of behavior-based techniques, they ultimately inherit the same issues from the basic underlying architecture.

Chapter 4

Symbolic reasoning on robots

This chapter discusses the implementation of symbolic reasoning systems on multi-robot controllers. Symbolic reasoning systems use search algorithms on dynamic graph structures stored in world model or knowledge base. Their output is generally a plan (i.e. sequence of steps or actions) that is performed by an executor. These symbolic planners can use representations that are far more powerful and expressive than the behavior-based systems discussed in the previous chapter because they are not constrained by propositional structures. However, they encounter some problems when faced with rapid and dynamic changes in the real world.

4.1 Model Coherency and Inference Tracking

One shortcoming of symbolic reasoning systems is their use of a database-like world model. The world model used by the reasoning system has to be grounded in the physical world, and kept constantly up to date. This information is stored as a set of logical assertions in a database, indexed perhaps by predicate name [Russell and Norvig 1995].

The symbol grounding problem is of particular concern for autonomous robots

that are forced to interact with a highly dynamic and unpredictable world. The autonomous robot literature contains numerous examples of work that has addressed this issue (e.g. [Hexmoor, Lammens and Shapiro 1993], [Bajcsy and Kosecka 1994] and [Wasson, Kortenkamp, and Huber 1999]). Some researchers have focused on the related issue of symbol anchoring, i.e. the problem of reacquiring a physical grounding after having lost sensory contact with the object. [Coradeschi and Saffiotti 2000] proposes a formal framework for creating and maintaining a correspondence between a symbol and a physical object in the real world; subsequently, [Coradeschi and Saffiotti 2001] extended this work to encompass action properties (i.e. properties necessary for an action represented by a symbol), partial matches and indefinite references. Other researchers have proposed allowing the robots to establish their own symbolic groundings. For example, [Steels 1998] and [Vogt 2000] describe multi-robot systems that use language games to construct symbol-meaning associations.

The sensors on a robot generate beliefs or assertions for the knowledge base. In this dissertation, I assume a broad definition of the term *belief*. Specifically, I take a belief to either be a proposition (e.g. *see(convoy232)*) or an assignment of a value to a function (e.g. *longitude-of(convoy232)*). The values of beliefs vary over time as data from the sensors changes. However, in a way, each sensor on the robot has its own small, fragmented model of the world. Somehow, these models have to be synchronized with the central knowledge base used by the symbolic reasoner for the robot to display coherent behavior. That is, the beliefs or assertions in the central knowledge base have

to reflect reality in a timely manner. If a tracking module in the sensory system updates its representation of the distance of an enemy convoy, then some mechanism must ensure the symbolic system's representation of that is updated too.

Additionally, beliefs in the central knowledge base can be dependent on other beliefs. For example, the belief that an area is safe could depend on the assertion that the robot does not currently observe any enemies in the area. If the latter assertion is withdrawn, then the former must be too. Hence, each update from the sensory 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.

The former issue is known as the *model coherency problem* and the latter the *inference tracking problem* [Horswill et al. 2000]. Behavior-based systems are particularly successfully at dealing with these issues; when new data arrives at the sensory systems, there is a natural data flow path from there to the rest of the reasoning system, seamlessly updating the robot's view of the world. On the other hand, at present, most symbolic systems have to solve these issues by relying on the programmer to handcraft rules in the domain theory to specify when to run epistemic actions. This

can lead to significant events being missed because the knowledge base was not updated at the proper time.

4.1.1 Implications for Multi-Robot Systems

Multi-robot controllers naturally inherit the characteristics of the systems they are based on. In general, the problems faced by symbolic systems are exacerbated in multi-robot scenarios. 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.

The robots on a cooperative team establish a consistent team-wide representation of the world through a set of joint beliefs, i.e. beliefs that have identical values across all robots. However, this synchronization of joint beliefs across distributed team members can be a difficult endeavor. The programmer is now responsible for maintaining coherence across multiple distributed platforms connected via tenuous RF links, which are well-known to have higher error rates [Eckhardt and Steenkiste 1996] [Xylomenos and Polyzos 1999], and hence higher message delays, than their regular wired counterparts. These problems complicate the task of ensuring that data is shared and joint beliefs are generated in a timely fashion.

Some researchers in the multi-agent community have proposed strong coordination protocols for controlling teams of agents. The Joint Intentions framework [Cohen and Levesque 1991] coordinates cooperating agents through the use of joint persistent goals. Agents on a team can form a joint persistent goal only if they mutually

believe that the goal is unsatisfied and is currently a mutual goal. If an agent comes to believe that a joint persistent goal has been satisfied, or can no longer be satisfied, then that agent is obligated to inform all other team members of this. The same mechanisms that ensure synchronization when attempting to form a joint goal are also used to enforce communication when plans break down. Each agent that wishes to enact a joint persistent goal first has to notify the other team members. Conversely, agents who have accepted the joint goal must confirm this through explicit communication.

The Joint Intentions framework is essentially a blocking commit protocol that ensures synchronization between teammates. At key points in the process of computation (e.g. when team members must agree to establish a joint goal), the agents are blocked from taking further action until the protocol completes. Joint Intentions ensures team-wide consistency by ensuring that agents always store relevant beliefs (in this case, joint goals) in their knowledge bases before subsequent actions are taken. However, the act of synchronization on joint goals is considered a primitive action in the original Joint Intentions theory and there is little discussion on how this is actually accomplished or how long it takes.

[Tambe 1997] describes an implementation of Joint Intentions where the joint persistent goals are created through the use of personal achievement goals. This implementation handles inconsistencies in world models by designating one agent as a team leader that maintains the “master copy” of the knowledge base; all other agents defer to the leader’s decisions, even if they disagree, i.e. have beliefs that contradict the

leader's choice. The team leader is the final arbiter of joint persistent goals. However, this can be problematic since the team leader's decision may not be the best one. Dissenting agents may have access to information the team leader does not have. Furthermore, should the current leader fail, the team is forced to spend valuable time electing a new one.

While the Joint Intentions framework specifies the establishment of joint goals as well as certain mutual beliefs upon goal failure, it does not prescribe a method of synchronizing beliefs in general. As my previous example from Chapter 1 shows, cooperation involves more than the creation of team goals, it also involves the sharing of joint beliefs that are pertinent to the team's objectives. This lack of explicit joint belief synchronization can lead to situations where the agents have inconsistent knowledge bases, e.g. when team members disagree with the team leader's decisions as noted in [Tambe 1997]. [Qiu and Tambe 1998] attempts to remedy this problem by reconciling any conflicts that are detected through the use of a negotiation protocol.

4.2 Alternatives to Traditional Symbolic Systems

In response to the shortcomings discussed above, researchers have proposed some modifications to symbolic reasoning systems. In most cases, these modifications have focused on the speed and reactivity of the system. By and large, less focus has been placed on a solution to the model coherency and tracking issues.

4.2.1 Tiered architectures

In the robotics community, the state of the art is to construct a tiered system that incorporates reactive and symbolic systems as components. That is, one or more symbolic systems are run in parallel with the behavior-based system; the symbolic system is responsible for high-level reasoning and planning, while the behavior-based system handles low-level actions.

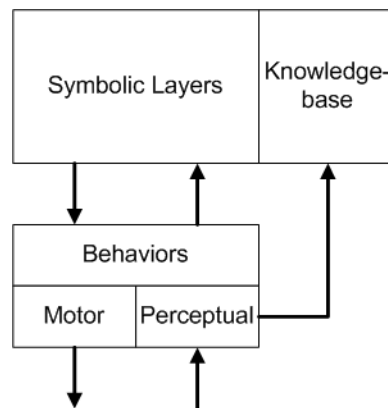


Figure 4.1: Typical Tiered Architecture

The Autonomous Robot Architecture (AuRA) [Arkin 1986][Arkin and Balch 1997] was among the first systems to combine deliberative and reactive systems. AuRA used a conventional planner that could reason over a modular behavior-based control system; [Arkin 1989a] describes the first robot navigational system to be presented in this integrated manner. Atlantis [Gat 1991] is a three tiered system that includes a deliberative planner, a sequencer modeled after the RAPs system [Firby 1989] and a reactive controller. [Lyons and Hendriks 1992] proposed using the planner as an

execution monitor that adapts the underlying behavioral control system to the agent's goals and changing environment. Other examples of successful tiered architectures include 3T [Bonasso et al. 1997], the Procedural Reasoning System (PRS) [Georgeff and Lansky 1987] and SSS [Connell 1992].

The main issue with this approach is that tiered architectures per se do not resolve the model coherency and inference tracking problem. The symbolic layers ultimately still use a separate knowledge base of logical assertions that must be kept in sync with the representations used by the behavior-based layer. Moreover, if the tiered system has multiple symbolic layers (e.g. [Bonasso et al. 1997]), the world model for each would have to be kept consistent with one another. These issues are typically left up to the programmer, who is responsible for adding the proper rules in the domain model to ensure proper world model updates. The fact that many tiered systems use the planner only during startup to configure the behavior-based layer is a reflection of this world model update problem.

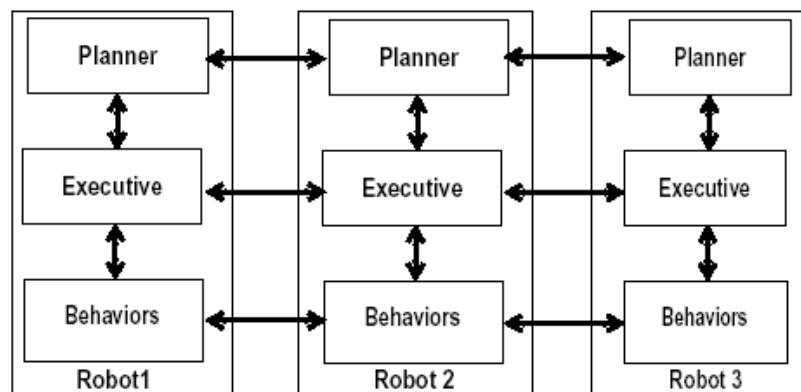


Figure 4.2: DIRA architecture and interaction paths [Simmons et al. 2000].

There are very few implemented physical multi-robot systems that utilize symbolic reasoning systems (either standalone or in a tiered system) on every robot while coordinating through active communication.

One instance of such an approach is the Distributed Robot Architecture (DIRA) [Simmons et al. 2000] [Simmons et al. 2002]. A planner decides the high-level goals, an executive layer sequences and monitors task execution, and the behaviors interface directly to the robot's sensors and actuators. The layers interact in the manner shown in Figure 4.2. This system has been demonstrated on a three robot visual servo-ing task as well as a multi-robot map-building task. The vast majority of communication occurred at the behavioral layer and almost no communication took place at the planner level.

Another example is [Jung and Zelinsky 2000], who describe a team of cleaning robots that communicate through the use of symbols. They define a symbol as a relationship between icons (physical representations), indexicals (associations between icons) or other symbols. For example, a robot observing a pile of litter may communicate this information to its teammate through a symbol containing two indexicals representing known or labeled locations, and a set of wheel encoder data. This symbolic relationship defines a grounded unique location that the teammate can use to navigate towards the litter.

4.2.2 Mapping to circuit semantic systems

An alternative approach is to find methods of mapping cases of symbolic reasoning onto parallel circuit semantic systems. The parallel reasoning networks can then compute the same input/output mappings as a planner, but run in bounded time.

[Rosenschein and Kaelbling 1986] is the earliest example of such a system; in this case, propositional logic axioms were compiled into sequential circuits. The GAPPS system [Kaelbling 1988] used goal regression to compile a propositional planner-like formalism into sequential circuits. [Maes 1989] describes a behavior network system that could compute an approximation of propositional STRIPS planning using spreading activation.

However, as discussed in the previous chapter, the use of propositional logic's lack of variables or predicate/argument structure is highly limiting. [Agre 1997] has argued that this problem is ultimately inescapable, and that we must consider alternative forms of abstraction and representation.

One solution is to grow the propositional reasoning network incrementally as new combinations of arguments are encountered. Some examples of this approach are TeleoReactive Trees [Nilsson 1994], Running Arguments [Agre 1988], Truth Maintenance Systems (TMS) [Forbus and Klerer 1993] and SOAR [Newell 1990]. This technique caches the chains of reasoning and beliefs that have been performed by the symbolic system in a propositional network, allowing subsequent searches along similar lines to be much faster. Traditionally, these networks grow monotonically in the size of

the cache, which can lead to high maintenance cost. One way of alleviating this is to remove the parts of the reasoning network that are no longer needed; for example, [Everett and Forbus 1996] describe a garbage collection algorithm that eliminates the storage overhead for Truth Maintenance Systems.

4.2.3 Deictic Representation

Rather than growing the propositional reasoning network, Agre and Chapman have argued for the use of indexical constants as a surrogate for variable binding to improve on the limitations of propositional representations [Agre and Chapman 1987]. Instead of putting a variable binding mechanism into the symbolic reasoning engine, they implemented a propositional reasoner whose inputs were driven by an active vision system. The outputs of the vision system directly measured the truth values of a fixed set of literals such as *near(the-tiger-that's-about-to-eat-me)*. In the metatheory, *the-tiger-that's-about-to-eat-me* is an indexical name whose denotation is determined by the current attentional state of the perceptual system. By redirecting attention of the visual system from one object to another, the system could effectively treat the name as a variable and rebind it from object to object. However, since the reasoning system did not represent the internal structure of the literals, it was purely propositional and so could be implemented as a feed-forward logic network. Moreover, since vision systems are generally attentive in practice, this “variable binding” mechanism was already present. No additional computations needed to be added to take advantage of it.

However, sensory systems only have a limited set of trackers that can be assigned to a small number of objects and continuously report their characteristics over time. The problem with this *indexical-functional* or *deictic* style of representation is that there tend to be more indexical names than there are trackers in the vision system. In practice, it becomes necessary to decide at design time that a certain set of indexicals will share one tracker, while another set will use another tracker. The designer then hopes that there will be no situations in which two indexicals that share a tracker will need to be bound to distinct objects. In addition, this approach does not provide the ability to utilize quantified inference, which is a useful and powerful representation to have in a reasoning system.

4.3 Role-passing

Role-passing [Horswill 1998] is a variation of deictic representation that solves the two problems associated with this approach (see Section 3.2.3). Rather than postulating a potentially unlimited number of task-specific variables, role-passing provides the system designer with a small, finite set of domain-independent indexicals. These indexical variables are in the form of linguistic role names such as *agent*, *patient*, *object*, *source*, *destination*, etc. Role-passing is similar to pronomes [Minsky 1986] and the SHRUTTI system [Shashtri and Ajjanadadde 1993]; however, the binding is performed in the sensory system rather than inside the central reasoning system.

When the control system on the robot binds a role to an object, a sensory tracker is dynamically allocated to it and tagged with the name of the role. The tracker is tagged with that role, and it can then be accessed associatively by that tag. Inference rules may utilize the information provided by the tracker through the role without needing to know the origin of that data. That is, role-passing allows behavior-based systems to abstract over both sensory systems and objects. In fact, an object may even have multiple representations across different sensory modalities.

As an example, suppose the robot binds the role *patient* to a blue colored ball; the tracker assigned to the ball is a standard color blob tracker that provides data such as whether the object is in view, the distance from the robot to the object, etc. Now suppose there is an inference rule

If see(X) and near(X), then can-grab(X)

Assuming the blue colored ball is directly in front of the robot, and close to it, then this inference rule will allow the robot to infer that it *can-grab* the object that is bound to the role *patient*. Notice however, that the inference rule did not know the origin of the information for the object bound to *patient*. The *patient* role can later be rebound to some other object (say, a green ball); however, all this is conveniently hidden from the inference rules that utilize the indexical roles.

4.3.1 Compact Storage

Two types of information can be accessed through the indexical roles about objects that are being tracked: Boolean unary predicates or numeric functional values. An example of a unary predicate would be *near(X)*, and a function would be *distance-to(X)* or *size-of(X)*.

Since the number of linguistic roles is finite and relatively small, unary predicates can be represented as bit-vectors, with one bit per corresponding role. If the *i*th bit of the vector is set, then the predicate is true of the object bound to the *i*th role, as shown in Figure 4.3. In the control system, the bit-vector with only the *i*th bit set would refer to the object bound to the *i*th role. On a standard Von Neumann machine, a unary predicate can be efficiently represented as a single memory word.

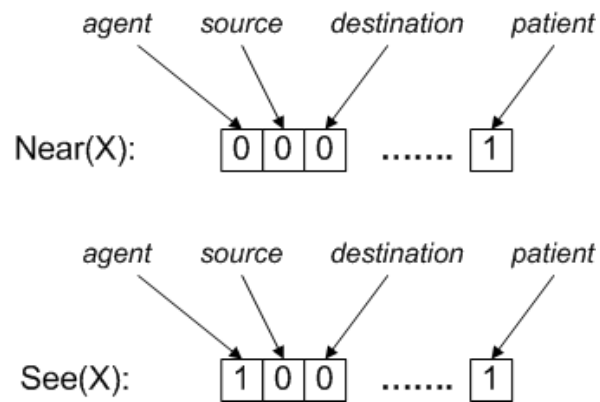


Figure 4.3: Unary predicates represented as bit vectors

Real-valued functions are represented in a similar manner. A function is a vector of numbers in which the i th slot of the vector contains the value of the function on the object bound to the i th role (see figure below).

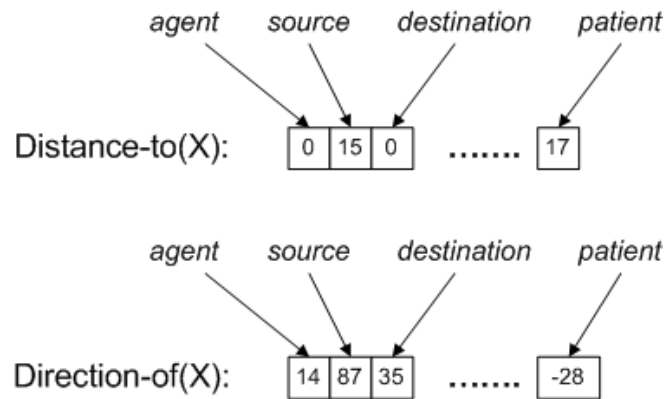


Figure 4.4: Functional outputs represented as numeric vectors

4.3.2 Efficient Inference

The representation of unary predicates as fixed-size bit-vectors at compile-time allows them to be efficiently implemented. Forward-chaining modal Horn clause inference can be implemented as simple feed-forward Boolean networks. Each predicate can be considered a compact bus of wires, with each wire holding the truth value for its corresponding indexical role. On conventional hardware, these reasoning networks can be compiled to straight-line code consisting only of loads, stores, and bit-mask instructions. Logical connectives can be implemented as bit-wise logic operations, e.g.

$$(P(x) \vee Q(x)) \wedge R(x)$$

can be evaluated as

$(\text{bitwiseand } (\text{bitwiseor } p \ q) \ r)$

This approach is closely related to the automated compilation of domain axioms to digital logic described in [Kaelbling and Rosenschein 1991].

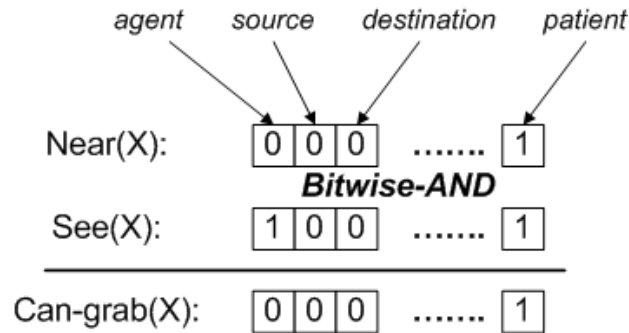


Figure 4.5: Efficient inference using bit-wise logic operations on unary predicates

Unfortunately, compiling role-passing inference requires hardware exponential in the arity of the highest arity predicate in the rule base. This is why role-passing is limited to unary predicates in practice. However, we have found that this approach is sufficient for most robot control programs so far. Cases where this technique falls short have been relatively infrequent. So, while the role-passing approach is more limited than a full logic-programming system, it does allow us to express much of the kinds of control reasoning that people really implement on physical robots today. Role-passing allows inference rules expressed using indexical roles be compiled into a form that can track all inferences at sensor rates. Since the reasoning process recomputes all inferences on every cycle of the system's control loop, the reasoning system is able to respond to contingencies as soon as they are sensed. Furthermore, the compilation process is very efficient, resulting in inference that is effectively free; 1000 Horn

clauses of 5 conjuncts each can be completely updated at 100Hz using less than 1% of a current CPU.

Chapter 5

HIVEMind

The HIVEMind architecture is a result of joining the broadcast-and-aggregate method for coordination with the role-passing architecture for robot control systems. The result is a multi-robot system that has the ability to use symbolic reasoning over tightly synchronized knowledgebases. This chapter describes the design of the HIVEMind architecture and its characteristics.

5.4 Control

The underlying control system on HIVEMind robots is based on role-passing. Therefore, the robots are able to use more structured representations while retaining the advantageous characteristics of conventional behavior-based systems. The entire HIVEMind team can be considered a single, parallel control network whose components happen to be distributed between the different robot bodies being controlled. It may seem inefficient for each robot to have its own separate copy of the inference rule network. However, to have a single robot perform each inference and share the results would require much more complicated coordination protocols

analogous to the multi-phase commit protocols used in distributed database systems. Since role-passing inference is essentially free, it is more efficient for HIVEMind robots to perform redundant computation.

The inference rules on HIVEMind robots are not based on propositional values such as *see-blue-object* or *see-red-object*, but rather on predicates such as *see-object(X)*. Furthermore, since the representation for such structures in role-passing is finite and highly compact, they can be easily shared using the broadcast-and-aggregate method. Each unary predicate and functional quantity (e.g. *size-of-object(X)*) is treated as a signal and communicated as such.

5.1 Coordination

The HIVEMind architecture uses the broadcast-and-aggregate method to coordinate the robots and the periodic data broadcasting technique for sharing data. The latter is accomplished using the non-blocking broadcast-oriented User Datagram Protocol (UDP). The present implementation of HIVEMind assumes that the set of signals shared by each robot is identical and known during compile-time. The shared data are generally role-passing predicates and functions. For example, the robots could share the values of two predicates (*see(X)* and *near(X)*) and two functions (*size-of(X)* and *distance-to(X)*).

In general, all shared signals can be conveniently crammed into a single packet per communication cycle since the representations are very compact. However, this

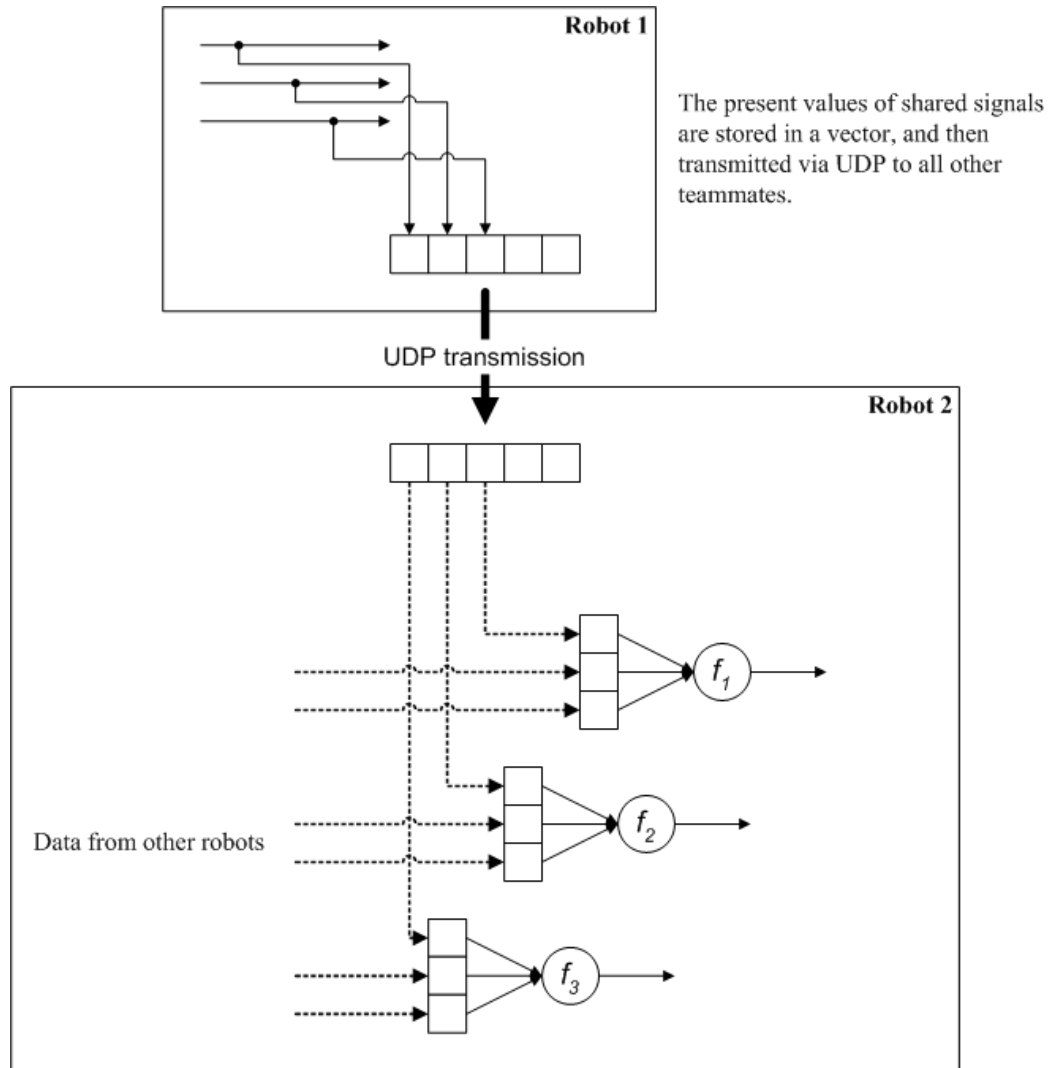
might not always be true. The current HIVEMind implementation can automatically split the shared data into multiple packets if it exceeds the 1024 bytes available in a single UDP packet, but this feature has never been utilized in practice.

By default, the shared data are transmitted once a second by all robots, which our experiments have shown to be sufficient for maintaining situational awareness among the team members for the tasks they perform. This interval can be increased or decreased depending on the needs of the application.

The majority of this data is in the form of the compact role-passing predicates and functional values discussed previously. However, nothing precludes other types of data from being shared. For example, the robots may wish to share information on locations that have been searched. This information could be represented as a bit vector of locations where a set bit indicates a location that has been personally searched by the current robot; a disjunction of these bit vectors would indicate the locations that have been searched by the robots as a team. This *locations-searched* bit vector for each robot can also be treated as a signal and sent using broadcast-and-aggregate.

5.1.1 Unique Identification Numbers

Each robot on the HIVEMind team has a unique identification number that is attached to all outgoing messages. This allows robots to discern the origin of each received communiqué.



Vectors from the incoming UDP messages are broken up into their individual components and stored into appropriate buffer slots (which are indexed by robot id numbers). On every decision cycle, the current values from each robot for the components are combined through an aggregation function (f_n).

Figure 5.1: The Hivemind Architecture in practice

5.1.2 Maximum Team Size

The amount of buffer space available for storing the contents of incoming messages is fixed beforehand. This is achieved by predetermining the set of shared data for all robots and also fixing the maximum number of robots that can be on a HIVEMind team during compile-time. There can be fewer members on the team than the maximum number during run-time, but never more than that upper limit. At present, the maximum team size is set at ten. However, there is no theoretical limitation to the number of robots that can serve on a HIVEMind team. In the present implementation, the programmer simply has to change the value of the variable *maximum-team-members* and recompile the code for the robots in order to increase the size of the team.

5.2 Aggregating Shared Data to form Joint Beliefs

When an incoming UDP packet is received by a robot, the components of the list it contains are stored into locations indexed by the originating robot's unique identification number. Returning to the previous example, the incoming messages from teammates contain their values for the two predicates (*see(X)* and *near(X)*) and two functions (*size-of(X)* and *distance-to(X)*). The local robot has a storage vector for each piece of shared data, which has size equal to *maximum-team-members*. The values in the incoming message are extracted and stored into the storage vectors at a location indexed by the originating robot's identification number. For example, the value of

$see(X)$ in a message from robot with an identification number of 3 will be stored into the third slot of the $see(X)$ storage vector.

On every process cycle, the current values for each component from all robots are aggregated into a single joint belief output value. In general, each joint belief has its own aggregation function. For example, the latest values in the $see(X)$ storage vector could be combined using an OR operation to generate the joint belief $team-see(X)$. The joint beliefs are then passed to the inference rules for reasoning.

As more HIVEMind systems are built, we will have a better understanding of the commonly used aggregation techniques. These commonly used aggregation functions would be collected in a toolkit that is available for use by multi-robot system developers. At present, I have written several aggregation functions that form the beginnings of such a toolkit:

- *predicate-or*

Since unary predicates in role-passing are bit-vectors containing information for all indexical roles, *predicate-or* is implemented as:

$$predicate-or(X) = \vee x_r, r \in R$$

where x_r is the bit-vector representing the unary predicate from robot r in team R , and \vee is a bitwise-or operation. A predicate value is true if at least one robot on the team believes it to be true. For example, *some-robot-see(X)* is generated using a *predicate-or* of individual $see(X)$ values from team members.

- *predicate-and*

The *predicate-and* function is implemented as:

$$\text{predicate-and}(X) = \bigwedge_{x_r, r \in R}$$

where x_r is the bit-vector representing the unary predicate from robot r in team R , and \bigwedge is a bitwise-and operation. The predicate value is true if and only if all robots on the team believe it to be true. For example, *all-see(X)* is true only when all members of the team currently observe the object bound to the role, i.e. the *see(X)* predicate is returns true for the bound role on all robots.

- *first-non-false*

Returns the identification number of the first robot that has a TRUE value in a predicate for a particular indexical role. For example, this function could be used on the value of the *patient* role for *see(X)* from the individual robots. The important point is that some subset of the team has seen the object bound to the *patient* role, and the programmer wants to know the id number for one of those robots. This aggregation function is generally used in conjunction with the following one.

- *get-value-by-index*

Returns a functional value based on a role variable and an identification number for a robot. Essentially, this function retrieves a current value of a function generated by a specific robot for a particular role variable. For example, if a teammate claims to *see(X)* for the object bound to the *patient* role, the current

robot may obtain the specific value of *location-of(X)* for *patient* using that teammate's identification number.

- *mean*

Returns the average value of the outputs for a function from all team members.

For example, *average-count* is the *mean* of *count(X)* from all robots on the team.

- *min* and *max*

Correspondingly, *min* and *max* returns the minimum and maximum value of the outputs from a function generated by all robots respectively.

5.2.1 Multi-robot task allocation (MRTA)

An important special case of joint beliefs is the assignment of responsibilities during the execution of a task. Ideally, the most suitable robot should be chosen to perform each goal or task, where “suitable” is defined by some metric represented as a reward or cost function. For example, the goal could be for one member of the team to move over to a particular location (say, the X spot in the figure below). A simple way of determining which robot should be responsible for that goal is by using the Cartesian distance from the robots' present positions to the target location; the closest robot could then be the most suitable member of the team to take the responsibility.

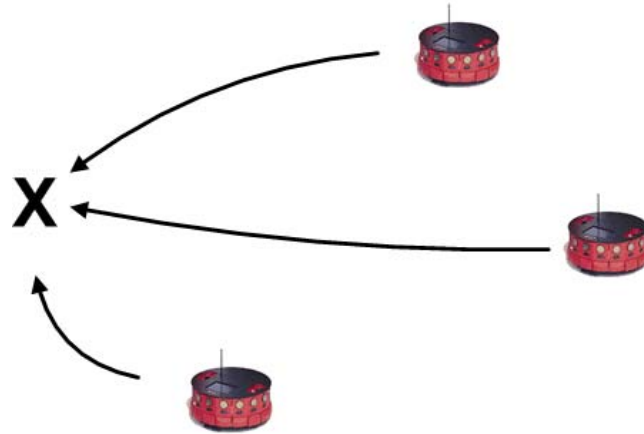


Figure 5.2: Determining which robot is responsible for going to the X position

Suppose there are concurrent jobs $T_1 \dots T_m$ that need to be accomplished by robots $R_1 \dots R_n$. If $m \neq n$, then we can add virtual goals or robots until both sets are the same size. Assume that only one robot will be assigned to each goal. Each robot-job pairing has an associated reward value s , which is computed as:

$$s_{ij} = f(b_{ij1}, b_{ij2}, \dots, b_{ijk}), s_{ij} \in R$$

where $b_{ij1}, b_{ij2}, \dots, b_{ijk}$ are the beliefs for robot i that are relevant to goal j and are used to compute the reward. In the example above, the relevant beliefs could be the robot's present coordinates and those of the target location. The multi-robot task allocation (MRTA) problem is to find a set of robot-goal pairs that maximizes the total reward function; in other words, we want a set of pairings

$$(r_1, t_1), (r_2, t_2), \dots, (r_m, t_m)$$

that maximizes

$$\sum_{i=1}^n S_{(r_i, t_i)}$$

The single-robot-single-task assignment problem has been shown to be a degenerate form of the scheduling problem [Gerkey and Mataric 2003] that can be solved using Kuhn's Hungarian method [Kuhn 1955] which runs in $O(n^3)$ time.

HIVEMind robots presently recompute all their inferences and joint beliefs on every process cycle, so determining the global optimum for teams on the order of, say, 1000 robots and concurrent goals could be prohibitively expensive. At the moment, however, the size of most physical robot teams and the complexity of achievable tasks are low enough that computing the global optimum for assignments on every cycle does not use a significant amount of time.

There are two ways of implementing the reward function for the team:

- All relevant beliefs can be shared among team members, so each robot can not only compute the suitability of itself for the task, but also the suitability of all other team members.
- The suitability value be calculated locally by each robot and then shared as a belief with all team members.

For the experiments described in Chapter 8, I used the former method; some of the beliefs used in the computation of the metric had to be shared for other purposes and hence it was more convenient to compute the metric for all robots locally.

5.2.2 Clearing slots in the storage vectors

Each storage vector is allocated a number of slots equal to the value of *maximum-team-members* during compile-time. During run-time, however, not all these slots will have robots associated with them. There could be fewer robots than the maximum size currently active, or a robot could have experienced communication or total system failure. Therefore, there must be some way of letting the aggregation functions know that a slot in the storage vectors does not contain a valid value.

Every storage vector has a value associated with it that is designated “invalid” or “nogood”. By default, this is simply the FALSE value (`#f`) in Scheme; the programmer may specify another value if necessary. When the values in the storage vector are being combined by the aggregation functions, any slots labeled with an invalid value are ignored.

5.3 Configuring a HIVEMind team member

HIVEMind robots are configured using a macro called *define-default-peer*. This macro invokes a set of Scheme functions and generates some interface structures in a programming language called Generic Robot Language (GRL), which is the language used for the control programs on the robots and is explained in greater detail in Chapter 6. The *define-default-peer* macro is invoked in this way:

```
(define-default-peer some-name  
  
  bot-id
```

...<list of aggregation functions> ...

... <list of data to be transmitted> ...)

Bot-id is the unique identification number for the current robot. The list of aggregation functions are either drawn from ones in the toolkit described in Section 5.2, or may be programmer-defined for the current task. Finally, the list of data to be shared may either be unary predicates or functions with role indexicals, or some other GRL data structure.

The macro itself is defined as:

(define-syntax define-default-peer

(syntax-rules ()

((define-default-peer ?peer-name

?bot-id

(?handler ...)

(?peerval ...))

(begin

(init-vector-storage (length (list ?handler ...)))

(init-role-handlers-list ?handler ...)

(set-peer-handlers)

(set-*peer-messages*)

(define-peer ?peer-name ?bot-id ?peerval ...))))))

init-vector-storage allocates the storage vectors used to hold the contents of incoming messages and initializes them with the “invalid” value. *Handlers* are the aggregation

functions used to combine beliefs from the team in order to generate joint beliefs; *init-role-handlers-list* and *set-peer-handlers* initialize the aggregation functions and store them in the appropriate data structures. The *set-peer-messages* function creates the data structure that will hold the latest belief values that are to be shared with teammates. Finally, *define-peer* is another macro; it is defined as:

```
(define-syntax define-peer
(syntax-rules ()
((define-peer ?peer-name ?bot-id ?peerval ...)
(begin
(define-signal peer-info
(set-peer-information ?bot-id ,(length (list '?peerval ...))))
(create-set-peervals-transducer
(length (list '?peerval ...)))
(define-signal peer-values
(set-peervals ?peerval ...))
(define ?peer-name
(list peer-info
peer-values
multiagent-send))))))
```

The *define-peer* macro is responsible for creating a set of GRL data structures that provides an interface from the HIVEMind system to the rest of the reasoning system on

the robots. The interface structures are used to connect the joint beliefs generated by the HIVEMind system to the inference rules on the robots.

5.4 Characteristics of HIVEMind

5.4.1 Communication costs

HIVEMind relies on the User Datagram Protocol for communication of team messages. The data size for UDP packets can ostensibly be set by the programmer in the packet header and be up to 65k bytes in length. In practice, however, this is often constrained by hardware imposed limitations; packets that are in excess of imposed limits may be treated as invalid and simply discarded. Larger data lengths can produce better throughput, but also increase the chance of errors in the packet during transmission.

The present HIVEMind implementation uses non-blocking broadcast-oriented UDP sockets. Any messages sent by a robot are received by anyone who is listening on the same port on the same subnet. The buffer size for the data portion of the UDP packet is capped at a maximum of 1024 bytes. By default, each robot sends one UDP packet per second, meaning that the aggregate bandwidth required for coordination is bounded by 1KB/robot/sec.

From a hardware perspective, the robots communicate in a peer-to-peer fashion using an 802.11b wireless LAN at 11Mbps for communication. At the one UDP packet per second rate, each robot is only using approximately 0.07% of the available bandwidth. As wireless LAN technology improves, each robot will use an even less

percentage of available bandwidth, assuming the communication rate of one UDP packet per second remains the same; e.g. the newer 802.11g specification is backward compatible with 802.11b but provides 20+Mbps of bandwidth. Therefore, robot teams of more than a hundred robots should be practical from a communication standpoint.

5.4.2 Failure awareness

Physical robots face the possibility of communication breakdown or even total system failure. Robots in a cooperative team must be able to somehow account for this occurrence. A related issue is that a new robot could join the team at any time or a robot could rejoin the HIVEMind after an interval of communication failure. So, it is also useful to have a system in place for handling the dynamic addition or subtraction of robots from the team.

The current implementation of the HIVEMind architecture is Receive Failure Aware (RFA); it is not Transmission Failure Aware (TFA), but can be trivially made so. HIVEMind robots maintain a *last-heard-from* value for each member of the team. If a teammate has not been heard from for over 10 seconds (a programmer tunable value), then it is considered “dead” and its data values in the storage vector are automatically cleared so that no stale information is used for reasoning. Robots only use information from “live” teammates for reasoning. The *last-heard-from* variable will be updated with a new value when communication is restored to an old teammate or a new robot joins the group, allowing the HIVEMind to seamlessly assimilate a new member into the

team. Newcomers are able to integrate into the HIVEMind within one communication transmission cycle because of the assumption of a convergence property for data values (discussed in Chapter 2), meaning that only the latest signal values matter.

5.4.3 Negotiation

The design of the HIVEMind architecture negates the need for negotiation amongst team members in many cases. HIVEMind robots are able to maintain a shared situational awareness in real-time and are therefore able to take the right actions in an appropriate fashion. In general, negotiation has been used in three cases to synchronize team actions.

The first case where negotiation is required is when the cooperating entities are self-interested and have different end goals. In this case, negotiation is required to satisfy all parties that their interests have been accounted for and to obtain an acceptable resolution to any conflicting actions. Communication of information can be tricky under these circumstances since the agent must balance the need to negotiate versus the danger of revealing too much. There is much research on techniques to achieve consensus for groups for self-interested agents, mostly in the realm of game theory (for example, see [Kraus 2001]). However, the HIVEMind approach assumes that all agents on the team share common objectives and may freely share any information. Therefore, self-interest is not an issue and this condition for negotiation does not apply.

The second situation requiring negotiation concerns differing world views among team members; even if the agents are fully cooperative, it is possible that team members may have different sets of beliefs due to sensory limitations. Therefore, one or more team members might lack information that leads them to undertake suboptimal or perhaps even contradictory courses of action. An example of this situation has been discussed previously in Chapter 4: The Joint Intentions framework [Cohen and Levesque 1991] explicitly synchronizes team goals, but does not specifically coordinate beliefs in general, which can lead to cases where team members explicitly lack the necessary information to perform agreed-upon team actions; [Qiu and Tambe 1998] specifies a negotiation framework that attempts to remedy this situation for agents that utilize joint intentions. HIVEMind robots have two features that allow them to circumvent this problem. First, the present values of any relevant team beliefs are communicated to all teammates at regular intervals, ensuring that fresh information is continuously being shared across the team. Second, the convergence property for belief values allows communicated information to be quickly integrated into the reasoning process. These two advantageous features allow HIVEMind robots to share a common situational awareness, and there is no necessity for additional negotiation mechanisms to ensure synchronization. While communication lag can cause HIVEMind robots to have slight different data (and hence different states) at the same time, this is a problem all teams using communication have to deal with and the HIVEMind robots generally converge to the same team state eventually.

Finally, different opinions on the appropriate course of action to take in a given situation could lead to a need for negotiation; that is, it is conceivable that even with perfectly synchronized beliefs, the robots could differ on the best action to perform in order to achieve team goals. Specifically, each member has its own internal plan, and at least some of the plans differ. In this case, 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. A simple example will serve to show this situation is not a problem for HIVEMind robots and certainly does not necessitate additional complex negotiation protocols to solve: Suppose two cooperating robots have different notions as to the right action to take in a given situation. There are two possibilities: one plan is better (by some measurable metric) than the other, or the plans are equally good. In the first case, since HIVEMind robots always have a shared situational awareness, it should be trivial to determine which plan is better given the team's current knowledge. For the second case, either plan is acceptable, so one simple solution is to utilize an arbitrary tie-breaker to determine which plan "wins"; in the HIVEMind case, since each robot has a unique identification number, the lowest number is chosen to always win in the case of a tie. It is easy to see how this two robot case generalizes to robot team with n members, so HIVEMind robots are also immune to this problem. Again, communication lag can be a problem, but as before, in most cases the HIVEMind robots will generally converge to the same conclusion after several communication cycles.

5.4.4 Heterogeneity

The HIVEMind architecture's negotiation-free approach to multi-robot coordination does not imply that all team members have to possess identical inference networks or even that they necessarily have similar physical forms. HIVEMind makes two key assumptions about the robots' reasoning system:

- That the robots share common goals, i.e. there is no self-interest; hence openly sharing any and all beliefs is safe and acceptable.
- That the robots have a pre-determined method of deciding the actions to take in order to achieve those goals, as well as the team member that is responsible for executing a particular action.

Neither assumption imposes homogeneity upon the robots, either in terms of software control or physical design. Robots that have different physical designs can utilize HIVEMind to share beliefs about the world and the continually updated situational awareness allows them to avoid interfering with each other during task execution.

5.5 Handling oscillation

In Section 5.2.1, I discussed the way HIVEMind robots assign responsibilities for executing goals amongst themselves. Based on the current beliefs of each team member, the robots use a function to compute a reward value that determines the suitability of each robot for each goal. Then, these rewards values are used to find the optimal robot-goal pairings. These assignments are recomputed by the team on every process cycle, so

if new opportunities or contingencies arise, the team can quickly react to it by switching responsibilities if necessary.

However, this can also lead to thrashing between team members. For example, suppose two robots on a trash collection task both observe the same piece of garbage in front of them. Normally, the robot that is closest to the garbage would take the responsibility for collecting it. In this case, both robots are approximately the same distance from the piece of garbage (less than 2 cm difference) and their sensors have an error of ± 5 cm. During run-time, the distance measurement from each robot to the piece of garbage will vary slightly due to sensor noise. This could result in the team first believing one robot is closer and then the other, which causes the robots to switch responsibilities for completing the goal continuously, hence thrashing and doing no useful work between them. Obviously, this is suboptimal and undesirable behavior.

Of course, this is a well-known phenomenon in control systems design. Techniques for dampening or smoothing such oscillatory behavior are common in the control systems and robotics literature. In the example above, a simple solution would be to introduce some hysteresis into the system. When one of the robots takes the responsibility of collecting the piece of garbage, only a significant change in the situation (e.g. the other robot suddenly realizes that it is actually far closer to the garbage than originally thought or the first robot fails completely) for it to surrender its charge to another teammate.

Chapter 6

The Experimental System and Environment

I used a team of physical mobile robots to explore the issue of synchronizing joint beliefs across multiple robots in a cooperative group. Named after psychological phobias, Ochlo (fear of crowds), Kineso (fear of motion), Alektoro (fear of chickens) and Zemmi (fear of the great mole rat) reside on the third floor of the Northwestern University Computer Science Department building. For the experiments described in this report only two of the robots were used at a time, even though there are four available.

In this chapter, I shall describe the task, environment and basic structure of the robots. Section 6.1 describes the robot team's habitat and the tasks they perform. Section 6.2 describes the basic hardware components of the robots. Finally, Section 6.3 briefly describes the programming language used to implement the robot control programs.

6.1 Task and environment

The robots' habitat is a network of corridors in the west wing of the third floor of the C.S. Department. Each robot has a topological map of its environment; see Figure 6.1. The labels on the map in the figure correspond to either people whose offices are

nearby, or features of that location. Each labeled node represents a landmark, e.g. a T-intersection or corner, and edges between nodes are corridors. The corridors span an area approximately 6 meters by 20 meters with an aggregate path length of 50m. The robots can perform three tasks in this environment, all of which involve the systematic coordinated traversal of space.

Note that in chapter 7, some of the labels are different, and the *lounge* node does not exist in some cases, replaced instead by a connecting corridor between the *aaron* and *pinku* nodes.

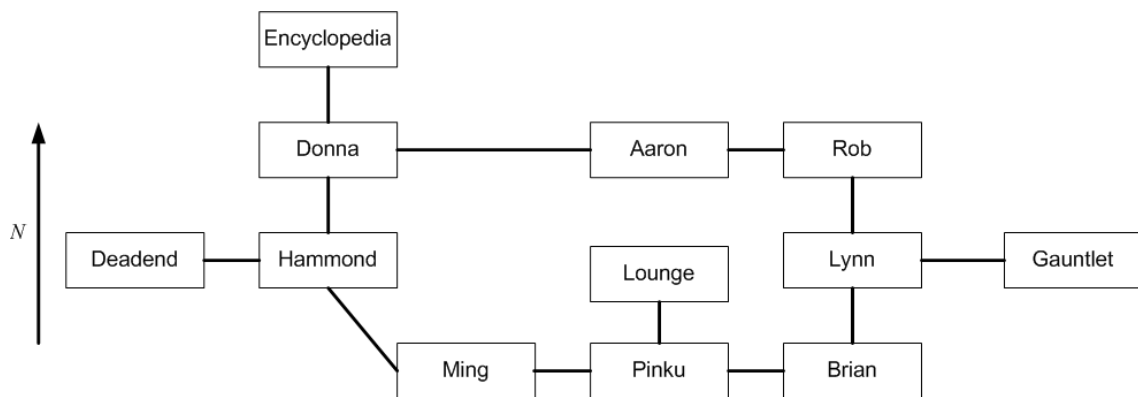


Figure 6.1: The topological map used by the robot team for navigation

6.1.1 Town Crier

The Town Crier task is the simplest of the three tasks that the robot team can perform. The two robots act as town criers who make announcements at each landmark on the map shown in Figure 6.1. Each landmark is visited at least once, and a verbal announcement is made there by the visiting robot. Both robots dynamically select

locations during execution of the task. If an announcement has already been made at a landmark, then the robots will no longer make an announcement there, even if a robot should happen to pass by that landmark again. There are two situations where a visited landmark could be revisited:

- The robot is traversing the visited landmark on its way to an unvisited one.
- The landmark recognition software misclassified the landmark on the previous visit, and the robot believes that this landmark has not yet been visited.

The town crier task is considered accomplished when announcements have been made at all the landmarks.

6.1.2 Find Static Object

In this task, the team cooperatively searches for a static object located somewhere in the operating environment. The robots explore the environment in a systematic manner until one of them locates the object or all searchable space is exhausted. When the object is found, both robots converge on its location.

The robots traverse the environment in a different way for this task compared with the previous town crier task. Merely visiting all the landmarks on the map is not sufficient; the object is more likely to be located in the hallways connecting the landmarks than at the landmarks themselves. Therefore, the robots systematically divide unexplored corridors during the execution of the task, rather than unvisited landmarks.

The objects that the robots are responsible for finding in this experiment are a set of brightly colored non-specular balls. One of these balls is placed somewhere in the network of corridors, and the human user is responsible for specifying the color of the ball as an argument to the command console. The robots can detect the balls by using a color blob tracking algorithm.

The find object task is considered accomplished when all team members have converged at the location of the object.

6.1.3 Capture Evading Target

The objective for this task is similar to the previous task, with one important exception: the target is now allowed to take evasive action. A human intruder is located somewhere within the network of corridors. The robots have to first search for the intruder, and then cooperatively maneuver into position to trap the target. There is only one exit point in this environment --- the *gauntlet* node. If the intruder manages to reach that location, then she would be considered victorious. The robots obviously operate at a physical and mental disadvantage compared to the intruder. To even the odds a little, I defined some basic rules of fairness:

1. The intruder cannot jump over the robots. The robots cannot help being short.

Furthermore, one could easily imagine a larger robot performing this task, which would prevent the human from dodging the trap in this manner.

2. The intruder must stay in the corridors at all times. Specifically, the human is not allowed to climb objects like tables to avoid being seen. Again, the robots cannot help being short and limited in their range of view.
3. The human is not allowed to damage the robots. Throwing things at the robot to disable it is not fair unless the robot is allowed to fight back with stun guns.

The rules of this game are obviously stacked in favor of the robots; essentially, for the human to win, the robots have to make mistakes and maneuver into bad positions, allowing the intruder to get around them. However, given the inherent limitations of our current robots, I believe this is a fair tradeoff.

The execution of this task is divided into four phases:

- First, the robots start out in *search* mode. This is similar to the find object task; the robots systematically explore all the corridors until one of them locates the intruder.
- The robot that sees the intruder then transitions into *sentry* mode. This robot remains stationary in its position while taunting with the human with the spoken phrase “Do not attempt to run, human intruder!”, which serves to indicate the current mode of the robot. The sentry will not follow should the human attempt to run away, but instead remains stationary and changes its taunt to “You can run, but you can’t hide!”
- In the meantime, the other robot is the *stalker*. Knowing the location of the sentry, the stalker selects a path that will bring it to the opposite end of the

corridor where the sentry is currently located. That is, the robots are attempting to cover both ends of the corridor, trapping the intruder in between them. If the human runs away from the sentry and encounters the stalker, then the robots will switch responsibilities. The stalker becomes the sentry and remains stationary, while the sentry takes on the stalker role and moves to intercept the intruder.

- When both robots have the intruder in sight, they close in and *trap* the human between them. At this point, the taunting switches to “We have you now, human scum”.

The *sentry* and *stalker* modes serve to emphasize the interaction between the two robots. The environment in which the robots reside is fairly limited; in fact, if the robots are started from the *gauntlet* landmark, then they could likely trap the intruder by simply going down the two long parallel corridors that form the majority of the space. The two modes show that the robots are not merely executing this simple strategy, but are in fact exchanging information and responsibilities dynamically as the situation changes.

As with the find object task, the robots locate their target through the use of color. This will be facilitated through the use of brightly colored non-specular pants worn by the human playing the role of the intruder. A human user is again responsible for providing the target’s color as an argument for the task to the command console.

The capture task is considered done when the human intruder has been trapped between the robots and is unable to move any further.

6.2 Hardware

Ochlo, Kineso, Alektoro and Zemmi are based on a commercial robotic platform, the Real World Interface (now iRobot) Magellan Compact Mobile Robot [RWI 1999], with some in-house modifications performed by other members of the Autonomous Mobile Robot Group (AMRG). The Magellan platforms are controlled through the use of laptops mounted on them. In Figure 6.2, Ochlo is shown in side profile, while Alektoro is shown from the front.



Figure 6.2: Ochlo and Alektoro shown in side and front profiles respectively

It is important to note that, while the robots are ostensibly duplicates of each other, there is significant variation in the sensitivity and accuracy of their sensors and effectors.

6.2.1 Mechanical Design

The Magellan robot platform is a small wheeled vehicle measuring 14.5 inches in diameter and 7.8 inches high. It weighs 29 lbs. and has two 4.5 inch rubber tires that can be driven differentially at speeds up to 2.5m/s. Caster wheels in the front and rear provide stability. Two onboard rechargeable batteries provide 12 AH capacity at 12 volts; in practice, this allows greater than 8 hours of continuous operation.

An additional structure was added on top of the original Magellan platforms to allow us to attach laptops onto them. This consists of two square plexiglass plates held together at a right angle by an L-bracket, and then attached to the Magellan by four steel support columns. Foam padding was added on the plexiglass plates for cushioning, and Velcro strips were applied to the foam. The Velcro strips, three in all, sufficed to hold the laptop firmly against the foam padding.

The laptops used on the Magellan platforms are Dell Latitude CPx models, each with Pentium III 500 MHz processors, 384 Mb of RAM, Gb of hard drive space, and running Windows98 Second Edition. Each laptop also has an Orinoco Silver Card that provides an 11Mbps data transfer rate for wireless communication. The laptop communicates with the base through the use of a standard 9-pin serial port.

6.2.2 Sensors

The Magellan platform is equipped with two types of sensors. Sixteen sonar sensors and sixteen tactile sensors are arranged in a ring around the circular base; each perimeter

panel has one sonar sensor and one tactile sensor. For the purposes of my experiments, none of the tactile sensors were used, and only the front three sonar sensors were utilized as a backup for the visual navigation system. A third sensory modality, infrared sensors, is present but non-functional on the Magellan platforms.

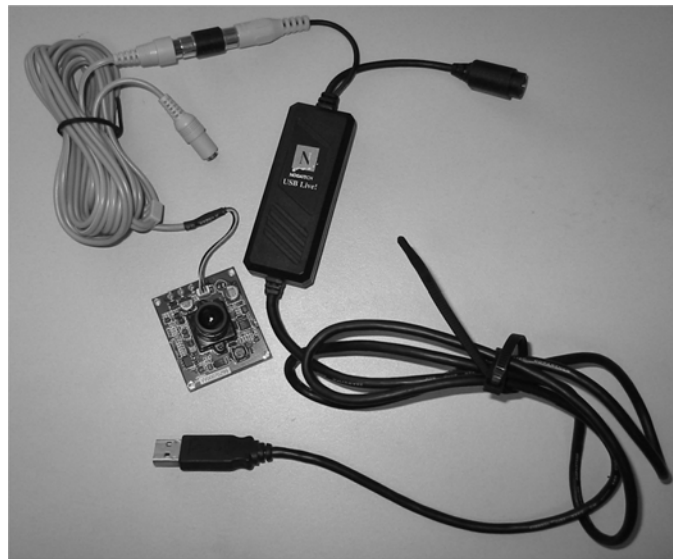


Figure 6.3: ProVideo Camera attached to the Nogatech framegrabber

The primary sensor used on the robots is a ProVideo CVC-514BC ¼” Color Board Camera mounted at a slight downward angle on the vertical plexiglass plate (see Figure 6.2). The camera is connected to a Nogatech NV 2000 framegrabber. Figure 6.3 shows an unmounted camera attached to a framegrabber. The framegrabber is connected to the laptop through a standard USB connection.

6.3 The GRL Programming Language

Most of the software for the experiments described in this chapter is written in Generic Robot Language (GRL) [Horswill 1999], which is a high-level language for designing behavior-based systems.

6.3.1 Primitive signals

GRL is an architecture neutral language embedded within Scheme and its programs consist of signals that are ostensibly computed in parallel and continuously updated. In practice, these programs are generally run on serial uniprocessors. Therefore, signals must be compiled down into a form that can be easily implemented in a language like Scheme or C. Primitive signals can be:

- A *constant* value.
- A *source signal*, for which the programmer has provided raw Scheme code.
- An application of a *primitive procedure* (e.g. +, -, log) to a set of signals. Procedures map signal values to values. A value can be scalar (integer, float, Boolean, etc) or a vector.
- An application of a *finite-state transducer* on a set of signals. The programmer is again responsible for the Scheme code for computing the output of the transducer from the current values of the input signals. Since transducers may contain state information, they are mappings from signal time histories to time histories.

Sources and transducers are the only ways a programmer may directly utilize raw Scheme code. However, the Scheme code is restricted to be statically typed and may not dynamically allocate memory.

Signals allow a programmer to write real-time control loops in a straightforward manner. For example, suppose the robot is looking for an open doorway using a directional sensor such as sonar or infrared while driving down a corridor. One approach could be to utilize the readings from the sideways-facing sensors:

```
(define-signal doorway?
  (> (true-time (> (max left-reading right-reading)
                  distance-threshold))
     time-threshold))
```

true-time is a transducer that returns the number of milliseconds for which its input signal has been true, and is a part of the standard GRL library. The use of this transducer prevents misidentification of doorways due to occasional sensor errors; it ensures that the robot will believe that a doorway exists only if the sensor reading has been true for a sufficient amount of time.

6.3.2 Compile-time abstractions

GRL supports compile-time procedural abstractions in the form of *signal procedures*. A signal procedure is a mapping from signal networks to signal networks. During

compilation, signal procedures are calls to signal procedures are expanded into networks of primitive signals.

GRL supports record-like data abstractions called groups. Records are generally defined using the *define-group-type* macro which defines constructor and accessor functions for the data type. For example, an *rt-vector* is defined as:

```
(define-group-type rt-vector
  (rt-vector rotation translation)
  (rotation rotation-of)
  (translation translation-of))
```

The *rt-vector* has two components, a rotational and a translation velocity, which are accessed through *rotation-of* and *translation-of* respectively. New *rt-vectors* can be created though the *rt-vector* function, which takes rotation and translation arguments.

6.3.3 Behaviors

Another commonly used data type is a *behavior*, which is represented as a group consisting of a motor-vector (what the behavior wants to do) and an activation-level (how badly it wants to do it).

```
(define-group-type behavior
  (behavior activation-level motor-vector)
  (activation-level activation-level)
  (motor-vector motor-vector))
```


Activation levels can be Booleans (i.e. on or off) or scalars (generally between 0 and 1). For the experiments described in this report, a motor-vector is actually an rt-vector representing the desired rotational and translational velocities of the robot.

The GRL library contains implementations of several popular behavior-based programming mechanisms. This allows the programmer to mix-and-match mechanisms as necessary, and also makes it convenient to experiment with variant and hybrid arbitration mechanisms.

As an example, the Motor Schema system has behaviors that generate motor-vectors that are combined through weighted summation. To model Motor Schemas, we can define a *weighted-motor-vector* using the activation-level of the behavior as a weight:

```
(define-signal (weighted-motor-vector beh)
  (* (activation-level beh)
     (motor-vector beh)))
```

The weighted sum of behaviors can then be defined as:

```
(define-signal (weighted-sum . behaviors)
  (apply + (weighted-motor-vector behaviors)))
```

The *weighted-motor-vector* signal procedure is implicitly mapped across the list of behaviors in the above example. Assuming we had defined two behaviors: *move-toward-destination* and *avoid-obstacle*. Then the final motor output could be computed as:

```
(define-signal motor-output
  (weighted-sum move-toward-goal
                avoid-obstacles))
```

Now suppose that we would prefer to use a weight average rather a weight sum of the behaviors. The following definitions would allow us to do so:

```
(define-signal (weighted-average . behaviors)
  (/ (apply weighted-sum behaviors)
     (apply + (activation-level behaviors))))
```

```
(define-signal motor-output
  (weighted-average move-toward-goal
                    avoid-obstacles))
```

6.3.4 Sequencing

GRL programs are typically a collection of signals running in parallel. However, at times, it is useful to have sequential control. *Plans* are a construct for compiling serial expressions into sequences implemented as GRL transducers, which means that they are finite state machines internally. Therefore none of the plan sequencers have a stack, so it is not possible to write recursive plans. Plans expressions are primarily responsible for starting or stopping behaviors (by switching their activation levels on or off) and other

plans. Other expressions include blocking expressions that temporarily halt the execution of the plan, or conditional expressions such as *if-then* or *while*.

The following plan first drives the robot down a hallway by activating the follow-corridor behavior, then turns it sharply leftward when the robot encounters a wall:

```
(define-plan (straight-and-left)
  (start follow-corridor)
  (wait wall-in-front?)
  (stop follow-corridor)
  (ballistic-left-turn))
```

6.3.5 Pools

Pools are representational modalities that contain items that may be conveniently bound to role variables. A pool is created using the signal-procedure **make-pool**; for example, the color pool is defined as:

```
(define-signal color-pool
  (make-pool `color-pool
    (vector 'green 'red 'blue)))
```

The first argument to *make-pool* is the name of the pool, and the second is a vector representing the items in the pool. In this case, the pool is being used as a memory

system that stores items in a set of registers. Items in the pool may be bound to role variables; e.g. the color *green* could be bound to the *object* role:

```
(bind-item! color-pool `green (roleset object))
```

roleset is a macro that returns a bit mask representing the set of roles passed to it as arguments; so, *bind-item!* may actually bind an item to multiple role variables simultaneously. Conversely, the item that is currently bound to a given role may be accessed by using *pool-lookup*:

```
(pool-lookup color-pool object #f)
```

The above signal-procedure call returns the present color bound to the *object* role; the last argument to *pool-lookup* (*#f* in the above example) is a default value that is returned if no binding is found.

Pools can also represent a group of sensory systems; for example, the tracker pool is defined as:

```
(define-signal tracker-pool
  (make-pool 'tracker-pool tracker-count))
```

where *tracker-count* is an integer representing the number of trackers in the pool. When a pool contains sensory systems, the items in the pool are collections of functionally equivalent sensory processes that can be dynamically allocated to different tasks, i.e. each tracker in the tracker pool may be dynamically assigned to track different objects during run-time:

```
(%pool-allocate->slot! tracker-pool current-context object)
```

%pool-allocate->slot! allocates an item slot in the pool to the role variable and returns the slot index; a free slot is allocated if one is available, otherwise the least recently used slot is allocated. Information from the sensory systems may be accessed using *pool-predicate* or *pool-function*:

```
(pool-predicate pool vector)
```

```
(pool-function pool scalar-vector default)
```

The former takes a pool and a Boolean vector of length equal to the items; it returns a roleset (bit mask) of the roles bound to items whose corresponding entries in the Boolean vector are true. The latter takes a pool, a vector of scalars and a scalar as arguments and returns a vector of length equal to the number of role variables. The *i*th element of this vector is the respective value from *scalar-vector* of whichever item in *pool* is bound to the *i*th role; if no item is bound to that role, then the element is *default*. So, if the *object* role variable is represented as the first role in the roleset and is bound to the second item in the pool, then the first element of the returned vector contains the second element in *scalar-vector*.

6.3.6 The GRL Compiler

The GRL compiler is always called with a list of top-level signals to compile. It recursively evaluates the compile-time constructs such as signal procedures until it has reduced the program to a graph of primitive signals. A topological sort is performed over the signal graph to determine the total ordering of primitive signals that drive the

original signals passed to the compiler. This ordering is used to decide the order of evaluation in the object code. GRL attempts to optimize the signal graph through standard optimizations such as inlining, algebraic simplification, constant folding and hoisting of loop invariants. Occasionally, there are cycles in the signal graph, and these are “broken” by inserting a unit delay before one of the signals in the cycle.

GRL may generate output code in various languages, including Scheme, C, C++, BASIC, and UnrealScript². For the experiments described here, the GRL code was compiled into Scheme.

6.3.7 Calling programs external to GRL

Given the lack of pointer semantics in GRL, it is difficult to perform certain computations. For example, the path planning code for the capture evading target task requires the use of a queue or of spreading activation. Fortunately, calling native Scheme code from within GRL is fairly straightforward; for more information on this, refer to [Horswill 2000]. Therefore, parts of the code used in the experiment (including the topological map, the path planner, and some of the team communication code) were written directly in native Scheme instead of GRL. The specifics of the implementation are discussed in greater detail in the next chapter.

² UnrealScript is an interpreted scripting language for a popular computer game.

Chapter 7

Design of A Robot Using HIVEMind

This chapter describes the experimental system used as a proof of concept for the HIVEMind architecture. This system was used in the execution of all three tasks described in Chapter 6: Town Crier, Find Object and Capture Evading Target. Figure 7.1 shows the high-level view of the entire system.

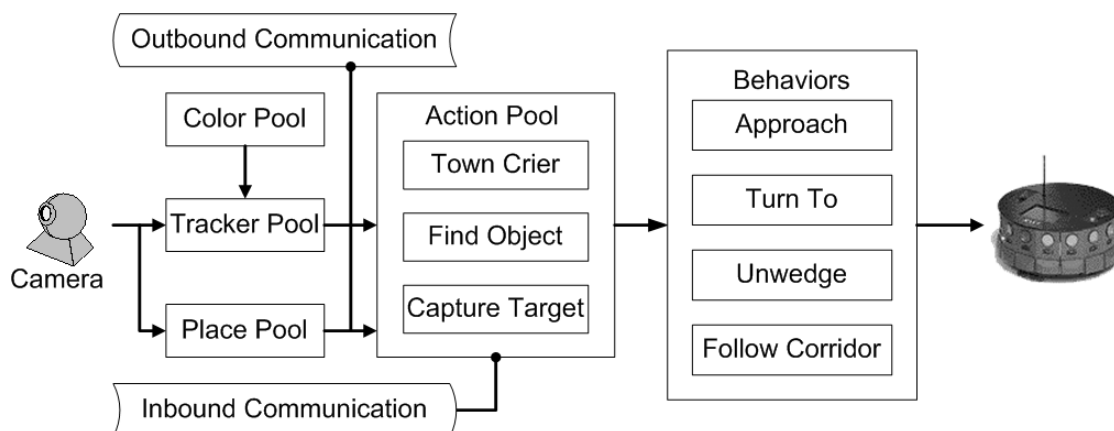


Figure 7.1: High-level layout of the experimental system

During startup, none of the actions in the action pool are active, so the robots do nothing. The user can assign a task through the use of a command console (see Section 7.6). The current task, as well as any arguments for it, is communicated to the other team members and the appropriate action is activated. During the execution of the task, the incoming video stream is processed by low-level visual routines written in C++, and

then passed to high-level procedures in the tracker or place pools. The active action may bind roles to items in the tracker or place pools. In return, information from one or both of these pools is passed back to the action, and is also communicated to the other team members. Percepts from the local sensors are merged with percepts from other team members before being processed by the current action. Finally, the active action drives the base by turning the four control behaviors on and off appropriately.

7.1 Color Pool

The color pool stores information about the colors that the tracker system (see Section 7.2) can detect. There are currently three colors in the pool: red, green and blue. Each color is represented as an intensity-normalized YUV tuple:

$$Color = \langle NU, NV \rangle \text{ where}$$

$$NU, NV = \langle mean, stdev, min_mean, max_mean, min_stdev, max_stdev \rangle$$

NU and NV are the normalized color components. The tuples contained in the color pool represent the initial values for each color. The color trackers utilize these values to acquire the object to be tracked, or reacquire the object, if it is lost.

There is enough variance across cameras on different robots that the resulting initial color values are fairly disparate. The table below shows a typical set of color YUV values for a robot.

	<u>NU</u>	<u>NV</u>
Red	<69.0, 15.0, 49.0, 89.0, 5.0, 20.0>	<38.0, 15.0, 18.0, 58.0, 5.0, 20.0>
Green	<-27.8, 15.0, -42.8, -12.8, 5.0, 20.0>	<24.6, 15.0, 9.6, 39.6, 5.0, 20.0>
Blue	<-60.0, 15.0, -105.0, -15.0, 5.0, 20.0>	<100.0, 15.0, 55.0, 145.0, 5.0, 20.0>

Table 7-1: Typical initial intensity-normalized YUV values

7.2 Tracker Pool

The adaptive color tracker system [Depristo 1999] is based on a limited N-Clusters algorithm. The tracker operates on intensity-normalized YUV, or NUV, images. Each pixel P in the image is viewed as a four-tuple, $P = \langle X, Y, NU, NV \rangle$, where X and Y are the position of the pixel in the image, and NU and NV are as defined in Section 7.1. The tracker partitions the set of pixels in the image into a collection of disjoint clusters through the use of a similarity criterion. In this case, the similarity criterion is based on a statistical measure; that is, each cluster is treated as a statistical distribution, with a mean and standard deviation. A pixel P is in a cluster C if, for each dimension D in $\langle X, Y, NU, NV \rangle$:

$$\left(\frac{D_P - \text{mean}(D_C)}{\text{stdev}(D_C)} \right) \leq D_{\text{Threshold}}$$

This similarity criterion ensures that pixels that are located close to each other, and are of a similar color, will tend to end up in the same cluster, i.e. the set of pixels in the

image is partitioned into clusters that are localized in both space and color. Observe that such clusters will tend to correspond to uniformly colored objects in the world.

The color tracker requires a set of mean and standard deviation values to bootstrap the tracking process. This is the responsibility of the color pool described in Section 7.1. During runtime, an active action may call on the tracker pool to track a target represented by a role; the target's color is bound to that role. The tracker pool selects an available tracker and assigns it to the role. This tracker then uses the role to index the mean and standard deviation values of the target color in the color pool.

Since the target may move during active tracking, the X, Y positions of the cluster could change, and different lighting conditions might affect the NU, NV values. The color tracker compensates for this by recomputing the mean and standard deviation for all four dimensions of the cluster on every cycle. That is, after the tracker computes the current set of pixels in the cluster, it calculates the mean and standard deviation of the $\langle X, Y, NU, NV \rangle$ dimensions for these pixels, and uses these new values to determine the next set of pixels in the cluster on the following processing cycle.

Percept	Information provided
see-object?	Predicate indicating if the target is seen.
near-object?	Predicate indicating if robot is sufficiently close to the target.
x-coords	x-position of target in the robot's field of view.
y-coords	y-position of target in the robot's field of view.

Table 7-2: Percepts provided by the tracker pool

The tracker pool provides an interface for binding a role to an available tracker, and conversely for checking if a role is currently bound to a tracker. In addition, the tracker pool also provides the percepts shown in Table 7-2 that the active action may use to obtain information about targets being tracked. The two predicates, *see-object?* and *near-object?*, are represented as bit-vectors, where each bit represents one of the thirty-two roles. A set bit indicates that the predicate is true for the corresponding role. *see-object?* is true when the number of pixels in the relevant cluster is sufficiently large, while *near-object?* is true when the robot sees the object and it is centered in the robot's field of view. *x-coords* and *y-coords* are both integer arrays of size thirty-two. Again, each slot in the array represents one of the roles.

The left image in Figure 7.2 shows a small green ball in the robot's camera view. The right image shows the pixels assigned to the cluster that represents the ball by the color tracker.

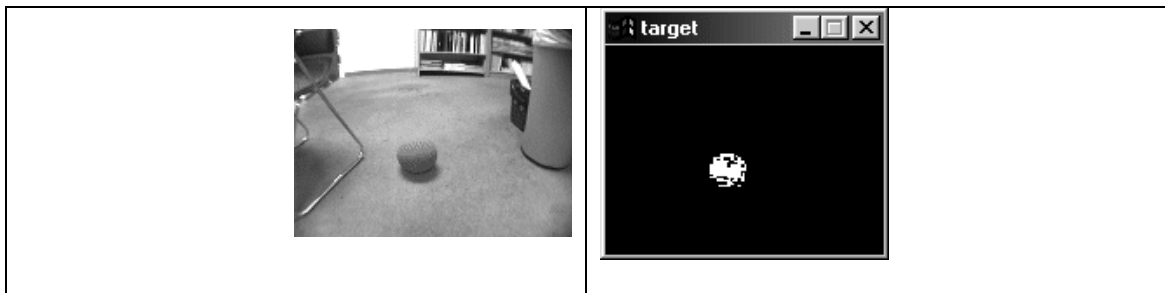


Figure 7.2: Camera image containing target object and silhouette of pixels in the representative cluster.

7.3 Place Pool

The place pool holds information on specific locations in the robots' habitat. These locations are landmarks on the map that the robots utilize for navigation. The place pool contains the name and index number of each landmark. A role may be bound to one or more landmarks simultaneously.

In the following subsections, I will discuss the internal format for the topological map. Then I will describe the probabilistic localization technique used to determine the current position of the robots. Finally, I will describe the algorithm that the robots use to search the area represented by the map.

7.3.1 Map Format

The map used by the robots is a topological one, i.e. it consists of a set of nodes or landmarks, connected by a set of edges representing hallways. A landmark is defined as an intersection (a corner, t-junction, etc) or a dead end.

Corridors can extend from landmark only in eight possible directions set at 45 degree intervals. This simplifies the map representation and the feature detectors required to find the landmarks. Fortunately, most office environments have rectilinear corridors that fit this straightforward design.

The system finds landmarks through the use of three simple feature detectors: **delayed-see-wall-ahead?**, **delayed-see-left-wall?**, and **delayed-see-right-wall?**. The

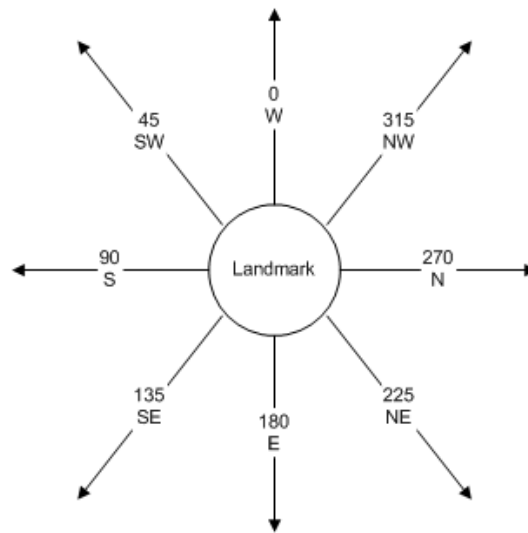


Figure 7.3: Possible corridor outlets from a landmark node

“delayed” in the name of each refers to the fact that each signal is delayed by a constant amount of time. The reason for this is that the cameras on the robots are angled high enough that features are triggered before the robot is physically at the correct location, and conversely when the robot actually arrives at the correct place, the feature can no longer be seen. Hence, the signals are delayed by a small time amount (approximately 1-2 seconds) to allow the robot to trigger feature detection only when it is actually at the correct location. Using these three feature detectors, a corridor is defined as:

```
(define-signal corridor?
  (and (not delayed-see-wall-ahead?)
        delayed-see-left-wall?
        delayed-see-right-wall?))
```

In turn, a landmark is defined as anything that is not a corridor:

```
(define-signal landmark?  
  (> (true-time (not corridor?))  
     min-landmark-seen-time))
```

true-time is a signal-procedure that returns the number of milliseconds for which its input argument has been true. The use of **true-time** and **min-landmark-seen-time** helps to eliminate noise in the vision input and ensure only real landmarks are identified.

The map used by the robots in the execution of all their tasks is that of the west wing of the third floor of 1890 Maple Avenue in Evanston, where the Northwestern University Computer Science Department is located. There are eleven landmarks on this map, as shown in Figure 7.1. Furthermore, as Figure 7.3 shows, the westward direction was arbitrarily chosen as 0 degrees in absolute world coordinates. During startup, the robots are assumed to be facing directly west or 0 degrees. This is due to the fact that the landmark recognition system needs to have an absolute world orientation in order to reference the map correctly. Since the robots begin with a “clean slate” each time they are started, they require some human assistance to gain their bearings. I was not attempting to solve the kidnapped robot problem, and therefore considered this an acceptable solution.

7.3.2 Probabilistic Localization

The robots utilize a probabilistic method to perform localization, the act of determining one's current location in the environment. The localization technique used is similar to the one described in [Thrun 2000]; however, the type of map I used is topographical rather than metric³.

The probabilistic approach to localization views the problem as a density estimation one, i.e. the robot is attempting to estimate a posterior distribution over the space of its possible locations conditioned on available data. A location in this context is a landmark on the topological map. Denoting the robot's location by s_t and the data leading up to time t by $d_{0..t}$, the distribution $b_t(s_t)$ can be expressed as:

$$b_t(s_t) = p(s_t | d_{0..t}, m)$$

where m is the map of the environment.

There are two types of available sensory data: observations and action data. The former characterizes the current, momentary situation (e.g. camera images), while the latter relates to the change of the situation (e.g. odometer readings). Assuming without loss of generality that both types of data arrive in an alternating sequence, then:

$$d_{0..t} = o_1 a_1 o_2 a_2 \dots \dots o_t a_t$$

³ A metric map divides the environment into a grid of evenly spaced cells. In general, metric maps are more precise and easier to construct but more expensive in terms of storage and computation.

Then, starting from this expression,

$$b_t(s_t) = p(s_t | o_1 a_1 \dots \dots o_t a_t, m)$$

we can apply Bayes rule and the theorem of total probability, and exploit the Markov assumption twice to obtain:

$$b_t(s_t) = \eta_t p(o_t | s_t, m) \sum_{s_{t-1} \in \mathcal{S}} p(s_t | a_{t-1}, s_{t-1}, m) b_{t-1}(s_{t-1})$$

where η_t is a constant normalizer which ensures the result sums up to 1. The Markov assumption states that the past is independent of the future given knowledge of the current state. Practical experience has shown that probabilistic algorithms are tolerant of mild violations of this assumption.

This approach is generally known as Markov localization [Burgard et al. 1996] [Fox, Burgard and Thrun 1999] [Kaelbling, Cassandra and Kurien 1996]. However, it equally represents the update equation in Kalman filters [Kalman 1960], Hidden Markov Models [Rabiner and Huang 1986], and dynamic belief networks [Dean and Kanazawa 1989].

To implement the above equation, we need to specify $p(s_t | a_{t-1}, s_{t-1}, m)$ and $p(o_t | s_t, m)$. The first density may be thought of as a probabilistic generalization of the mobile robot's transition to a landmark (represented as a state) given some motion from previous landmark. The second density is a probabilistic model of perception. In this system, $p(o_t | s_t, m)$ is calculated using the three feature detectors described in Section 7.3.1; this density relates the perceived walls (or lack therefore) to the expected features

as indicated by the topological map, under the assumption that the robot is located at s . The distribution $b(s_t)$ is updated each time the robot believes it has arrived at a landmark, and is computed in the following manner for this system:

Let M be an $n \times n$ matrix of vector distances between landmarks, where n is the total number of landmarks. Create a transition matrix M' by taking each element m in M and computing:

$$m' = e^{-\left(\frac{k-m}{\sigma^2}\right)}$$

where k is the magnitude of the measured motion of the robot, and σ is the standard deviation of the assumed Gaussian error; this essentially weights the probability for each landmark in terms of the total distance traveled since the previous landmark. Let V be a vector of size n . For each v_j in V :

$$v_j = w_{1j} w_{2j} w_{3j}$$

where

$$w_{ij} = \begin{cases} \text{if } f_i \text{ and } r_{ij}, \text{ then } 1 - x \\ \text{if } f_i \text{ and not } r_{ij}, \text{ then } 1 - y \\ \text{if not } f_i \text{ and } r_{ij}, \text{ then } x \\ \text{if not } f_i \text{ and not } r_{ij}, \text{ then } y \end{cases}$$

f_i is the boolean valued output from the feature detector i .

r_{ij} is the expected feature from feature detector i at landmark j .

x is the probability of making a false negative observation (i.e. not seeing a wall where one exists).

y is the probability of making a false positive observation (i.e. seeing a wall where none exists).

V represents the probability of the robot observing a given set of features at a landmark given a priori knowledge of the actual features that that landmark; in this case, the features being walls in front, to the left of and to the right of the robot. Putting this all together, $b(s_t)$ is computed using the following expression:

$$b(s_t) = \eta(b(s_{t-1})M')V$$

$b(s_t)$ is computed by first multiplying the previous distribution $b(s_{t-1})$ by the transition matrix M' , and then an element-wise product is done over the resulting vector and the evidence vector V . Finally, the normalizer η is applied to ensure the distribution sums up to 1.

For actual navigation purposes, the robot has to commit to a particular landmark each time it comes to one. Therefore, whenever the robot arrives at a landmark, it computes $b(s_t)$ and decides that the landmark with the highest probability value in that distribution is the one where it is presently located.

7.3.3 Landmark-to-Landmark Navigation

During normal navigation, the robots do not utilize an online path planner (the exception to this is discussed in Section 7.3.4) Instead, all landmark-to-landmark paths are stored in a lookup table that is generated a priori from the topological map. This code was primarily written in Scheme, with some interface functions to GRL.

Each landmark in turn is used as the root of a tree, and a breadth-first walk is performed over all other landmarks, ignoring cycles. When an unvisited landmark is encountered during the breadth-first walk, its parent is checked. If the parent is the root, then the direction (represented as degrees in absolute world terms) from the root to that landmark is recorded in the lookup table. If the parent is not the root, then the parent's directional entry in the lookup table is propagated to the current landmark. As an example, the breadth-first walk in Figure 7.4 uses landmark 1 ("Lynn") as the root. The number on the links between nodes indicates the direction the robot should turn towards to reach that landmark. For example, if the robot is currently at Lynn and wishes to travel to the Lounge, it should turn to face 90 degrees in absolute world terms (which would be southward). Notice that all nodes share the same direction as their parent, with the exception of the root node.

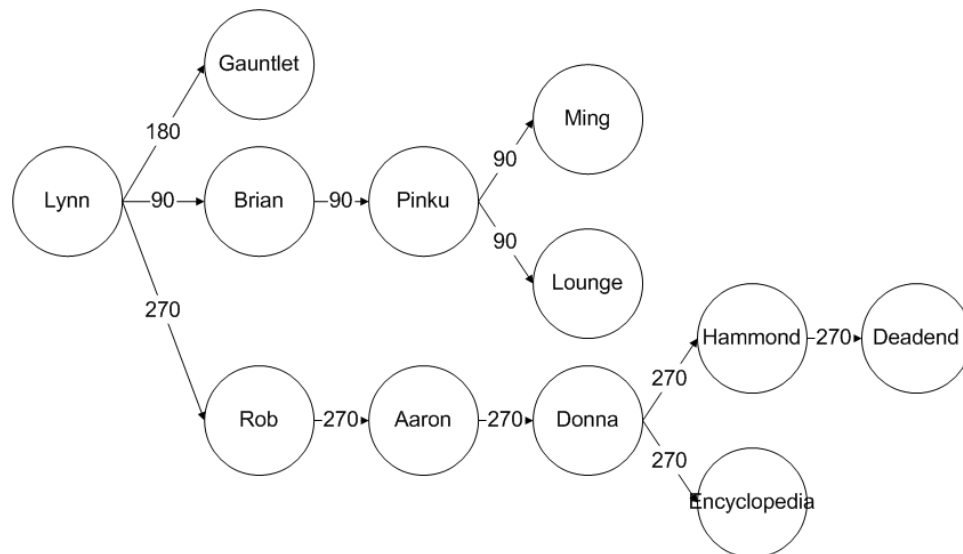


Figure 7.4: Breadth-first walk using landmark 1 as the root node

Robots are assumed to start out at a known landmark during navigation. The robot indexes the table using the current landmark and its intended destination. The table entry indicates the direction the robot should turn to. After turning appropriately, the robot moves forward until it reaches another landmark, whereupon it localizes to determine its current location. If this landmark is its original destination, then it is done. Otherwise, it indexes the table again with the destination and this new landmark as the source, and repeats the process.

It is easy to see that determining the next landmark given the current landmark and ultimate destination is an $\Theta(1)$ time operation. Storing the lookup table requires $O(N^2)$ space in the number of landmarks; this is not a significant problem because total landmarks in the test environment is very small. The total running time for a breadth-first traversal is $O(N+E)$ time in the number of landmarks (N) and corridors (E). Hence, creating the lookup table takes $O(N^2+NE)$ time. While expensive, this lookup table only needs to be generated once.

7.3.4 Navigating around blocked landmarks

For the Capture Evading Target task, the robots have to trap the human between each other. Suppose the human is observed by Robot1 as it is traveling from landmark A to landmark B. Robot2 has to maneuver to landmark B in order to trap the human in that corridor. However, since the robots all shared the same navigation paths, Robot2 will attempt to reach landmark B by going through landmark A; obviously, this is not the

appropriate approach. Robot2 should realize that Robot1 is already at landmark A, and it should seek an alternative path to landmark B.

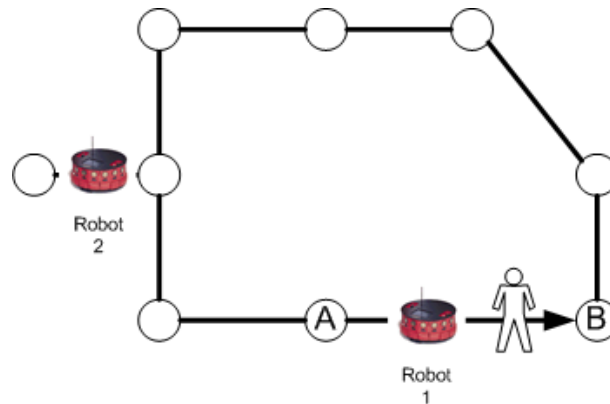


Figure 7.5: Robot1 spots the intruder as it goes from A to B

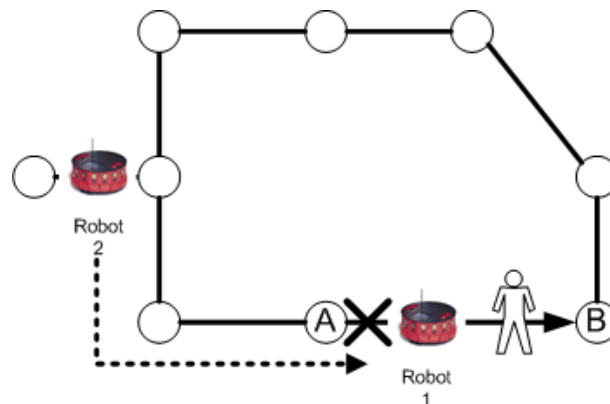


Figure 7.6: The normal path that Robot2 would take to B is blocked by Robot1

This is implemented by creating a lookup vector for Robot2 dynamically when it realizes that it is in the situation described above. Using its current position as the root, a breadth-first walk is again performed over all other landmarks. The lookup vector that is generated has two significant differences from before:

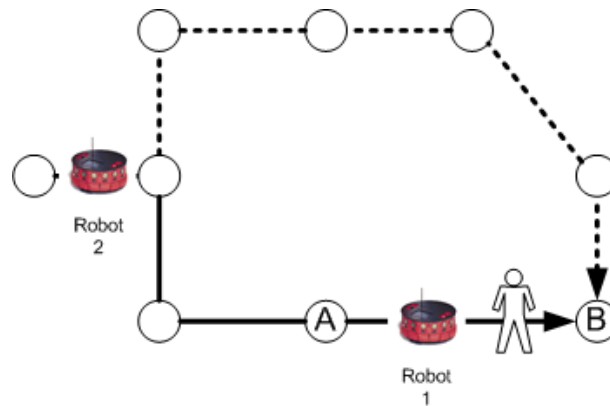


Figure 7.7: Robot2 takes an alternative path to B in order to trap the intruder

- The landmark where its teammate is located is now marked as “blocked”, and the links to it are removed. Therefore the breadth-first walk of the tree will not go through it.
- Rather than storing the direction to face for the next landmark, the lookup vector simply stores the next landmark itself. The system then utilizes the original lookup table to determine the direction.

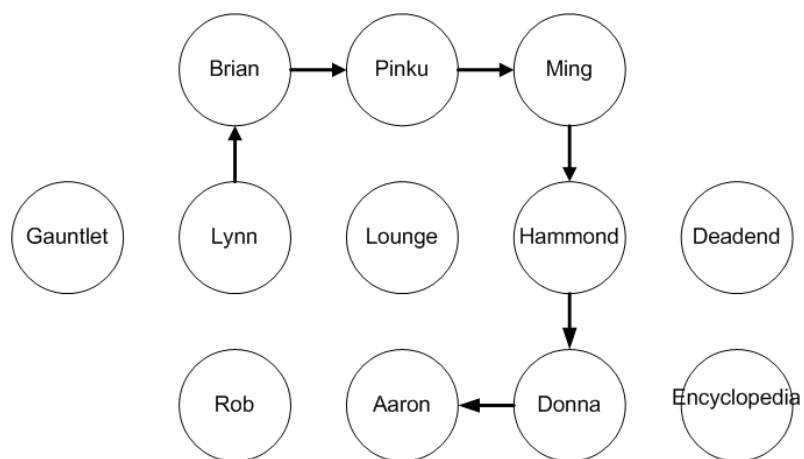


Figure 7.8: Visual representation of lookup vector

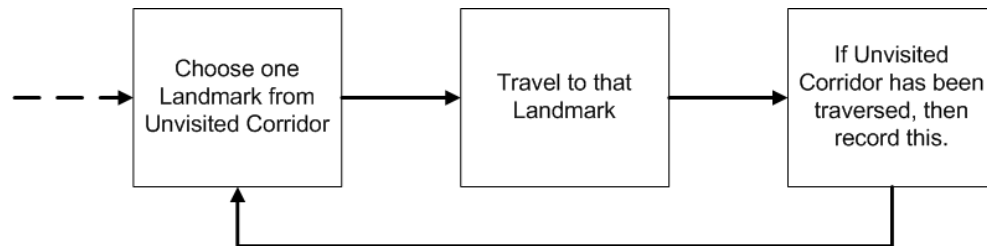
The figure above shows the lookup vector that would be generated if the current robot is at Lynn, and its teammate spotted the human target while traveling from Rob to Aaron. Only the path from Lynn to Aaron is stored in the lookup vector. All nodes with no outward links point back to themselves; if the robot wanders into one of these nodes during its journey, it will have to replan the path using that node as root.

The code for this was also written in Scheme, with some procedures provided as interface functions. The GRL code uses these interface functions to connect to this system. As before, it is an $\Theta(1)$ time operation to determine the next landmark given the current landmark and ultimate destination is. Populating the lookup vector is a breadth-first traversal operation that takes $O(N+E)$ time in the number of landmarks (N) and corridors (E). However, this is not a procedure that is executed frequently, and the number of landmarks and corridors on the map is very low.

7.3.5 Searching the Map

Searching the environment is not equivalent to simply traversing all the landmarks. The links joining landmark nodes represent real physical space where an object could be located. Merely visiting landmarks could leave some corridors unsearched. Given the vast proportion of corridor space compared to landmark space, any target object is more likely to be in a corridor than at a landmark. Therefore, the search system records unvisited corridors, rather than unvisited nodes. However, the navigation system

expects landmarks as input, not corridors; so, the search system has to take this into account. On an abstract level, the search procedure is a three-step process:



The first step (choosing a landmark) is accomplished in the following manner:

1. Check the list of unvisited corridors. Each corridor is assumed to have two landmarks anchoring it.
2. If the list is empty, then all corridors have been searched.
3. If there is an unvisited corridor with the current landmark as an anchor, then return the other anchoring landmark; the robot can simply turn to the appropriate direction and travel down that corridor to search it.
4. If all nearby corridors have been searched, then get the list of nodes neighboring the current landmark. For each node in the neighbor list that has not been previously examined, push that node onto a queue. Since the map is a graph and not a tree, it is possible for loops to occur. To prevent the algorithm from going into an infinite loop, nodes that have already been examined are not put back onto the queue.
5. Remove the first node from the queue. Check the unvisited corridor list to see if there exists an unvisited corridor with this node/landmark as an anchor. If so, return this node. Otherwise, go back to step 3 using this node as the 'current landmark'.

The second step (traveling to the landmark) uses the navigation system described in Section 7.3.3. This step terminates whenever the robot reaches any landmark, i.e. the robot does not have to actually reach the targeted landmark. At this point, the third step (recording searched corridors) runs:

1. Suppose the robot started out at Landmark1, and is currently at Landmark2.
2. If there is a corridor anchored by both Landmark1 and Landmark2, then mark it as searched.
3. Otherwise, check to see if there exists Landmark3 such that (Landmark1, Landmark3) and (Landmark3, Landmark2) are both corridors. If so, then assume that the localization system somehow missed identifying Landmark3. Mark both (Landmark1, Landmark3) and (Landmark3, Landmark2) corridors as searched. Of course, this is a heuristic approach; however, practical experience has shown this to be useful.
4. If neither of the above two cases hold, then do not mark any corridors as searched.

7.4 Action Pool

The action pool consists of three reflected plans, one for each task the robots can perform. Reflected plans are GRL plans that can be bound to roles; the plan can then be activated by calling “start” with the bound role as an argument. The code segment below shows the *main* plan for the system:

```
(define-plan (main)
  (trigger-when bindings-changed?)
  (stop-indirect activity)
  (global-unbind! -1)
  (bind-activity-roles!)
  (bind-color-roles!)
  (start-indirect activity))
```

The *main* plan is activated when the bindings from the Command Console (see Section 7.6) change, i.e. when the user inputs a new command for the robot team. First, *main* stops the current activity and undoes all current bindings. Then, it binds the current inputs to the appropriate roles; the activity role holds the name of the current reflective plan. Finally, it activates the current reflective plan by passing the activity role to the start-indirect procedure.

Each reflected plan consists of a set of inference rules that are recomputed on every process cycle. This continual recomputation alleviates the need for error detection and recovery logic that could otherwise be necessary, hence simplifying the inference rules.

7.4.1 Town-Crier

The inference rules contained in the Town-Crier plan are:

1. If `at-landmark(X)` and `unvisited-landmark(X)`, then `speak-string()`.
2. If `TRUE`, then `bind(Y, get-unvisited-landmark())` and `goto(Y)`.

The first rule causes the robot to speak the announcement if the landmark it is currently at has not been visited previously. The announcement should be made only once at each landmark to avoid annoying the inhabitants of local offices with repetitive announcements.

The second rule is always true. The procedure `get-unvisited-landmark()` returns the current landmark if all landmarks have been visited. Otherwise, the next landmark

to be visited is bound to a role and passed to `goto()`, which is a polymorphic function that is explained further in Section 7.4.4.

In the original specification of the Town-Crier task, the user was supposed to provide the robots with the text of the announcement via the Command Console. However, a component of the Command Console necessary for this was not implemented in time. Therefore, the current announcement is actually hardwired into the Town-Crier plan. Nevertheless, this does not affect the HIVEMind coordination system in any way. So, it was considered an acceptable flaw.

7.4.2 Find-Object

The inference rules for Find-Object are:

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

`See-object()` is a predicate provided by the color trackers in the tracker pool. It takes a role as an argument, and returns true if the object bound to the role is visible in the camera. The first rule moves the robot towards the target object if it is in sight.

`Location()` is a function that takes a role as an argument and returns a list of two landmarks from the place pool. If the current location of the target object is known but

not in sight of the current robot, then the current location is bound to a role. Recall from Chapter 4 that roles may be bound to multiple items in a pool. Then, the role is passed to `goto()`. The `goto()` function attempts to navigate the robot to the first landmark bound to the role, then the second; hence traversing the corridor where the target is located. During this traversal, the first inference rule should fire when the target becomes visible.

If the target is not currently seen, and the robot does not know the target's position, then the third rule calls `next-unsearched-location()`. This function returns a landmark that is bound to a role and then passed to `goto()`. The procedure `next-unsearched-location()` is the first step of the search functionality described in Section 7.3.5; the `goto()` function subsumes both the second and third step.

7.4.3 Sentry

The Capture Evading Target task is performed by the “Sentry” reflective plan. The inference rules for this task are:

1. If `not(observed(X))`, then `bind(Y, next-unsearched-location())` and `goto(Y)`.
2. If `see-object(X)` or `location(X)` is known, then set `observed(X)` to `TRUE`.
3. If `not(all-see(X))` and `see-object(X)`, then `set-current-mode(SENTRY)`.
4. If `current-mode()` is `SENTRY`, then `stop-moving()` and `announce-taunt()`.
5. If `not(see-object(X))` and `location(X)` is known, then `set-current-mode(STALKER)`.

6. If `current-mode()` is STALKER, then `bind(Y, trapping-path(location(X)))` and `goto(Y)`.
7. If `all-see(X)`, then `set-current-mode(TRAP)`, `goto(X)` and `announce-capture()`.

The rules for this task are somewhat more complicated than the other two tasks because the task itself is a more complex one.

Recall from the previous chapter that the execution of this task involves the robots going through several different modes. Rule 1 is the *search* mode. If the target has not yet been observed, either by the current robot or its teammate, then the robot should perform a normal search, i.e. move through any unexplored corridors and eliminate them from consideration. When the target is observed, Rule 2 terminates the search mode.

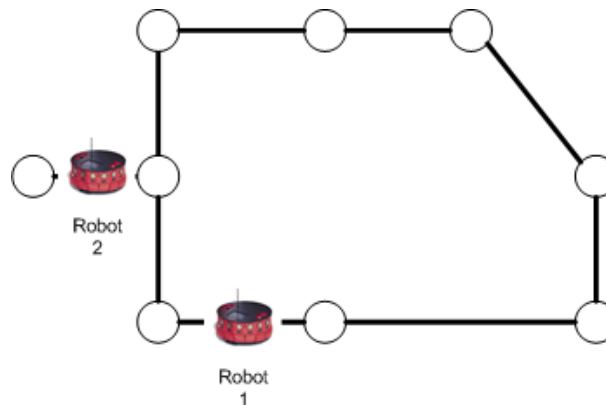


Figure 7.9: Rule 1 – The robots search for the intruder

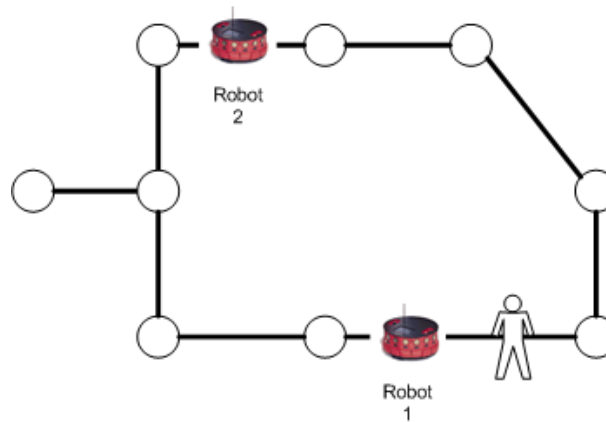


Figure 7.10: Rule 2 – The intruder has been spotted

Rules 3 and 4 concern the *sentry* mode, i.e. the current robot sees the target, but its teammate does not. The former rule sets the mode, while the latter halts the robot, and has it taunting the target with spoken insults. If the target attempts to escape by running away, the sentry does not move, but simply remains in position and continues taunting.

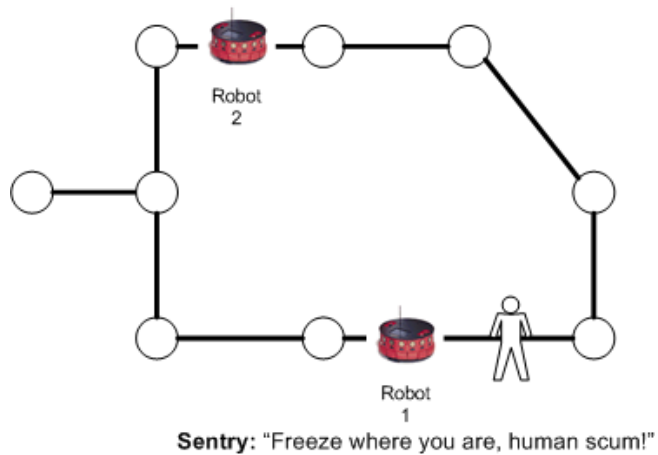


Figure 7.11: Rules 3&4 – Robot1 halts and taunts the intruder in *Sentry* mode

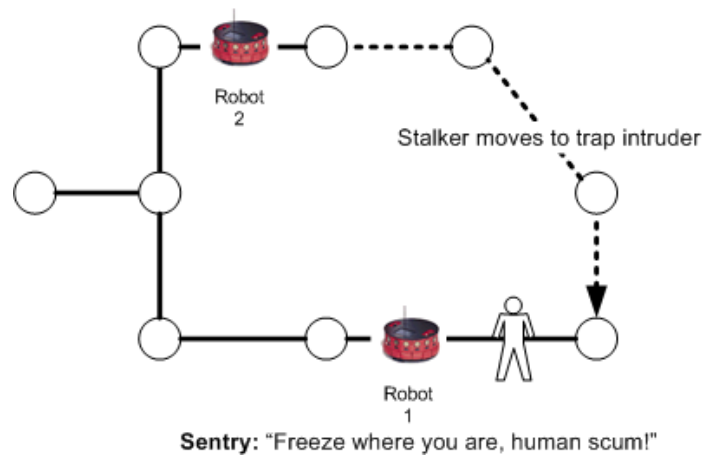


Figure 7.12: Rules 5&6 – Robot 2 (the *Stalker*) moves around to trap the intruder

The *stalker* mode is handled by rules 5 and 6. The former activates the stalker mode if the target is not seen, but its location is known from a communiqué by a teammate. Rule 6 computes a path to the target location using the `trapping-path()` function, which is described in Section 7.3.4. Remember that the objective of the stalker is to arrive at opposing landmark from its teammate in the corridor, hence trapping the target between them.

Finally, rule 7 activates when both robots see the target. This rule changes the mode to *trap*, and moves the robots toward the target. Both robots move forward, reducing the area in which the target until he or she can no longer move, while simultaneously calling out taunts.

The sentry and stalker modes were put in place primarily for the purpose of showing the robots dynamically switching responsibilities. When the target runs away from the sentry, he or she will run into the stalker robot. The stalker then becomes the

sentry, and vice versa. This demonstrates the ability of the robots to immediately react to changing circumstances and switch functions when necessary.

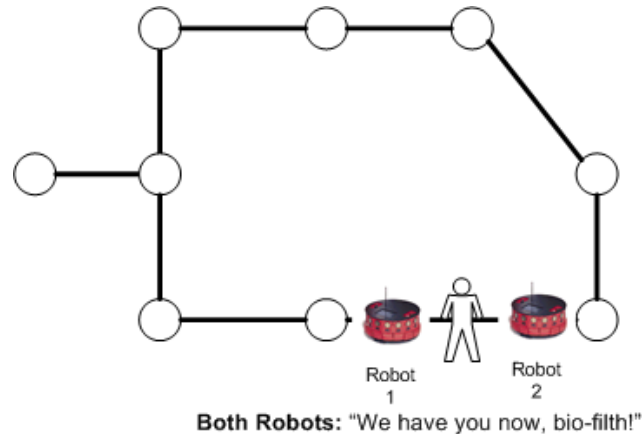


Figure 7.13: Rule 7 – Both robots move to trap the intruder while taunting

In addition, the navigation system for the robots relies on visual feature detectors. This means that the robots are severely hampered when a large object, such as the target, is located directly in front of them. By stopping in place, the sentry is able to latch its current position and be assured that its localization system will not be confused by the target. When both robots are in trap mode, the target is already cornered, so localization is less important.

7.4.4 Goto

`Goto()` is a *generic action* in GRL. Generic actions provide a data-directed programming mechanism similar to Common Lisp generic functions. When a generic action is called, all methods associated with it attempt to match themselves to the argument values. All matching methods then run in parallel until one of them

terminates. In this HIVEMind implementation, there are three methods associated with `goto()`, and each one is designed to run exclusively of the others. `Goto()` takes one argument: a role variable.

The first method is `goto-visual`, which is active when the role passed it is bound to a color tracker and the `see-object` predicate provided by that tracker returns true. This method triggers the approach behavior (see Section 7.5.1) with the current role argument when it is active.

The second method is `goto-place`. This method is triggered when the role argument is bound to a landmark. When active, this method calls `goto-landmark`, which is part of the navigation system described in Section 7.3.3.

The final method is `goto-locations`, which is triggered when the role is bound to multiple landmarks (rather than just a single landmark, as in the previous method). This method calls `goto-landmark` on each landmark bound to the role in turn, so the robot can navigate to a sequence of landmarks using this approach.

7.5 Behaviors

Behaviors in the HIVEMind system are feedback control loops that have a “switch”, i.e. can be turned on or off. The switch can be connected directly to the sensory systems, or may be driven by expressions in plans. A behavior continually measures the difference between its goal and the true state of the world (via the sensory system), computes an adjustment to compensate or partially compensate for it, and issues the adjustment to

the robot motors. The output vector from the behaviors has two components, one representing the rotational velocity, and the other the translational velocity.

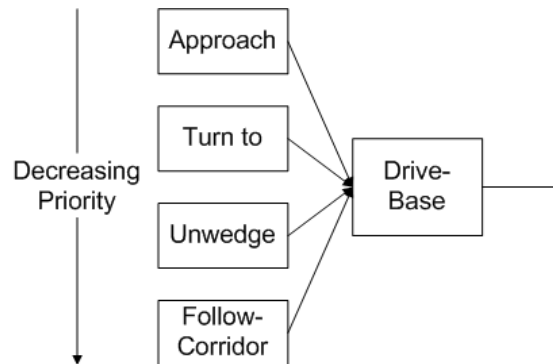


Figure 7.14: Priority of Behaviors

All behaviors are ostensibly run in parallel, and any number of them may be active at any time. The system arbitrates between multiple simultaneously active behaviors through the use of prioritization, i.e. a fixed ordering has been imposed on the set of behaviors. If multiple behaviors are active, the one that has the highest priority gains control of the motors. There are four behaviors in this system, with the priority ordering shown in Figure 7.14.

7.5.1 Approach

The *Approach* behavior takes one argument: a role that is bound to the color of the target object. This behavior tries to keep the target in the center of its field of view at all times. The camera field of view is represented as a set of Cartesian coordinates, with $(f_{width}/2, f_{height}/2)$ as the center point. We use the two rules

$$v_{rotation} = \alpha_r \left(\left(\frac{f_{width}}{2} \right) - r_x \right)$$

$$v_{translation} = \max \left(\alpha_t \left(\left(\frac{f_{height}}{2} \right) - r_y \right), 0 \right)$$

where $v_{rotation}$ and $v_{translation}$ represent the rotational and translation velocities for the robot respectively; r_x and r_y are the current x, y coordinates of the target in the field of view. α_r and α_t are the rotational and translational gain parameters.

The robot smoothly rotates or moves forward in order to track the target in its field of view, decelerating as the target approaches the center of the camera view. If the target is below the center of the camera view (i.e. it is closer than necessary), the robot does not back off; hence the max component of $v_{translation}$.

7.5.2 Turn-to

When the robot reaches a landmark or intersection, it executes a ballistic turn in order to switch from one corridor to another. This ballistic turn is performed by issuing an open-loop ballistic command to the robot base. This behavior is driven by the *turn-to-dest* plan, which computes the desired heading given the current landmark and the next destination. This final heading is passed to *turn-to* as an argument. *Turn-to* keeps the translational velocity at zero when it is executing (it is not a good idea to move forward while turning fast blindly), and terminates when the current robot heading is within some small tolerance of the desired heading.

7.5.3 Unwedge

Occasionally, the robot will get stuck in a local minimum. When this happens, the robot performs an open-loop turn in the direction it thinks has the most open space in order to free itself. The stuck rule is defined as

$$stuck? = (> (truetime(< vel_t vel_{t_{min}})) t_{stuck})$$

That is, the robot is stuck when the translational velocity (vel_t) has been below some threshold vel_{min} for at least t_{stuck} amount of time.

7.5.4 Follow-corridor

The *follow-corridor* behavior actually consists of four subordinate behaviors arbitrated once again through prioritization.

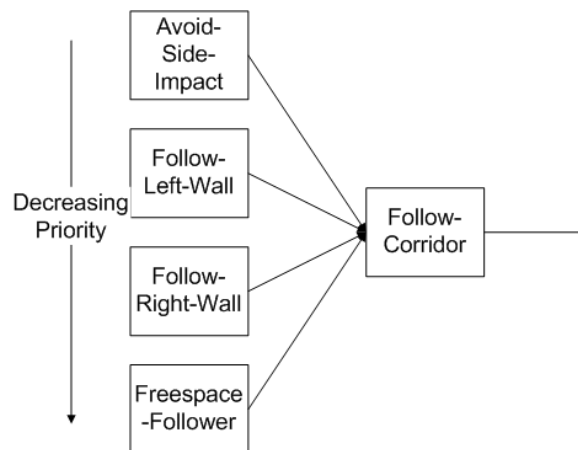


Figure 7.15: Priority of Follow-Corridor Behaviors

The translational velocity component for all four behaviors is identical:

$$vel_t = \max(\alpha(d_{center} - d_{stop}), 0)$$

where d_{center} is the distance to the closest object in the center of the robot's field of view, d_{stop} is the stopping distance and α is the gain parameter. That is, the robot will move forward as long as the closest object in front of it is beyond some stopping distance away. The further away the center object, the faster the robot will move. As it approaches the object, it will decelerate and come to a stop.

The remainder of this section will focus on the rotational velocity component of the behaviors.

The *avoid-side-impact* behavior is active when there is an obstacle too close to the left or right side of the robot. When this happens, this behavior overrides any wall-following behaviors and turns the robot away from the offending object.

Follow-left-wall and *follow-right-wall* align the robot with the walls that form the corridor. The Polly algorithm [Horswill 1994] computes a depth map from the camera image by labeling floor pixels and finding the image plane height of the lowest non-floor pixel in each column (see below).

The walls along the corridor are found by projecting a line along the left and right regions of the depth map, and using the following rule

$$wall? = (l_{error} < l_{min_error}) \text{ and } (l_{slope} > l_{min_slope})$$

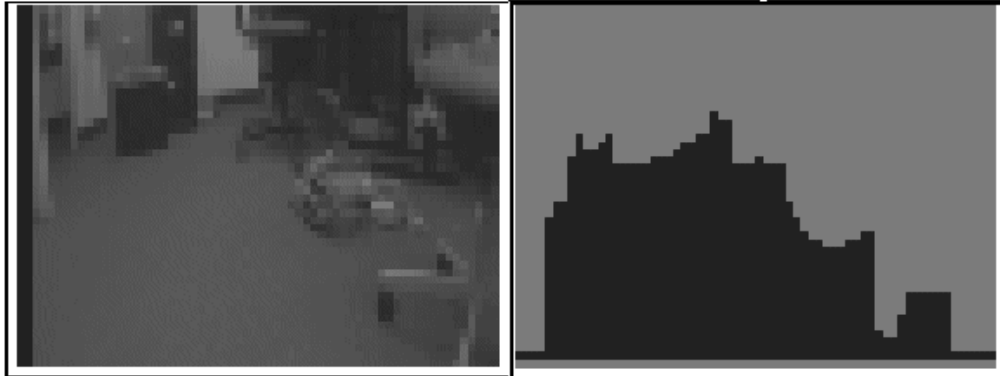


Figure 7.16: Original image and depth map computed from Polly algorithm. Used with permission from [Horswill 1994].

That is, the robot sees a wall if the line fit error l_{error} is below a threshold l_{min_error} , and the slope of the projected line l_{slope} is above a threshold l_{min_slope} . The wall-following behaviors are active when their respective walls are observed using the above rule. The rotational velocity of the robot is then calculated in the following way:

$$vel_r = \alpha_1 v_{freespace} + \alpha_2 (vp_{current} - vp_{desired})$$

$v_{freespace}$ is the rotational component of the freespace-follower behavior (described below), $v_{current}$ is the estimate of the vanishing point based on the line/wall projection, and $v_{desired}$ is the ideal vanishing point in the center of the camera image. α_1 and α_2 are gain parameters that acts as weights, making the rotational velocity a weighted sum of the freespace rotation and vanishing point computation.

The *freespace-follower* behavior runs by default if no other behaviors are active. Its purpose is simply to guide the robot to the area with the greatest amount of open space within sight. The rotational velocity of this behavior operates using the rule

$$vel_r = \alpha(d_{left} - d_{right})$$

where d_{left} and d_{right} are the distances to the closest object in the left and right regions of the camera image respectively. α , as usual, is a gain parameter.



Figure 7.17: Line projections along the left and right walls of the corridor

7.6 Command Console

The command console allows the human user to provide the robot team with the current task, as well as any arguments that may go along with it. A status display shows the user a map of the environment with the current positions of the robots, as well as any other status information. The console has the exact communication interface as the physical robot members of the team. Hence it appears as ‘just another robot’ as far as the robots are concerned; albeit one that performs no useful work.

7.6.1 Command Interface

The (very limited) natural language interface for the command console is based on a simplified version of the system described in [Horswill et al. 2000]; specifically, the

scaled-down system can only handle imperative input statements, such as “find green”, or “announce”. The interface system uses a rudimentary finite-state parser that contains a small lexicon of words. As words are typed into the system, the parser looks up their lexical categories and binds them to appropriate roles. A typical interaction with the system is shown below:

```

Hello.

    > find green

ok.

Cerebus find green.

OK. Activating team.....

```

The task (“find” in this case) is bound to the *activity* role, and the argument *green* (indicating the color of the object to be found) is bound to the *object* role. The bindings are stored in an alist format and transmitted to all members of the team. The alist formed by the above interaction would be: ((find . 16384) (green . 16)).

7.6.2 Status Display

The above figure shows the status display window of the command console. The display shows the topological map used by the robots, with the names of the nodes marked, and the current position of any robots. The display window originally had a black background, with the map nodes and edges in bright yellow, and the robots in different shades (including blue, red, etc). For the purposes of this document, I reversed

the colors and converted the image to grayscale for better viewing. In the above figure, there is a robot at the gauntlet node on the far left, shown by a lighter colored circle with a reverse v-shape protruding from it indicating the heading of the robot.

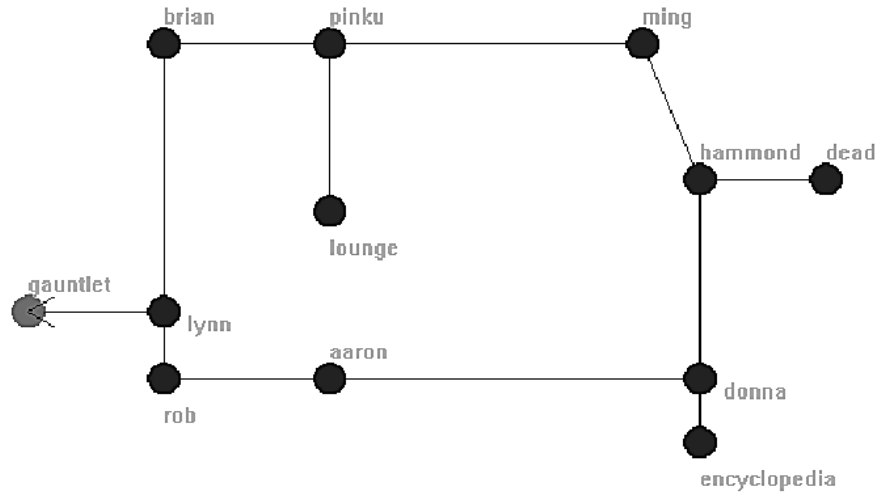


Figure 7.18: Map display on the command console

During the find object task, the line representing the corridor where the target object is located will be highlighted in green. For the capture evading target task, there are two cases. If the target is observed by only a subset of robots, the corridor is highlighted in blue; if all robots in the team see the target, then the corridor is highlighted in red.

7.7 Communication

To perform the current set of tasks, the robots need to communicate four pieces of data:

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

The status window on the command console requires an additional four pieces of information in order to render the current position of the robot correctly:

- The last landmark visited by the robot (the source node).
- The landmark that the robot is heading towards (the destination node).
- The current base odometry reading in x-coordinates.
- The current base odometry reading in y-coordinates.

The number of aggregation functions and the number of shared GRL signals should be the same. Using the *define-default-peer* macro discussed in Chapter 5, the configuration for the HIVEMind communication system for a single robot is:

```
(define-default-peer peer-comm
  1
  (set-alist-bindings
    merge-matrices
    set-team-know-location
    set-locations
```

```

no-op
no-op
no-op
no-op)
(#f
  (source (searched? ils-search-record))
  self-know-location?
  role-locations
  source-node
  dest-node
  (round->integer base-x)
  (round->integer base-y))

```

In the above case, the current robot identification number is 1, and there are eight pieces of communication data corresponding to the ones described previously. However, there is a difference for *role-locations*. Rather than passing a locations array representing all roles, *role-locations* is a list containing a subset of the roles and the locations associated with them. Roles that do not appear in the list are assumed not to have any locations associated with them.

The *no-op* function is essentially a placeholder combination function that does no actual work. Recall that the physical robots do not need the last four pieces of data; they are purely for the benefit of the command console. Physical robots always transmit

`#f` (FALSE) for the role bindings; the task of setting the role bindings is solely the responsibility of the command console. All the physical robots have identical definitions for *define-default-peer*, with the exception of their unique identification numbers. On the other hand, the command console has a different structure:

```
(define-default-peer peer-comm
  0
  (no-op
    merge-matrices
    set-team-know-location
    set-locations
    set-robot-source-vect
    set-robot-dest-vect
    set-robot-x-vect
    set-robot-y-vect)
  (transmitted-lexicon-bindings
    #f
    #f
    #f
    #f
    #f
    #f
```

#f)

By convention, the command console always has an identification number of 0. The console does not transmit any useful data except for the current role bindings in *transmitted-lexicon-bindings*; hence the list of *#f* values. However, it does make use of the latter four data communicated by the robots to determine where to put them on the map display, as represented by the four aggregation functions *set-robot-source-vect*, *set-robot-dest-vect*, *set-robot-x-vect*, and *set-robot-y-vect*.

Chapter 8

Tasks implemented using HIVEMind

The HIVEMind architecture is designed to be an efficient coordination mechanism for multi-robot systems that retains the variable binding capabilities normally found only in traditional symbolic systems. I have successfully constructed a team of three members, consisting of two physical robots and the command console, that perform the three tasks involving tightly coordinated search described in Chapter 2. The control and coordination components that form the core of the HIVEMind architecture worked flawlessly in all the tests performed on this multi-robot team. All team members maintained a shared situational awareness in real time (assuming no outright communication failures) using the efficient virtual wires model that employed very little of the available network bandwidth. The reasoning system on the physical robots operated on inference rules that utilized indexical roles bound in the sensory system, giving them more representational power than traditional behavior-based techniques.

At present, the landmark recognition system is the weak point of the implementation. This system relies on vision provided by a camera fixed on the robot that is only about two feet off ground level. The recognition of landmarks is tuned to the robot being in the center of the corridor; any deviation from this, perhaps due to a passerby or other such unexpected obstacles, could lead to misclassifications. Minor

errors are common, but the navigation system is generally robust enough to handle these. False negatives (i.e. the robot does not see a landmark even though one has been encountered) can lead to sub-optimal behavior since the robots will try to revisit an encountered landmark since they did not detect it initially; however, this does not lead to catastrophic failure. On the other hand, false positives can occasionally lead to serious problems. A robot could believe that it has visited a landmark that it actually has not, and this information propagates to the entire team; now, there is a location that will go unexplored because the team believes it has already been searched. While the landmark recognition system presents some challenges, it should be emphasized again that the HIVEMind architecture itself performed exactly as it should have.

In the following sections, I will discuss experiments pertaining to each of the three tasks performed by the robot team. It should be noted that the HIVEMind approach makes no claims to create teams that produce more optimal behavior, i.e. cooperative robots that can provably complete a task in less time or consume less power. Multi-robot team programmers may use the HIVEMind approach to create such teams, but that is orthogonal to the current discussion. For this reason, the data presented below is qualitative rather than quantitative. The experiments described here serve to demonstrate that the HIVEMind approach may be successfully used to create robots that can cooperate using an efficient virtual wires model to solve complicated tasks that require communication and tight coordination.

All experiments were run in the environment described in Chapter 7. The robots drive at approximately 1m/s in straight corridors, although stopping for ballistic turns at corners and intersections somewhat reduces their mean velocity. Sensing, inference and control decisions are each performed at 10Hz. Each robot transmitted its beliefs to the others at the rate of 1Hz.

8.1 Town Crier

For the Town Crier experiments, all team members were started from a central point at the extreme east end of the wing near the node labeled “Gauntlet” on the map. Generally, the physical robots were able to visit all the landmarks and make the announcements under 5 minutes. In three out of twelve test runs, a false positive identification of a landmark led to the robots not announcing at all landmarks.


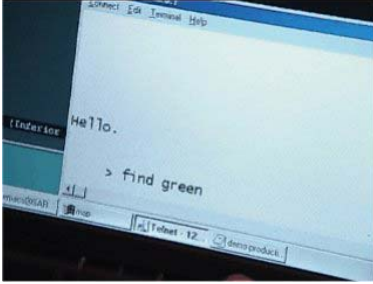
8.2 Find Static Object





In the Find Static object experiments, team members were again started from the Gauntlet location. The goal object, a green ball, was placed out of view, about 15-20 meters from the starting point; it was always located 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 began a systematic search of the wing for the goal object. The systematic search approach guarantees that each landmark is searched at most once and that all landmarks are searched if necessary. The robots were

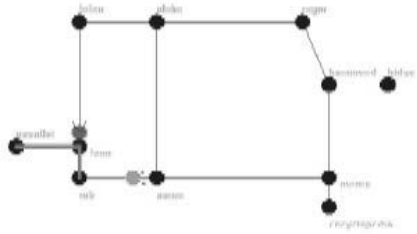


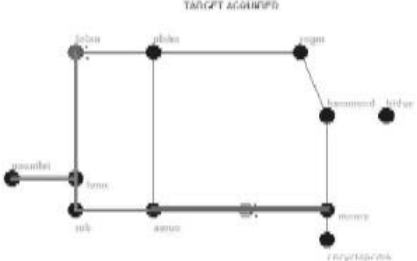
consistently able to locate the green ball within 30 seconds provided that there were no catastrophic failures of the landmark recognition system. Once the first robot discovered the position of the target object, its teammate was typically able to converge on that location within another 20-30 seconds.

8.2.1 Transcript of a Find Static Object test run

The following screen captures were taken from a video of one of the test runs for the Find Static Object task.

<u>Comments</u>	<u>Screen Capture</u>
Both physical robots are started from the Gauntlet location.	
The human user enters “find green” on the command console. This information is propagated to all team members via virtual wires.	

<p>The robots begin their mission to locate a green-colored static object within the environment.</p>	 A photograph showing two mobile robots in a hallway. The robot on the left is smaller and has a monitor on top. The robot on the right is larger, red, and also has a monitor on top. They are positioned at the start of a hallway.
<p>The first robot reaches the t-intersection and proceeds to turn right.</p>	 A photograph of a long, narrow hallway with a carpeted floor. A small robot is visible in the distance, positioned at the end of the hallway.
<p>The second robot arrives at the t-intersection; meanwhile, the first robot has started up the right corridor.</p>	 A photograph of a long, narrow hallway with a carpeted floor. A small robot is visible in the distance, positioned at the end of the hallway.
<p>The second robot knows that the right corridor is already being searched by its counterpart, and proceeds to turn towards the left corridor.</p>	 A photograph of a long, narrow hallway with a carpeted floor. A small robot is visible in the distance, positioned at the end of the hallway.

<p>The user monitors the team on the command console's status display window. This shows the positions of the robots, and the corridors that have been searched (thicker lines).</p>	
<p>The first robot approaches the green ball down the hallway.</p>	
<p>The image on the right shows a view from the first robot's camera. The horizontal lines represent distance to the closest obstacle within that line area. The small green ball can be seen in the middle near the top of the image as a small circle.</p>	
<p>When the first robot locates the green ball, it informs the rest of the team of this. On the command console, the corridor where the ball is located is highlighted green. Meanwhile, the other robot moves to that</p>	

corridor to join its teammate at the green ball.	
--	--

8.3 Capture Evading Target

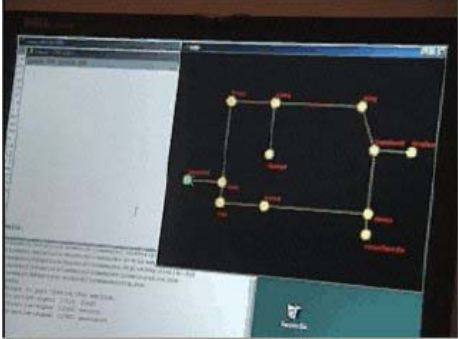
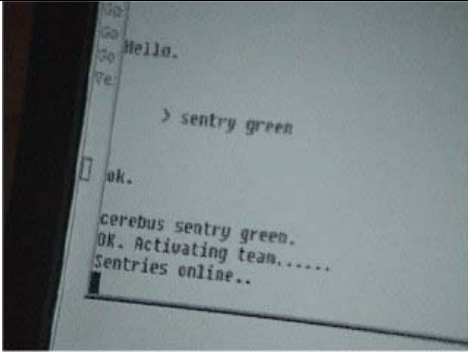
The robots were once again started at the Gauntlet position for the Capture Evading Target task. These experiments were conducted with the aid of some fellow graduate students, as I played the part of the evading target. They ran the commander console and the camcorders during the experiment. The evading target always started out at the *deadend* node, on the far end from the starting position of the robots. From there, I always took an initial route through the nodes *Hammond* → *Donna* → *Aaron* → *Rob*. If a robot was encountered, I would turn around and attempt to escape down the other corridor (*Hammond* → *Pinku* → *Brian*).


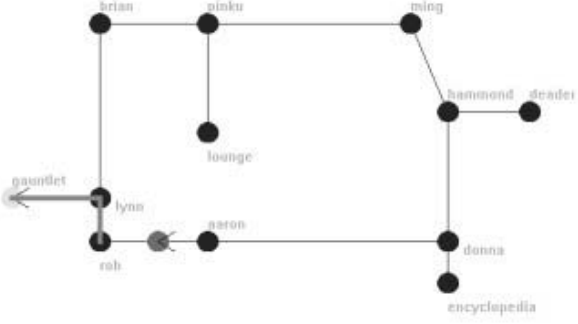

The robots were consistently able to locate me quickly and begin their *sentry-stalker* routine, as described in Chapter 7. Generally, the first robot would see me within 30-40 seconds, whereupon it would halt and begin taunting me. Its counterpart would then find me as I backtracked and tried to escape through the other corridor. Finally, the robots would trap me between the two of them between one or two minutes.


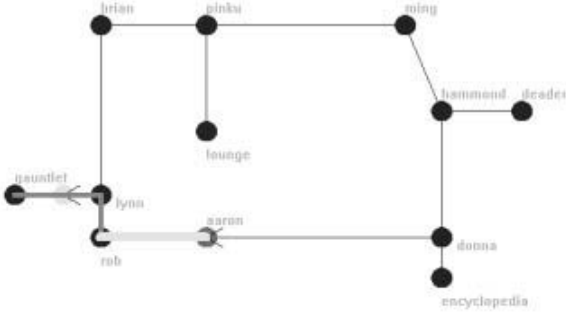

8.3.1 Transcript of a Capture Evading Target test run


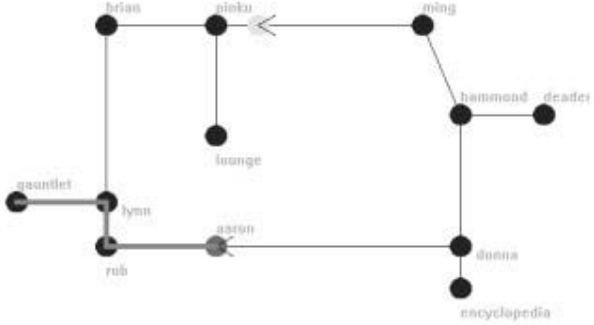

The following screen captures were taken from a video of one of the test runs for the Capture Evading Target task. This particular test run was interesting because one of the

robots actually failed during the run. However, the ability of HIVEMind to robustly handle failed members and to rapidly integrate new robots came to the rescue. The failed robot was quickly rebooted, whereupon it rejoined the HIVEMind and was able to continue functioning as before.

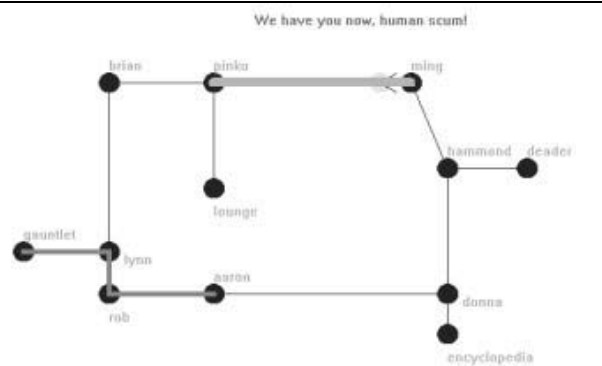
<u>Comments</u>	<u>Screen Capture</u>
<p>As with the other two tasks, the user is responsible for entering the current task via the command console.</p>	
<p>This time, the command is “sentry green”, which activates the Capture Evading Target program, and sets the color of the intruder to be green.</p>	 <pre> 100 101 Hello. 102 Te. 103 104 > sentry green 105 106 [] ok. 107 cerebus sentry green. 108 OK. Activating team..... 109 Sentries online.. </pre>

<p>Both robots begin at the <i>gauntlet</i> node; the first robot heads down the hallway once the current command has been received.</p>	
<p>The Status Display window on the command console shows the current state of the robots. The first robot is between the <i>rob</i> and <i>aaron</i> nodes, while the second is just starting out from the <i>gauntlet</i>. A thicker line indicates that an area has been searched.</p>	
<p>The first robot is heading down the long hallway.....</p>	

<p>.....when it spies an intruder wearing green! The robot halts in its tracks and begins taunting the intruder.</p>	
<p>In the meantime, all other members of the team have been informed of the intruder's presence and location. The thick, lighter colored line between <i>rob</i> and <i>aaron</i> indicates the current location of the intruder.</p>	<p style="text-align: center;">You can run, but you can't hide!</p> 
<p>Knowing the location of the intruder, the second robot plans a path that will trap the intruder by blocking the path to the exit.</p>	

<p>The second robot proceeds along its path (traversing <i>gauntlet</i> → <i>lynn</i> → <i>brian</i> → <i>pinku</i> → <i>ming</i>)</p>	
<p>While the second robot is coming around, the intruder tries to flee from the first robot. That robot doesn't pursue, but remains in position and continues taunting. The Status Display window shows that the first robot no longer has the intruder in sight.</p>	 <pre> graph TD gauntlet((gauntlet)) --- lynn((lynn)) lynn --- brian((brian)) brian --- pinku((pinku)) pinku --- ming((ming)) ming --- hammond((hammond)) hammond --- reader((reader)) ming --- donna((donna)) donna --- encyclopedia((encyclopedia)) lynn --- aaron((aaron)) aaron --- rub((rub)) rub --- donna pinku --- lounge((lounge)) lounge --- aaron </pre>
<p>The intruder runs right into the second robot as he tries to escape through the other long corridor.</p>	

However, at this point, the operating system on the first robot (Windows 98) crashes! Notice this robot has disappeared from the Status Display. The second robot reacts accordingly and moves towards the intruder. In the meantime, frantic rebooting is being performed at the first robot.



After the first robot has returned to life, it rejoins the HIVEMind within one transmission cycle and proceeds to the current location of the intruder (in truth, the intruder did not move during the reboot of the first robot). The second robot waits for its counterpart to arrive at the intruder's position.



When both robots have the intruder in view simultaneously, they both move forward to “trap” the intruder between them.



Chapter 9

Other Multi-Robot Control Strategies

The joint beliefs approach I have discussed so far assumes a coordination model where the beliefs of each participant are strongly synchronized through explicit communication. Obviously this is not the only possible view of coordination. There is plenty of work in the robotics literature describing different strategies for coordinating multiple robots. Some researchers have investigated techniques that rely on eventual convergence over time. Others have eschewed active communication altogether, relying instead on plan recognition through passive observation or a single world model located on one physical machine that is shared by all robots. In this chapter, I will briefly discuss some alternative strategies for coordination.

9.1 Swarm Cooperation

One major area of research in cooperative multi-robot systems involves the study of large numbers of homogeneous cooperative robots; these large robotic groups are generally called *swarms*. The control techniques used in this research draws inspiration from a variety of fields, including neurobiology, evolutionary biology and sociology [Kennedy and Eberhart 2001]. These approaches generally make use of a set of simple

control laws and do not utilize any world models. The typical methodology used is a stochastic one, relying on mathematical convergence results (such as the random walk theorem [Chung 1974]) that produce the desired outcome over a sufficiently long period of time. Swarm researchers apply principles such as *stigmergy*, i.e. indirect communication between individuals via modification of a shared environment, to achieve the collective behavior. Stigmergy was first described by Grasse [Bonabeau, Dorigo and Theraulaz 1999] as a way of explaining how insect societies could collectively generate complex behaviors while each individual seemingly works on its own.

Swarm robots are very reactive to the environment and characteristically rely on local interactions to generate desired global effects. This has the advantage of overall robustness, i.e. reducing sensitivity towards individual robot failures, but at the cost of increased difficulty in maintaining global team coherency.

Most of the tasks studied using these methodologies involve group behaviors such as harvesting, collecting, foraging and flocking. [Deneubourg et al. 1990] describe strategies for the collection and transportation of objects; the cooperative behavior between the simulated robots emerged either through implicit or explicit communication. The experiments were performed on simulated and physical “ant-like” robots. In [Kube and Bonabeau 2000], the authors present a formal model of cooperative transportation of objects by ants, and describe an implementation of this on a group of physical robots. [Melhuish, Holland, and Hoddell 1998] show that very

simple, homogeneous robots are able to sort and segregate items of different colors by sensing only the object colors and not the local density of the objects. [Steels 1990] present simulations of rock sample collection on a distant planet using dynamical systems (partially random movement, potential fields and a dissipative structure). [Mataric 1992] describes implementing group behaviors such as dispersion, aggregation and flocking on groups of up to 20 physical robots. [Kube and Zhang 1992] present an emergent control strategy used on a group of five physical robots performing a box pushing task. They extend this work in [Kube and Zhang 1994] by examining the phenomenon of *stagnation*, i.e. members of a team are not cooperating efficiently with each other, leading to suboptimal or even detrimental results. For example, in box pushing, stagnation may result from robots pushing from opposite directions, canceling each other's efforts. The paper describes a solution requiring no explicit communication between the robots; the robots cooperate solely through direct interactions with the box itself.

9.2 Coordination through observation

Some researchers have proposed systems that cooperated solely through mutual observation alone. Explicit communication, it is argued, can be unsuitable if the method of communication is unavailable (e.g. an enemy jams the transmission signals) or risky (e.g. stealth is a key component of the mission). Moreover, it is possible that systems designed with explicit inter-robot communication in mind will have a difficult time

cooperating with systems that are outside the scope of that communication methodology (e.g. a human being).

[Kuniyoshi et al. 1994] present a framework for cooperation by mutual observation, and discuss several issues such as viewpoint constraint and role interchange between robots. They also introduce the concept of *attentional structure*, which defines a set of relationships and roles between members of the cooperative team. Their approach is illustrated on a group of physical robots that cooperatively perform tasks such as posing (following a leader), passing an object between two team members and unblocking (i.e. one robot clears an obstructed path for another).

[Tambe and Rosenbloom 1995] describes RESC (REal-time Situated Commitments), an approach for tracking a single agent's on-going actions in the context of the current situation. RESC heuristically commits to a model of the agent being tracked, relying on a repair mechanism to correct any errors. [Tambe 1996b] extends this work to a multi-agent team setting. This work was demonstrated on a simulation of helicopters on an attack mission.

[Wie 2000] presents a probabilistic solution for mutual observation between robots. Plans are treated as scripts that are converted to Hidden Markov Models (HMMs). The HMMs are then used during the plan recognition process to determine the current intent and actions of the other robots. This approach was implemented on a Robocup simulation league team.

The main problem with passive observation-only approaches is that all team members have to be within sensory range of each other. A robot cannot infer its teammates' actions or intent if it cannot perceive them and cannot communicate with them. The techniques in this report are intended for use in domains where passive communication through observation is insufficient. That is, team members must explicitly communicate with each other in order to coordinate their activities.

9.3 Centralized world model

Research has been done in domains where there are multiple acting agents that share a single global perspective. For example, in the Robocup small-sized robot league (also known as the F180 league), an overhead camera provides global vision of the field. Reasoning is generally performed at a central server location where the master knowledgebase is located (although each robot may have its own client thread on the server), and then actions are transmitted to the individual robots. Little, if any, reasoning is done on the physical robots themselves. See [Veloso et al. 1999], [D'Andrea and Lee 1999] and [Kiat et al. 2001] for typical examples of teams that have been built around this environment. However, [Asada et al. 2000] notes that some participants in Robocup 2000 did manage to integrate on-board vision into the size limitations of the small-sized robot league.

The shared knowledgebase eliminates any inconsistencies between world models for different team members, resulting in a simpler coordination process.

However, this type of centralized control is only relevant when a “god’s eye view” sensor that can perceive the entire environment at once and update in real-time is available.

Chapter 10

Conclusions and Future Work

10.1 Discussion on coordination

The space of cooperative team activity is very large, and there are a correspondingly large number of possible solutions for coordinating a team of robots. Ultimately, choosing the right approach depends on many factors, including:

- The nature of the operating environment; for example, the characteristics of an urban disaster area are vastly different from that of a tightly controlled industrial factory.
- Time restrictions; does the application face tight time constraints (e.g. playing soccer) or can it be solved over a long period (e.g. vacuuming a room)?
- The availability of communication. While some applications face no communication constraints, other applications demand little or no communication (e.g. a covert military operation) and still others simply do not have it available (e.g. there is too much interference in the environment).
- The need for tight coordination; some applications require close-knit coordination between team members (e.g. searching room-to-room in a

building) while others do not have such a requirement (e.g. cleaning the floor in a large, empty warehouse).

The HIVEMind architecture is not intended to coordinate all possible robot teams. It is designed to be used for the case where the robots have to tightly coordinate their activities by explicitly sharing data with each other and be aware of the teammate failures. Still, this describes a large class of applications. For instance, many “search” tasks can be solved by small robot teams using this strategy (e.g. clearing buildings in an urban battlefield, surveillance operations, search-and-rescue, etc). In these situations, I have shown that the HIVEMind approach works exceedingly well in these situations. The communication strategy used by the robots is optimal in the number of packets transmitted. The broadcast-and-aggregate coordination mechanism generates a shared situational awareness among the robots, ensuring tight synchronization of their team actions. Any failures are quickly noticed and can be swiftly dealt with, and new resources (i.e. additional robots) can be dynamically integrated into the team.

The HIVEMind architecture has also been implemented in a virtual world. Robots operating in the physical world face the constraints of noisy sensors, limited effectors and unreliable communication. *Bots* that function in a virtual world, on the other hand, are less susceptible to such problems. FlexBot [Khoo et al 2002][Khoo and Zubek 2002] is an SDK for building agents or bots in a first-person shooting game called Half-Life. The agents participate in a deathmatch game where they function in teams that attempt to maximize opposition kills while minimizing their own number of

deaths. All agents, as well as the game engine, operate on a single PC desktop. Since all bots are located on the same physical machine, communication is reduced to a set of shared global variables. The bots use HIVEMind in much the same way as their physical counterparts; on each program cycle, a bot sends (writes) its data into the appropriate slots of shared arrays, and forms aggregate values by combining the values of those arrays. Using the HIVEMind architecture, I implemented a team of bots that dynamically form squads in the game. When a member of the squad is attacked or sees an enemy, the rest of the team reacts to it within one communication cycle. Each bot only uses 0.3% of the CPU cycles and 52 bytes of static data memory during runtime. We have been able to run 32 bots (a hard limitation imposed by the game, not the AI system) and the game engine simultaneously on a present day desktop (1.8 GHz Pentium VI with 1 Gb of RAM).

10.2 Future Work

The implementation of HIVEMind on physical robots has only been tested on the small team described in Chapters 7 and 8 thus far. More experiments should be performed using HIVEMind using larger teams of robots and for different task domains in order to better understand its strengths and weaknesses in practice. Furthermore, while I have discussed some methods for extending the periodic data broadcasting technique for larger teams and data, I have not yet tested these ideas in practice. One possible route is to implement HIVEMind on robots that use infrared for communication. These robots

will require line of sight for communication and also have less bandwidth available.

So, we can experiment with different strategies for maintaining shared situational awareness under these conditions.

In addition, during the development of the HIVEMind architecture, I realized that most existing physical robots systems have very short lives. They are activated, perform their tasks in a few minutes and are then deactivated. While there are now efforts to extend the active lifetimes of these robots to 24 hours and beyond, many of these efforts are hampered by hardware constraints. In particular, a lot of research for long-lived robots has focused on power issues, i.e. getting the robot to recharge itself when batteries are low.

One piece of potentially interesting future work is to use the commander console as the long-lived collective knowledge of the robot team. Throughout the development of HIVEMind system on the physical robot team, the commander console was always considered part of the team. The inter-robot communication interface for the console is identical to that of the physical robots; for all the robots know, the console is an extra member of the team that does no useful work. However, it does receive all relevant information from the team and, more importantly, is the most stable, longest lived member of the team. I would often leave the console running for long periods of time while the other robots were being debugged. When the robots were activated, they hooked into the running HIVEMind on the console seamlessly. The present console does not store data persistently, but this can be easily implemented. Shared data from its

fleeting teammates can be stored and processed, perhaps when no physical robots are active or in a background batch process. Potential areas of research for the shared data on the commander console:

- Data mining on the shared data. When sufficient amounts of shared data from different runs have been collected, this data can be mined for interesting patterns and relationships using data mining algorithms.
- Representation of long term memory. The commander console should store information gleaned from multiple runs and load it onto robots that are newly activated so they immediately possess the knowledge of their predecessors. However, behavior-based systems generally have fixed control structures. How do we represent long term memory in an appropriate form that it can be transmitted efficiently to the robots and be used effectively by their control systems?
- Stabilization of the environment. Humans often modify their surrounding environment in order to make it more hospitable. [Hammond, Converse and Grass 1995] called this “stabilizing the environment”. Is it possible for robots to reason about their environment and change it in ways that make their lives easier? This research problem entails figuring out how the robots can take advantage of long term memory to determine what parts of the operating environment present challenges, and what actions they can take to effect changes in a way that would alleviate those problems in the future.

The ultimate goal of this proposed research direction is very difficult to accomplish, to say the least. While robotic research has progressed to the point where a robot can now traverse the world reliably and even know to a high degree of confidence its location in that environment, many questions remain unanswered. Robots, for the most part, remain sensor and effector poor. However, the ability to alter the environment around us to make our lives easier is one of the hallmarks of human intelligence. The stabilization of the environment frees us from some immediate concerns, allowing us to focus on other interesting tasks and problems. It would be highly beneficial if autonomous robots could be imbued with similar capabilities.

Bibliography

- Agre, P. 1988. *The dynamic structure of everyday life*. Technical Report 1085. Massachusetts Institute of Technology, Artificial Intelligence Lab.
- Agre, P. 1997. *Computation and Human Experience*. Cambridge University Press, Cambridge.
- Agre, P. and D. Chapman. 1987. Pengi: An Implementation of a Theory of Activity. *Proceedings of the Sixth National Conference on Artificial Intelligence*. 268-272. Menlo Park, CA : AAAI Press/MIT Press.
- Albus, J., H. McCain and R. Lumia. 1987. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*. NBS Technical Note 1235, Robot Systems Division, National Bureau of Standards.
- Arkin, R. C. 1986. Path Planning for a Vision-based Autonomous Robot. *Proceedings of the SPIE Conference on Mobile Robots*, Cambridge, MA, pp. 240-249.
- Arkin, R. C. 1989a. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research*, 8(4): 92-112.
- Arkin, R. C. 1989b. Neuroscience in Motion: The Application of Schema Theory to Mobile Robotics. *Visuomotor Coordination: Amphibians, Comparisons, Models and Robots*. Edited by J.-P. Ewert and M. Arbib. 649-672. New York: Plenum Press.
- Arkin, R. C. 1992. Cooperation without Communication: Multi-agent Schema Based Robot Navigation. *Journal of Robotic Systems*, 9(3): 351-364.
- Arkin, R. C. 1998. *Behavior-Based Robotics*. Cambridge, MA: MIT Press.
- Arkin, R. C. and T. R. Balch. 1997. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175-188.
- Asada, M., A. Birk, E. Pagello, M. Fujita, I. Noda, S. Tadokoro, D. Duhaut, P. Stone, M. Veloso, H. Kitano, and B. Thomas. 2000. Progress in RoboCup Soccer Research in 2000. *Proceedings of the Seventh International Symposium on Experimental Robotics*.

- Bajcsy, R., and J. Kosecka. 1994. The problem of signal and symbol integration: A study of cooperative mobile autonomous agent behaviors. *Proceedings of KI-95*, LCNS, pp. 49-64, Berlin Germany: Springer.
- Baker, D. J. and A. Ephremides. 1981. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, 29(11):1694-1701.
- Balch, T. 1999. The impact of diversity on performance in multi-robot foraging, *Proceedings of Autonomous Agents 99*, Seattle, WA.
- Balch, T. R. and R. C. Arkin 1995. Communication in Reactive Multiagent Robot Systems. *Autonomous Robots*. 1(1):27-52.
- Balch, T. R. and R. C. Arkin. 1998. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6): 926-939.
- Basagni, S.. Distributed Clustering for Ad Hoc Networks. 1999. In *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks*, pp. 310-315.
- Beer, R., H. Chiel, and L. Sterling. 1990. A Biological Perspective on Autonomous Agent Design. *Robotics and Autonomous Systems*, 6: 169-186.
- Bernstein, P., V. Hadzilacos, and N. Goodman. 1987. *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison-Wesley.
- Bonabeau, E., M. Dorigo, and G. Theraulaz. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford, UK: Oxford University Press.
- Bonasso, P., R. J. Firby, E. Gat and D. Kortenkamp. 1997. Experiences with an architecture for Intelligent Reactive Agents. *Journal of Theoretical and Experimental Artificial Intelligence*. 9:237-256.
- Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*. RA-2(1): 14-23.
- Burgard, W., D. Fox, D. Hennig, and T. Schmidt. 1996. Estimating the absolute position of a mobile robot using position probability grids. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, CA : AAAI Press/MIT Press.

- Chiang, C-C., H-K. Wu, W. Liu, and M. Gerla. 1997. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. In *Proceedings of IEEE Singapore International Conference on Networks (SICON)*, pp. 197-211.
- Chung, K. L. 1974. *Elementary Probability Theory with Stochastic Processes*. New York, NY: Springer-Verlag.
- Cohen, P. R. and H. J. Levesque. 1990. Intention is choice with commitment. *Artificial Intelligence*, 42:213-261.
- Cohen, P. R. and H.J. Levesque. 1991. Teamwork. *Nous: Special Issue on Cognitive Science and AI*, 25(4):487-512
- Connell, J. H. 1989. *A colony architecture for an artificial creature*. Artificial Intelligence Lab Tech Report 1151, Boston, MA: MIT.
- Connell, J. H. 1992. SSS: A hybrid architecture applied to robot navigation. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*: 2719-2724. New York, NY: IEEE Press.
- Coradeschi, S. and A. Saffiotti. 2000. Anchoring symbols to sensor data: Preliminary report. *Proceedings of the 17th AAI Conference*: 129-135. Austin, TX: AAI Press.
- Coradeschi, S. and A. Saffiotti. 2001. Perceptual Anchoring of Symbols for Action. *Proceedings of the 17th IJCAI Conference*: 407-412. Seattle, WA: AAI Press.
- D'Andrea, R. and J. Lee. 1999. Description of Cornell BigRed Small League RoboCup Team. In *Linköping Electronic Articles in Computer and Information Science*, ISSN 1401-9841, 4: 006/03.
- Davis, R. and R. G. Smith. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20: 63–109.
- Dean, T.L. and K. Kanazawa. 1989. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142-150.
- Depristo, M. 1999. *The Adaptive Color Tracker*. Unpublished technical note. Northwestern University Autonomous Mobile Robot Laboratory (AMRG).

- Deneubourg, J., S. Goss, G. Sandini, F. Ferrari and P. Dario. 1990. Self-organizing collection and transport of objects in unpredictable environments. *Japan-USA Symposium on Flexible Automation*, pp. 1093-1098.
- Eckhardt, D. and P. Steenkiste. 1996. Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network. *Proceedings of the ACM SIGCOMM*: 243-254.
- Everett, J. and K. Forbus. 1996. Scaling up logic-Based truth maintenance systems via fact garbage collection. *Proceedings of the 13th National Conference on Artificial Intelligence*.
- Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds*. PhD Dissertation, Technical Report YALEU/CSD/RR #672, Yale University, New Haven, CT.
- Ferrell, C. 1994. *Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators*. M.S. Thesis, MIT AI Laboratory: Cambridge, MA.
- Floyd, S., V. Jacobson, C. Liu, S. McCanne, and L. Zhang. 1997. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6): 784-803
- Fontan, M. and M. Mataric. 1997. A study of territoriality: The role of critical mass in adaptive task division. *From Animals to Animats 4: Proceedings of the Fourth International Conference of Simulation of Adaptive Behavior*: 553-561.
- Forbus, K. D. and J. D. Klerer. 1993. *Building Problem Solvers*. MIT Press: Cambridge, MA.
- Fox, D., W. Burgard, and S. Thrun. 1999. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391-427.
- Gat, E. 1991. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD Dissertation, Virginia Polytechnic Institute and State University, Blacksburg.
- Georgeff, M. and Lansky, A. 1987. Reactive Reasoning and Planning. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 677-682.
- Gerkey, B. and M. J. Mataric. 2000. Murdoch: Publish/Subscribe Task Allocation For Heterogeneous Agents, *Proceedings of the Fourth International Conference on Autonomous Agents*: 203-204.

- Gerkey, B. and M. J. Mataric. 2002. Sold!: Auction methods for multi-robot coordination". *IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems*, 18(5):758-768.
- Gerkey, B. and M. J. Mataric. 2003. Multi-Robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures. To appear in *Proceedings of the IEEE International Conference on Robotics and Automation*, New York, NY: IEEE Press.
- Goldberg, D., V. Ciciello, M.B. Dias, R. Simmons, S. Smith, and A. Stentz. 2003. Market-Based Multi-Robot Planning in a Distributed Layered Architecture. *Multi-Robot Systems: From Swarms to Intelligent Automata: Proceedings from the 2003 International Workshop on Multi-Robot Systems*, edited by A. Schultz, L. Parker, and F. Schneider. (2):27-38. Kluwer Academic Publishers.
- Goldberg, D. and M. J. Mataric. 2000. *Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks*. USC Institute for Robotics and Intelligent Systems, Technical Report IRIS-00-387.
- Gray, J., P. Helland, P. O'Neil and D. Shasha. 1996. *The Dangers of Replication and a Solution*, Proceedings of the ACM SIGMOD Conference.
- Gray, J. and A. Reuter. 1993. *Transaction Processing: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann.
- Grosz, B. and S. Kraus. 1998. The Evolution of SharedPlans. *Foundations and Theories of Rational Agencies*, edited by A. Rao and M. Wooldridge.
- Hammond, K. J., T. M. Converse, and J. W. Grass. 1995. The stabilization of environments. *Artificial Intelligence*. 72, 305-327.
- Hasegawa, T., Y. I. Nakano, and T. Kato. 1997. A Collaborative Dialog Model Based on Interaction Between Reactivity and Deliberation. In *Proceedings of the First International Conference on Autonomous Agents*, pp. 83-87. Marina del Rey, CA: ACM Press.
- Hexmoor, H., J. Lammens, and S. C. Shapiro. 1993. Embodiment in GLAIR: A grounded layered architecture with integrated reasoning for autonomous agents. In *Proceedings of the Florida AI Research Symposium*, pp. 325-329.

- Horswill, I. 1994. *Specialization of Perceptual Processes*. Technical Report AI TR-1511. Cambridge, MA: MIT AI Lab.
- Horswill, I. 1998. Grounding Mundane Inference in Perception. *Autonomous Robots*, 5: 63-77.
- Horswill, I. 1999. Functional programming of behavior-based systems. *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*.
- Horswill, I. 2000. *The GRL Primer*. Unpublished technical note. Northwestern University Autonomous Mobile Robot Laboratory (AMRG).
- Horswill, I. 2001. Tagged Behavior-based Architectures: Integrating Cognition with Embodied Activity. *IEEE Intelligent Systems*: 30-38. New York: IEEE Computer Society.
- Horswill, I., R. Zubek, A. Khoo, C. Le, and S. Nicholson. 2000. The Cerebus Project. *AAAI Fall Symposium on Parallel Cognition and Embodied Agents*.
- Huang, H-M. 1996. An Architecture and A Methodology for Intelligent Control. *IEEE Expert: Intelligent Systems and Their Applications*, 11(2): 46-55.
- Jennings, N. R. 1995. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2): 195-240.
- Jung, D. and A. Zelinsky. 2000. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots Journal*, 8(3).
- Kaelbling, L. P. 1988. Goals as parallel program specifications. *Proceedings of the Seventh National Conference on Artificial Intelligence*. 60-65. Menlo Park, CA : AAAI Press/MIT Press.
- Kaelbling, L. P., A. R. Cassandra, and J. A. Kurien. 1996. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Kaelbling, L. P., and S. Rosenschein. 1991. Action and Planning in Embedded Agents. *Designing Autonomous Agents*. Edited by P. Maes, pp. 35-48. Cambridge, MA: MIT Press.

- Kalman, R.E.. 1960. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, 82:35-45.
- Kennedy, J., and R. C. Eberhart. 2001. *Swarm Intelligence*. San Francisco, CA: Morgan-Kaufmann Publishers.
- Khatib, O. 1985. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 500-505.
- Khoo, A., G. Dunham, N. Trienens, S. Sood. 2002. Efficient, Realistic NPC Control Systems using Behavior-Based Techniques. *AAAI tech. report SS-02-01*, Menlo Park, CA: AAAI Press.
- Khoo, A. and R. Zubek. 2002. Applying Inexpensive AI Techniques to Computer Games. *IEEE Intelligent Systems*, Special Issue on Artificial Intelligence in Infotainment, 17(4).
- Kiat, N. B., Q. Y. Ming, T. B. Hock, Y. S. Yee and S. Koh. 2001. LuckStar II: Team Description Paper. In *RoboCup 2001: Robot Soccer World Cup V*. Lecture Notes in Computer Science 2377, pp. 22-25. Edited by A. Birk, S. Coradeschi, S. Tadokoro. Springer-Verlag.
- Kinny, D., M. Ljungberg, A. S. Rao, E. A. Sonenberg, G. Tidhar, and E. Werner. 1992. Planned Team Activity, *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*.
- Kitano, M., M. Asada, Y. Kunoyoushi, I. Noda, E. Osawa. 1997. Robocup: The robot world cup initiative. *Proceedings of Autonomous Agents 97*. Marina Del Rey, CA.
- Kraus, S. 2001. *Strategic Negotiation in Multiagent Environments*. Cambridge, MA: MIT Press.
- Krogh, B. 1984. A Generalized Potential Field Approach to Obstacle Avoidance Control. SME-RI Technical Paper MS84-484, Society of Manufacturing Engineers, Dearborn, Michigan.
- Kube, C. R. and E. Bonabeau. 2000. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1-2): 85-101.

- Kube, C. R. and H. Zhang. 1992. Collective Robotic Intelligence. *Second International Conference on Simulation of Adaptive Behavior*: 460-468.
- Kube, C. R. and H. Zhang. 1994. Stagnation Recovery Behaviours for Collective Robotics. *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pp. 1893-1890.
- Kuhn, H. W. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1): 83-97.
- Kuniyoshi, Y., S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. 1994. Cooperation by Observation: The framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation*: 767-774.
- Latombe, J-C. 1991. *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers.
- Lefebvre, D. and G. Saridis. 1992. A Computer Architecture for Intelligent Machines. *Proceedings of the IEEE International Conference in Field, Factory and Space (CIRFFSS 94)*, pp. 842-849.
- Levesque, H. J. and R. J. Brachman. 1985. A fundamental tradeoff in knowledge representation and reasoning (revised edition). *Readings in Knowledge Representation*. Edited by R. J. Brachman and H. J. Levesque, pp. 42-70, Los Altos, CA: Morgan Kaufman.
- Lin, C. R. and M. Gerla. 1997. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265-1275.
- Lumia, R. 1994. Using NASREM for Real-Time Sensory Interactive Robot Control. *Robotica*, 12: 127-135.
- Lyons, D. and A. Hendriks. 1992. Planning for Reactive Robot Behavior. *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2675-2680.
- MacKenzie, D. C. 1997. *A Design Methodology for the Configuration of Behavior-Based Robots*. PhD Dissertation. Georgia Institute of Technology Tech Report GIT-CS-97/01.

- MacKenzie, D. C., J. Cameron and R. Arkin. 1995. Specification and Execution of Multiagent Missions. *Proceedings of the International Conference on Intelligent Robotics and Systems (IROS '95)*, pp. 51-58.
- Maes, P. 1989. *How to do the right thing*. AI Memo 1180, MIT Artificial Intelligence Laboratory.
- Maes, P. 1990. Situated Agents Can Have Goals. *Robotics and Autonomous Systems*, 6: 49-70.
- Mataric, M. 1992. Designing emergent behaviors: From local interactions to collective intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*: 432-441. Edited by J. Meyer, H. Roitblat, and S. Wilson. Cambridge, MA: MIT Press
- Melhuish, C. R., O. E. Holland, and S. E. J. Hoddell. 1998. Collective sorting and segregation in robots with minimal sensing, *Proceedings of the 5th International Conference on Simulation of Adaptive Behaviour*.
- Minsky, M. 1986. *The Society of Mind*. New York, NY: Simon and Schuster.
- Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
- Nilsson, N. 1994. Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research*. 1: 139-158.
- Parker, L. E. 1994a. *Heterogeneous Multi-Robot Cooperation*. PhD Dissertation, Massachusetts Institute of Technology.
- Parker, L.E. 1994b. ALLIANCE: An Architecture for Fault Tolerant, Cooperative Control of Heterogeneous Mobile Robots. *Proceedings of the 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS '94)*: 776-783.
- Parker, L.E. 1996. Task-Oriented Multi-Robot Learning in Behavior-Based Systems. *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '96)*: 1478-1487.
- Parker, L.E. 1997. Cooperative Motion Control for Multi-Target Observation. *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '97)*: 1591-1598.

- Parker, L. E. 1998. ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2): 220-240.
- Pynadath, D. V. and M. Tambe. 2002. The Communicative Multiagent Team Decision Problem: Analyzing Teamwork Theories and Models. *Journal of Artificial Intelligence Research*, 16:389-423.
- Rabiner, L.R. and B.H. Juang. 1986. An introduction to hidden markov models. *IEEE ASSP Magazine*.
- Real World Interface (RWI). 1999. *Magellan Compact Mobile Robot User's Guide*. Jeffrey, New Hampshire: Real World Interface.
- Rosenschein, S. J. and L. P. Kaelbling. 1986. The synthesis of machines with provable epistemic properties. *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*. Edited by J. Halpern. 83-98. Morgan Kaufman.
- Rosenschein, S. J. and L. P. Kaelbling. 1995. A Situated View of Representation and Control. *Artificial Intelligence*. 73(1-2).
- Rosenblatt, J. 1995. DAMN: A Distributed Architecture for Mobile Navigation. *AAAI Technical Report SS-95-02*, 1995 Spring Symposium Series: Lessons Learned from Implemented Software Architectures for Physical Agents, pp. 167-178, Palo Alto, CA: AAAI Press.
- Roth, M., D. Vail and M. Veloso. 2003. A world model for multi-robot teams with communication. *Proceedings of the IEEE International Conference on Robotics and Automation 2003*. Taipei, Taiwan. Under submission.
- Royer, E. and C. K. Toh. 1999. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, *IEEE Personal Communications Magazine*, 6:46-55.
- Russell, S. and P. Norvig. 1995. *Artificial Intelligence : A Modern Approach*. Engelwood Cliffs, NJ: Prentice-Hall.
- Shastri, L. and V. Ajjanagadde. 1993. From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*. 16(3): 417-494.

- Simmons, R., S. Singh, D. Hershberger, J. Ramos, and T. Smith. 2000. First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly, *Proceedings of the International Symposium on Experimental Robotics*.
- Simmons, R., T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, R. Zlot. 2002. A Layered Architecture for Coordination of Mobile Robots. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, Edited by A. Schultz and L. Parker, Kluwer Academic Publishers.
- Steels, L. 1990. Cooperation between distributed agents through self-organization. In *Decentralized A. I.* Edited by Y. Demazeau and J. Muller. Elsevier-Science.
- Steels, L. 1998. Synthesising the origins of language and meaning using co-evolution, self-organisation and level formation. In *Approaches to the evolution of language: 384–404*. Edited by J. R. Hurford, M. Studdert-Kennedy & C. Knight. Cambridge: Cambridge University Press.
- Streenstrup, M. 2000. Cluster-Based Networks. In *Ad Hoc Networking*. Edited by C. E. Perkins. Addison-Wesley.
- Tambe, M. and P. S. Rosenbloom. 1995. RESC: An approach for Real-Time, Dynamic Agent Tracking. *Proceedings of the International Joint Conference on Artificial Intelligence*: 103-111.
- Tambe, M. 1996a. Teamwork in real-world, dynamic environments. *Proceedings of the Second International Conference on Multi-Agent Systems*. Menlo Park, Ca: AAAI Press.
- Tambe, M. 1996b. Tracking Dynamic Team Activity. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Thrun, S. 2000. Probabilistic Algorithms in Robotics. *AI Magazine*. 21:4, pp. 93-109.
- Vail, D. and M. Veloso. 2003. Dynamic Multi-Robot Coordination. *Multi-Robot Systems Volume II: From Swarms to Intelligent Automata*, edited by A. Schultz, L. Parker, and F. Schneider. Kluwer Academic Publishers.
- Veloso, M., M. Bowling, S. Achim, K. Han, and P. Stone. 1999. The CMUnited-98 Champion Small Robot Team. In *RoboCup-98: Robot Soccer World Cup II*, edited by M. Asada and H. Kitano. Berlin: Springer Verlag.

- Vogt, P. 2000. *Lexicon Grounding on Mobile Robots*. PhD Dissertation. Vrije Universiteit Brussel.
- Wasson, G., D. Kortenkamp and E. Huber. 1999. Integrating active perception with an autonomous robot architecture. *Robotics and Autonomous Systems*, 26: 175-186.
- Wie, M. V. 2000. A probabilistic method for team plan formation without communication. *Proceedings of the Fourth International Conference on Autonomous Agents*: 112-113.
- Xylomenos, G. and G.C. Polyzos. 1999. Internet Protocol Performance over Networks with Wireless Links. *IEEE Network* 13: 55-63.