A Decentralized Architecture for Active Sensor Networks

Alexei A. Makarenko

A thesis submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy

Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

November 2004

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Alexei A. Makarenko

November 17, 2004

Abstract

Alexei A. Makarenko
The University of Sydney

Doctor of Philosophy
November 2004

A Decentralized Architecture for Active Sensor Networks

This thesis is concerned with the Distributed Information Gathering (DIG) problem in which a Sensor Network is tasked with building a common representation of environment. The problem is motivated by the advantages offered by distributed autonomous sensing systems and the challenges they present. The focus of this study is on Macro Sensor Networks, characterized by platform mobility, heterogeneous teams, and long mission duration. The system under consideration may consist of an arbitrary number of mobile autonomous robots, stationary sensor platforms, and human operators, all linked in a network.

This work describes a comprehensive framework called Active Sensor Network (ASN) which addresses the tasks of information fusion, decision making, system configuration, and user interaction. The main design objectives are scalability with the number of robotic platforms, maximum flexibility in implementation and deployment, and robustness to component and communication failure. The framework is described from three complementary points of view: architecture, algorithms, and implementation.

The main contribution of this thesis is the development of the ASN architecture. Its design follows three guiding principles: decentralization, modularity, and locality of interactions. These principles are applied to all aspects of the architecture and the framework in general. To achieve flexibility, the design approach emphasizes interactions between components rather than the definition of the components themselves. The architecture specifies a small set of interfaces sufficient to implement a wide range of information gathering systems.

In the area of algorithms, this thesis builds on the earlier work on Decentralized Data Fusion (DDF) and its extension to information-theoretic decision making. It presents the Bayesian Decentralized Data Fusion (BDDF) algorithm formulated for environment features represented by a general probability density function. Several specific representations are also considered: Gaussian, discrete, and the Certainty Grid map. Well known algorithms for these representations are shown to implement various aspects of the Bayesian framework.

As part of the ASN implementation, a practical indoor sensor network has been developed and tested. Two series of experiments were conducted, utilizing two types of environment representation: 1) point features with Gaussian position uncertainty and 2) Certainty Grid maps. The network was operational for several days at a time, with individual platforms coming on and off-line. On several occasions, the network consisted of 39 software components. The lessons learned during the system's development may be applicable to other heterogeneous distributed systems with data-intensive algorithms.

Acknowledgements

I would like to begin by thanking Professor Hugh Durrant-Whyte for his guidance and encouragement throughout this work. Thanks Hugh, you are my favorite kind of advisor.

This work was funded largely through the Australian Research Council's IPRS scholarship and I gratefully acknowledge its support.

Many thanks to the entire ACFR team, past and present, for their support in the field of robotics and otherwise. I owe a special thank you to Dr. Ben "This-is-nonsense" Grocholsky for research ideas, controversial statements, and tours of Sydney. Thanks to Stef and Salah for discussions, field trips, and for letting me squat in your offices. To Matt for often-needed expert advice. To Eric, Jeff, and Tomo for our common struggle through the lab's demographic gap. To Sharon for the chocolate. To Ben and Tobie for the help in making this thesis readable. To Fred for 150,000 miles travelled together and for finding Australia on the map in the first place. To Tobie, Alex Br., Alex Ba., and Martin for sharing a few hundred midnight alarms.

I must thank many people outside of ACFR who made my stay in Sydney as good as it was. The 2001 international student draft was an exceptional group: Joey, Cath, Udo, Stan, Paul, and Christine — it was not the same afterwards. Many thanks to the SERC crowd back stateside who never came to visit but let me visit them. Personally, to Dr. Eric "Yeh-baby-yeh" Prechtl who showed me how to think like a doctor. To Dr. Brian "How-hard-can-it-be" Bingham for always having a plan. Collectively, to the ex-residents of Krazy Haus^{TM} for keeping the legend alive, even if on the West Coast.

Finally, this work would hardly be possible without the support of Newtown and Camperdown businesses who were there when I needed them most. To Happy Chef for the Emperor's Crispy Chicken (#37, $7.50). To Rose, Random, Royal, Landsdown and other hotels for being close and open. Special thanks to team Zanzibar for the never-ending entertainment. I would also like to thank the process of globalization for the miracle of around-the-world tickets.

I owe a lot to my parents for just the right kind of nature and nurture.

Thank you C. and danke Aldi.

Инженерам в моей семье:
деду, дяде и конечно
родителям.

Contents

List of Figures

List of Tables

List of Examples

Nomenclature

Notation

$(\cdot)^T$	Matrix transpose
$(\cdot)^{-1}$	Matrix inverse
$\hat{(\cdot)}$	Estimate of $(\cdot)$
$\tilde{(\cdot)}$	Partial estimate of $(\cdot)$
$(\cdot)^k$	$(\cdot)$ at discrete time t_k
$(\cdot)^k$	A set $(\cdot)$ from current time up to time t_k
$< \ldots >$	Ordered set
$a \triangleq b$	a is defined as b
$\rightharpoonup$	Implies

Platform Subscripts

$(\cdot)_i$	$(\cdot)$ pertaining to platform i
$(\cdot)_j$	$(\cdot)$ pertaining to platform $i \neq j$
$(\cdot)_{-i}$	$(\cdot)$ pertaining to all platforms other than i
$(\cdot)_\alpha$	$(\cdot)$ pertaining to all platforms in team α

General Symbols

a	Action
$\mathbf{a}$	Action vector
A	Action set
$\mathcal{A}$	Action space
b	Belief
B	Belief set
$\mathcal{B}$	Belief space
$\mathbf{B}$	Platform state influence matrix
D	Feature state belief update function
$E\{\cdot\}$	Expected value of $(\cdot)$

$\mathcal{E}n(\cdot)$	Entropy
F	Platform state transition function
$\mathbf{F}$	Platform state transition matrix
F'	Platform state belief prediction function
G	Feature state transition function
$\mathbf{G}$	Feature state transition matrix
G'	Feature state belief prediction function
H	Sensor model
$\mathbf{H}$	Observation matrix
$\mathbf{i}$	Information gain vector
$\mathbf{I}$	Information gain matrix
$L(\cdot)$	Likelihood
$\mathcal{I}(\cdot)$	Mutual information
J	Expected utility function
m	Message
M	Message set
$\mathcal{M}$	Message space
p	Platform state
$\mathcal{P}$	Platform state space
$\mathbf{P}$	Covariance matrix
$P(\cdot)$	Probability distribution
π	Policy
$\mathbf{Q}$	Process noise matrix
ρ	Reward
R	Reward set
$\mathbf{R}$	Sensor noise matrix
s	System state, including states of features and platforms
$\mathcal{S}$	System state space
t	Time
u	Feature input
U	Utility function
$\mathbf{v}$	Sensor noise
V	Value
$\mathbf{w}$	Process noise
x	Feature state
$\mathbf{x}$	State vector, including states of features and platforms
$\mathcal{X}$	Feature state space
$\mathbf{y}$	Information vector
$\mathbf{Y}$	Information matrix
z	Observation
$\mathbf{z}$	Observation vector

Z	Observation set
$\mathcal{Z}$	Observation space

Typefaces

My Use Case	DIG use case
my role	ASN component role
MYCOMPONENT	ASN component type
Myinterface	ASN interface
`MyComponent`	Executable component name

Abbreviations

ASN	Active Sensor Network
BDDF	Bayesian Decentralized Data Fusion
BDDM	Bayesian Decentralized Decision Making
CBSE	Component Based Software Engineering
CG	Certainty Grid
CI	Covariance Intersect
DIG	Distributed Information Gathering
DDF	Decentralized Data Fusion
DDM	Decentralized Decision Making
GUI	Graphical User Interface
IF	Information filter
IR	Infra-red
KF	Kalman filter
MDP	Markov Decision Process
MRS	Multi-Robot Systems
MSN	Macro Sensor Networks
PDF	probability density function
PTZ	Pan-tilt-zoom (camera)
QoS	Quality of Service
SLAM	Simultaneous Localization and Mapping
SN	Sensor Network
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UI	User Interface
UML	Unified Modeling Language
μSN	Micro Sensor Networks
WSN	Wireless Sensor Network

Chapter 1

Introduction

The aim of this thesis is to develop a software framework for *scalable, autonomous, and cooperative* information gathering. Figure 1.1 illustrates the vision of a system which this work tries to bring about. The figure shows a communication network formed by mobile autonomous robots, stationary sensor platforms, and human operators. The environment contains a spatially-distributed dynamic phenomenon which has to be measured, located, mapped, or simply monitored. The system's objective is to build a common representation of the phenomenon. This task will be called the problem of Distributed Information Gathering (DIG).

The term *scalable* implies that the desired solution must be able to deploy an arbitrary number of platforms like the ones shown in the figure. The term *autonomous* here means that the system should form a network automatically and easily adapt to the changing environmental conditions. The term *cooperative* here means that, while able to function independently, platforms have common objectives and work together to achieve them when they can.

1.1 Motivation

The DIG vision statement paints a picture of a complicated system. To justify the development effort, some of the stated objectives need to be motivated.

Why sense the environment? In many applications, decisions must be made which depend

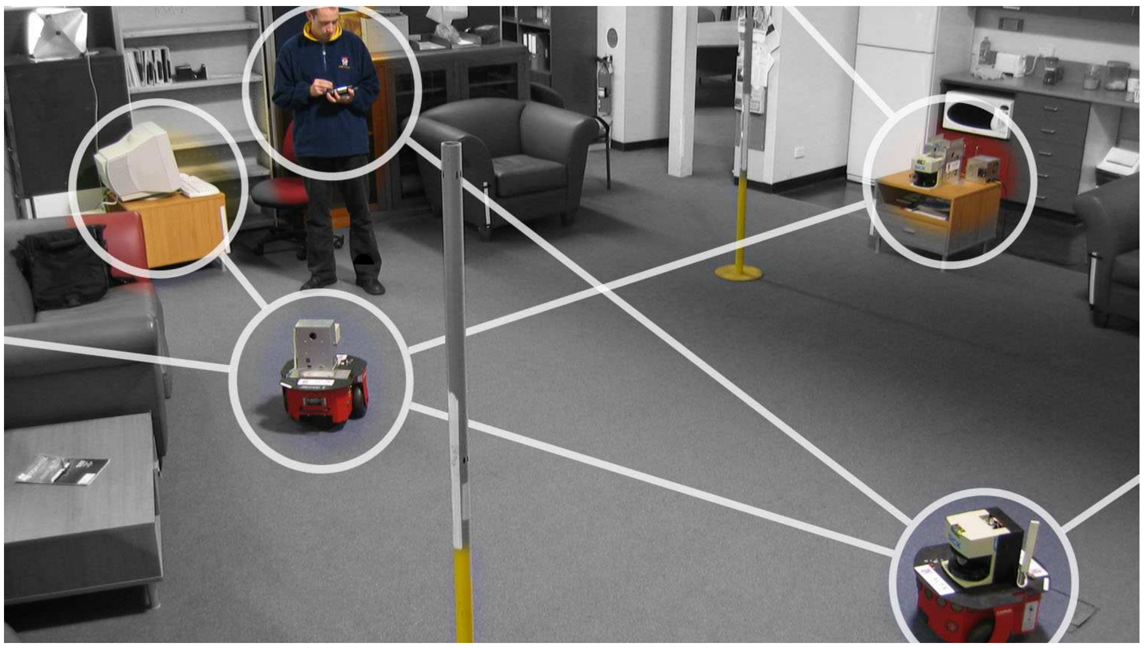


Figure 1.1: The vision of interacting autonomous sensing platforms gathering information about the environment. The image is taken from one of the experiments described later in the thesis. Although illustrated here with an indoor system, the concept is applicable to outdoor applications as well.

on the current state of the environment. Environment information can only be obtained by making observations. This applies to both human and robotic decisions. In many applications, the success depends on the accurate and timely information about the environment. These include transportation, experimental science, intelligent spaces, military and security, natural disaster management, warehousing and manufacturing, and others. While humans are still vastly better than autonomous systems at making decisions in complex domains, collecting information to support the decision making process is a manageable and well-defined task for a robotic system.

Why active platforms? This thesis will consider platforms which are active, i.e. have actuators. Means of locomotion is the most common type of actuator. Benefits of using mobile sensing platforms include improved coverage, adaptiveness to changing and unpredictable environment, the ability to overcome occlusions and inaccessible terrain, and more opportunities for replenishing energy supplies.

Why distributed systems? There are many situations when the designer has no choice and the solution must be distributed, for example if distributed hardware already exists. In other cases, distributed and monolithic solution options must be compared. At the cost of added complexity, robotic systems with multiple sensors (and, in a dual argument, actuators) offer the following advantages [46, 54, 135]:

1. Redundancy: existence of multiple resources suitable for making observations (performing actions).
2. Complementarity in sensing (actuation): multiple sensors (actuators) may perceive features (perform actions) which are impossible by each individual sensor (actuator) operating separately.
3. Timeliness: ability to obtain near-simultaneous measurements (perform actions) at multiple spatially distributed locations.
4. Flexibility: capacity for local adaptation to local changes in the environment.

Modern high-technology systems, including robotic systems, are increasingly driven by software.[1] With this in mind, it is worth stepping outside the field of robotics and note the reasons why distributed software architectures are increasingly adopted in the wider computing community. Emmerich [49] lists the following advantages of distributed software:

1. Scalability: can accommodate growing load in the future
2. Openness: can easily be extended and modified
3. Heterogeneity: in programming languages, operating systems, hardware, network protocols
4. Resource sharing
5. Fault-tolerance.

Why Macro Sensor Networks? Distributed sensing systems are often referred to as Sensor Networks (SN).[2] This thesis introduces the term *Macro Sensor Network* for a SN in which

[1] "The only thing you can do with an F-22 that does not require software is to take a picture of it", attributed to Lt. Gen. Jim Fain, while F-22 Program Director [134].

[2] There is a variety of derived terms, from arguably redundant Distributed Sensor Networks to Wireless Sensor Networks.

the platforms are not inherently constrained to small physical size. It is the argument of this thesis, that this constraint has a profound effect on the available solution options and the SN capabilities, making MSN's better suited for certain applications.

What is an architecture and why architect? The dictionary defines architecture as "the manner in which the components of a system are organized and integrated" [98]. Before defining architecture, Clements gives a very pragmatic answer to the question of *why architect* by describing a typical design process of a complex system [32]:

> Architecting is "divide and conquer",
> followed by "now mind your own business",
> followed by "so how do these things work together?"

He argues that success in the first and the last steps of the design process *requires* an architecture.[3] One would argue that a separate discussion of the architecture is only warranted when the system in question is large and complex. According to these criteria, SN's certainly qualify as they incorporate a large number of interacting hardware elements, algorithms, developers, and users.

Why decentralized architecture? A proper definition of decentralization will be given later in the thesis, but the main idea is that a decentralized system is a network of peers with no part of the system playing a special commanding role. The decentralized approach is considered in this context because it leads to properties which are important to SN's such as the ability to maintain performance levels as the size of the network grows and robustness to failure of individual subsystems.

Why modular, component-based architecture? The basic idea of modular design is to take a large problem and decompose it into subproblems with the hope that the subproblems are easier to solve than the original problem. In software development, the simplification comes from the fact that different components can be developed in "mutual ignorance" [224]. The second reason for adopting component-based design is *software reuse.* The premise is that capability of a complex system can be increased incrementally by adding to the system already in place.

[3]Szyperski et al. draw attention to the same two features of any architecture by pointing out that it is "simultaneously the basis for independence and cooperation" [224].

The application scope for SN's is very diverse. Some of the specific SN applications reported in the literature include habitat sensing [28], seismic, traffic and home monitoring [127, 152], smart home [121, 155], environmental control in office buildings [246], industrial sensing and diagnostics [78], health monitoring [14], battle-field awareness [36, 45, 90, 151, 221], fire fighting [41, 87], competitive games [43, 235], surveillance [42, 181, 196, 216], wide area search [21, 94, 114], planetary exploration [95], widespread weather and environmental sampling [143, 229].

1.2 Related Fields

Several theoretical and practical research field are closely related to Active Sensor Networks. An illustration is given in Figure 1.2. The fields are briefly mentioned below and a thorough review of related work is conducted in Chapter 2.

Bayesian Filtering and Communication. The probabilistic framework is the foundation of the current approach. It is chosen to account for the uncertainty inherent in all interactions of a robotic system with the real world. Bayesian information fusion techniques are used to combine sensor observations. In distributed systems, the question of what information to communicate becomes important. This is the subject of Bayesian Communication [67, 68] which gives the answer in terms of common information.

Information Theory. To make sensor management decisions, it is often valuable to measure the amount of information contained in a given probability distribution [37].[4] Specific areas of intersection include quantitative measures of information quality and the expected information gain associated with different observation options.

Statistical Decision Theory. Statistical decision theory is concerned with making decisions in the presence of uncertainties [16]. The concept of expected utility of actions helps in making plans.

Team Theory. Several decision makers with the same objective make up a team. Marschak and Radner originated the study of decentralized decision making in economic systems [146, 147]. Team theory in the context of optimal control was developed by Ho et al.

[4]Even the design of Figure 1.2 is patterned after this influential text.

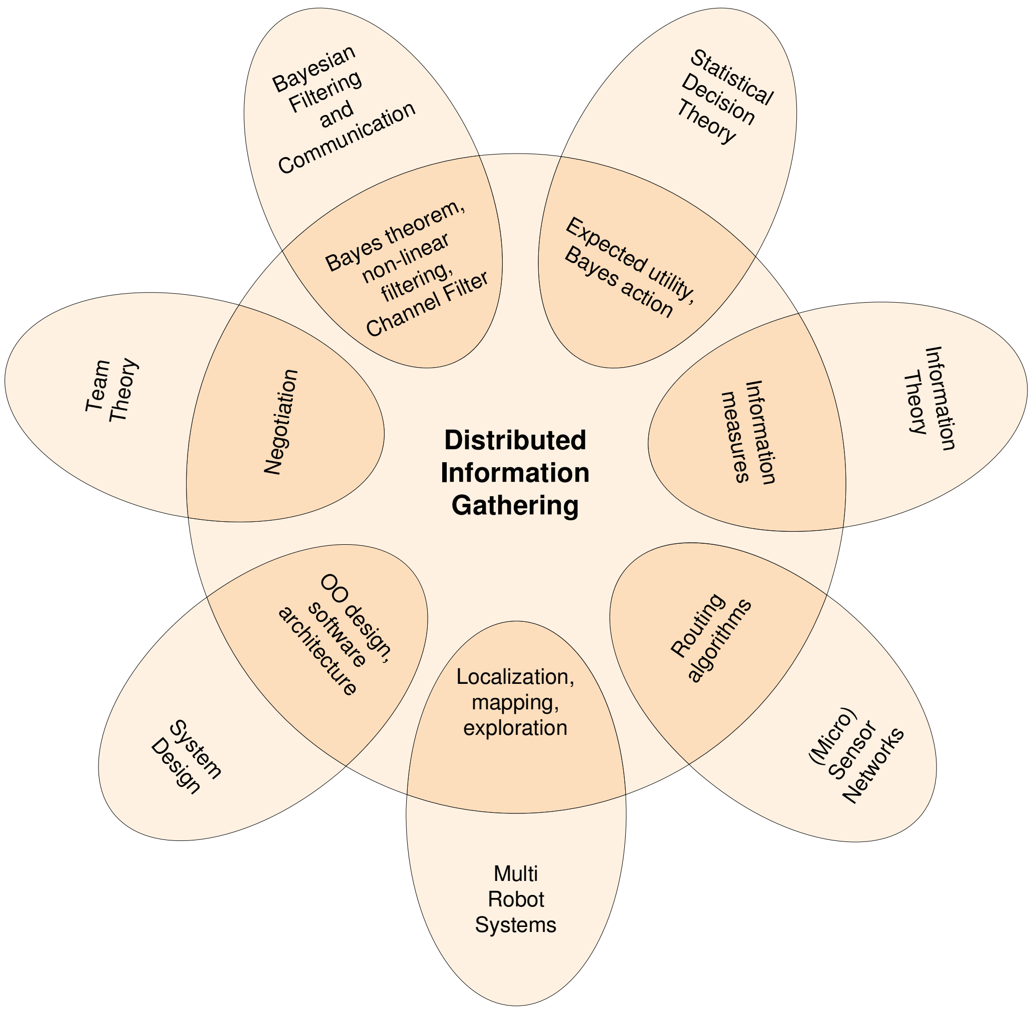

Bayesian Filtering and Communication
Statistical Decision Theory
Bayes theorem, non-linear filtering, Channel Filter
Expected utility, Bayes action
Team Theory
Negotiation
Distributed Information Gathering
Information measures
Information Theory
OO design, software architecture
Routing algorithms
System Design
Localization, mapping, exploration
(Micro) Sensor Networks
Multi Robot Systems

Figure 1.2: Research areas related to Active Sensor Networks.

[82, 83, 84, 85]. The formulation presented in this work is based on the more recent work of Pynadath and Tambe [178].

Component-Based Software Engineering. In describing the solution framework, this thesis uses the methods of software architecture design [13, 88, 185] and documentation [32]. This research community has produced the Unified Modelling Language (UML) [75], a graphical notation for describing object-oriented system which is used extensively in this thesis. Component-Based Software Design [76] is an integral part of the implementation strategy adopted in this work.

Multi-Robot Systems. In the experimental validation, this work relies on the existing algorithms developed by the robotics community. These include algorithms for localization, mapping, exploration, obstacle avoidance and path planning.

Micro Sensor Networks. Typical sensing platforms used in this field are physically smaller and have less energy storage compared to those considered in this thesis. Dictated by fundamental differences in hardware infrastructure, the solution approaches remain largely orthogonal but there are ample opportunities for cross-fertilization. Some of the routing algorithms originated in this community are discussed for comparison.

1.3 Thesis Objectives

The objectives of the thesis fall into one of three categories: architectural, algorithmic, and implementation. In the area of architecture, the goal of this work is:

1. To develop a coherent and practical breakdown of networked components suitable for a modular implementation.

Algorithms follow the functional breakdown of the system: information fusion and decision making. In terms of algorithms, the goals of this work are:

2. To extend the existing information fusion and decision making algorithms from Gaussian to general Bayesian distributions.

In terms of implementation, the goal of this work is:

3. Validate the approach experimentally.

1.4 Thesis Contributions

The contributions of this thesis are:

- Statement of the Distributed Information Gathering problem in descriptive and probabilistic terms

- Formulation of a fully decentralized and scalable framework for Active Sensor Networks, suitable for implementing a wide range of information gathering tasks.
- Extension of the Decentralized Data Fusion (DDF) algorithm to the general Bayesian case.
- Application of DDF approach to the Certainty Grid algorithm.
- Extension of the Decentralized Decision Making (DDM) algorithms to the general Bayesian case.
- Development and implementation of the Integrated Exploration algorithm with Certainty Grids.
- A concrete implementation of the Active Sensor Network (ASN) framework and its experimental validation on an indoor sensor network.

1.5 Thesis Structure

The thesis starts by defining the DIG problem and proceeds to develop the ASN framework which solves it. Figure 1.3 shows how the individual chapters contribute to describing the three aspects of the solution.

Chapter 2 describes the DIG problem. The problem is first defined in a system-theoretic sense: by listing functional and non-functional requirements. Then a probabilistic reference model is presented which is general enough to cover a broad spectrum of possible solutions. The reference model establishes the structure of the problem and solution spaces which helps to describe and classify related work.

Chapter 3 is central in many ways. It defines the ASN framework and describes its architecture. It justifies the high-level decisions which define the solution approach. The algorithmic and implementation structure of the solution is also described.

Chapters 4 and 5 focus on the algorithms used to implement the information fusion and decision making tasks respectively. Both chapters first define general Bayesian algorithms and then specialize them to specific probabilistic representations. A number of existing algorithms are referenced and shown to be special cases of the general approach.

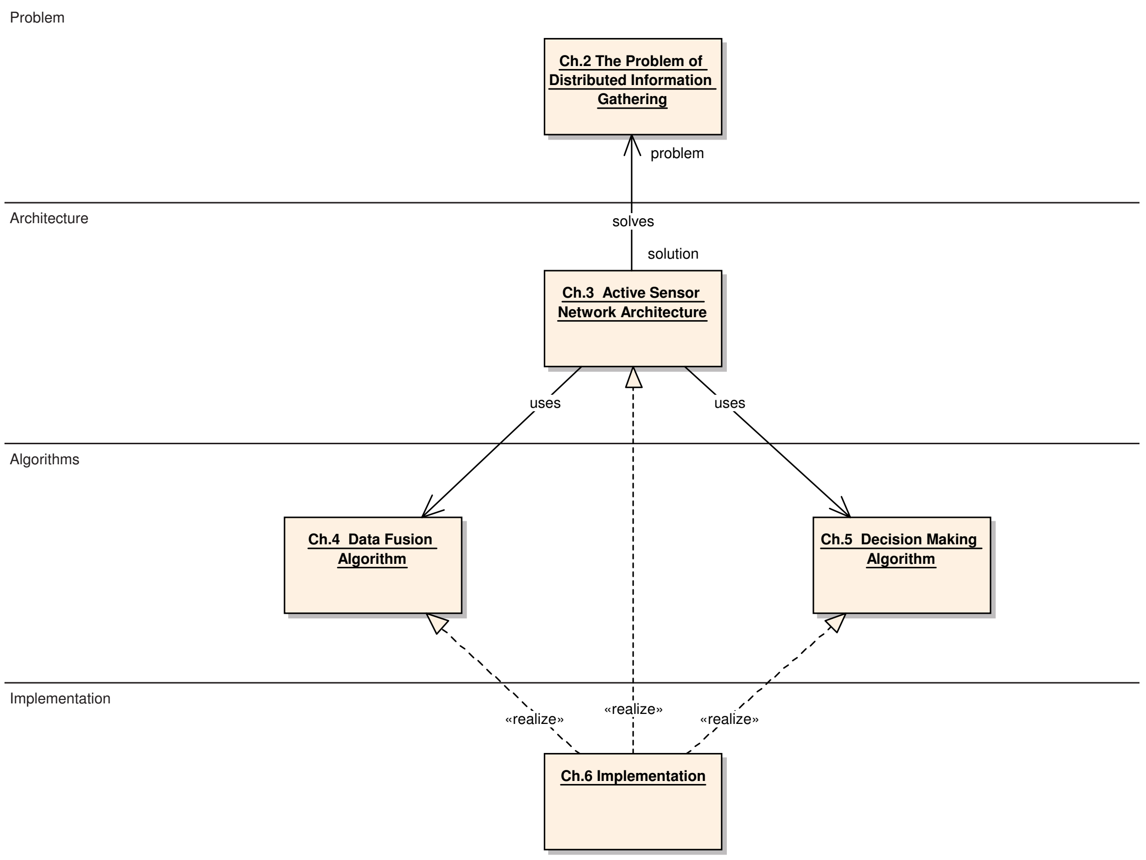

Problem
Ch.2 The Problem of Distributed Information Gathering
problem
Architecture
solves
solution
Ch.3 Active Sensor Network Architecture
uses
uses
Algorithms
Ch.4 Data Fusion Algorithm
Ch.5 Decision Making Algorithm
Implementation
«realize»
«realize»
«realize»
Ch.6 Implementation

Figure 1.3: Thesis breakdown into chapters. The architecture ties together the information fusion and decision making algorithms. The framework implementation instantiates the architecture and the algorithms. The figure is drawn using Unified Modelling Language (UML) notation which is reviewed later in the thesis. A reader not familiar with UML may view this particular figure as a simple block diagram.

Chapter 6 describes the concrete implementation used in illustrating the operation of the system throughout the thesis.

Finally, Chapter 7 draws conclusions and proposes a number of future directions for research in active sensor networks.

Note, that while the thesis includes experimental validation of the ASN framework, there is no “Results” chapter. The implementation is multi-faceted and largely follows the thesis structure. For this reason, it is more natural to describe the experimental results immediately after the theory is presented. Two groups of experiments were conducted, categorized according to the environment representation: 1) point features with Gaussian position un-

System Aspect		Representation	
		Gauss Point Features	*Certainty Grid Map*
Architecture		Chapter 3	
Algorithms	*Information fusion*	Section 4.3.3	Section 4.4.5
	Decision making	Section 5.3.3	Section 5.4.3
Implementation	*Components*	Section 6.3.1	
	Deployment	Section 6.3.5	

Table 1.1: Locating the description of experimental results.

certainty and 2) Certainty Grid maps.

Table 1.1 shows how to find the description of the two series of experiments. Both experimental systems use the ASN architecture described in Chapter 3. The results related to the task of information fusion are presented in two sections of Chapter 4 as shown in the Table 1.1. The results related to the task of decision making are presented in two sections of Chapter 5. Many individual components are shared between the two systems due to the modular design of the architecture. Implementation of all components is discussed in Section 6.3.1. Details of deployment of the two systems are discussed in Section 6.3.5.

Chapter 2

The Problem of Distributed Information Gathering

2.1 Introduction

This chapter presents the Distributed Information Gathering (DIG) problem. This is a *systems problem* and the problem statement must be sufficiently general to describe all of its aspects. It must also be be sufficiently detailed to identify the desired properties of the solution and to allow meaningful description and classification of related work.

Section 2.2 defines the problem in a system-theoretic sense: by listing functional and non-functional requirements. To allow quantitative description of the design space, several useful metrics are defined. They describe the environment and the sensing infrastructure independent of the solution approaches.

A probabilistic reference model is defined in Section 2.3. It is general enough to cover a broad spectrum of possible solutions. Based on this model, the solution space is categorized by listing a number of options available to the designer including five key design choices: system architecture, information fusion method, utility evaluation and fusion method, policy selection method, and system implementation. The extensive notation introduced by the reference model is also used in the following chapters to describe the algorithms and the structural breakdown of the proposed architecture.

Related work is described in Section 2.4 starting with a basic taxonomy of the field based on several quantitative metrics. As a result, three types of systems are identified: Micro Sensor Networks, Macro Sensor Networks and Multi-Robot Systems. Typical examples from the literature are classified into these three categories and reviewed with an emphasis on the five aspects of the solution listed in Section 2.3. The work related to Macro Sensor Networks is described in more detail because the framework proposed in this thesis targets this SN category.

2.2 Problem Description

This section describes the DIG problem from the system-theoretic view point, i.e., what functionality is required, how should this functionality be achieved, and what is the relative importance of various aspects of the solution.

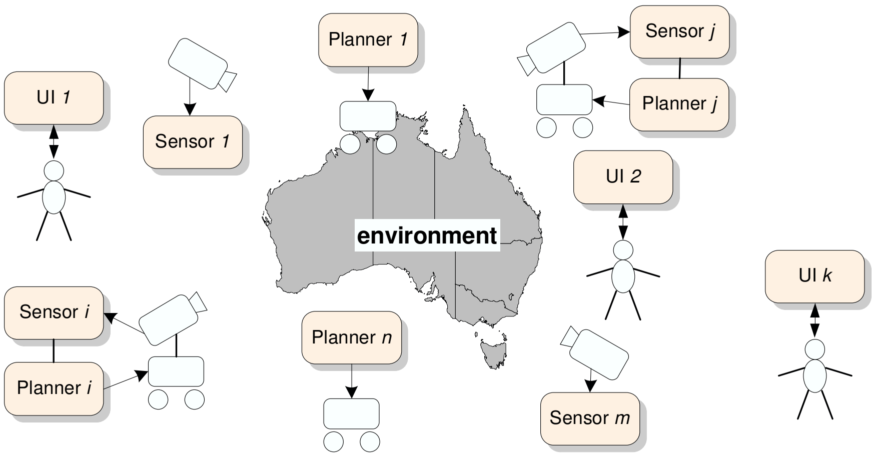


Figure 2.1: A conceptual view of the DIG problem shows the environment to be monitored, human operators, and robotic entities: sensors and actuators.The team objective is to obtain as much information about the environment as possible by forming a network, exchanging information, and coordinating actions. Sensor Network is formed by software: User Interfaces (UI) represent people, Sensor and Planner modules represent robotic hardware.

The problem setting is illustrated in Figure 2.1. The environment contains a spatially-distributed dynamic phenomenon which has to be measured, located, mapped, or simply monitored. There is a set of heterogeneous robotic platforms equipped with sensors and actuators. The platforms have means of communication and, together, form a Sensor Network (SN). Platforms with sensors can observe the environment and contribute information to the network. Platforms with actuators make decisions on what actions to take and what observations to make in the future. So far, the term *platform* has been used in a common sense of a “vehicle used for a particular activity or purpose” [98]. In this thesis, a platform is also synonymous with *decision maker* and *agent*.[1]

There is also a set of human operators interacting with the network through User Interfaces (UI). Operators can request information about the environment from the SN or contribute information about the environment by observing it directly using human senses.

All entities, human and robotic, are considered to be members of a team working towards a common goal: maximizing information quality about the environment. This could be a goal in itself or the collected information may serve as the basis for further actions. Such domain-specific extensions are not considered to be part of the DIG problem.

This section *specifies* the DIG problem as a set of requirements — an approach common in the field of software engineering [13, 139, 140]. Also borrowing from object-oriented software engineering, the Unified Modelling Language (UML) [75] is used extensively throughout the thesis. The UML notation is introduced as needed and brief explanations are given in footnotes specially marked with `UML` symbol. Note that, for a number of reasons, the UML models were not used for code generation.

The hierarchical breakdown of requirements is shown in Figure 2.2 using the UML *aggregation* relationship.[2] Functional requirements capture the intended behavior of the system, expressed as services, tasks, or functions the system is required to perform. Non-functional (or technical) requirements describe *how* and *how well* the functional requirements have to be fulfilled. Non-functional requirements include *constraints* and *qualities.* Qualities capture required properties of the system, such as performance. Constraints are not subject to negotiation and, unlike qualities, are off-limits during design trade-off analysis.

[1]Sensing platforms considered in this thesis are independent, proactive, self-describing, interactive, physically mobile — qualities typically attributed to agents [69, 206].

[2]`UML` An *aggregation* relationship shows that an element contains or is composed of other elements. Used in class models to show how more complex elements are built from a collection of simpler elements (for example a car from wheels, tires, motor, etc.) [223].

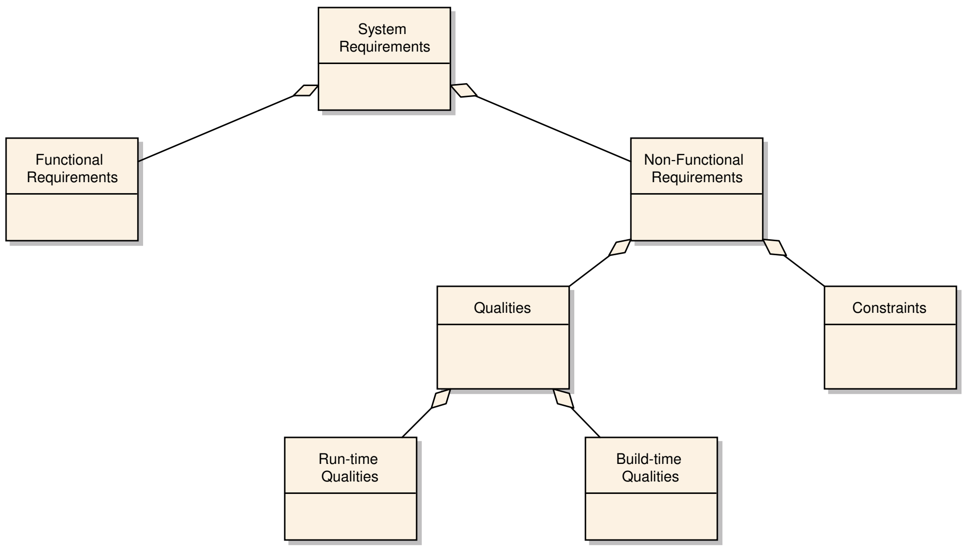


Figure 2.2: System specification using functional and non-functional requirements. *Run-time* qualities describe the final product, the executing system. *Build-time* qualities describe the work product, for example the code produced as part of the development process.

2.2.1 Functional Requirements

Functional requirements are documented with a UML *Use Case diagram.*[3] Figure 2.3 shows a use case diagram which specifies functional requirements for a system solving the DIG problem. The world is partitioned into three subsystems: the sensor network, the part of the environment which contains the phenomenon, and the rest of the world. The operation of the SN can be separated into two broad areas (use cases): collection of information and interaction with the operator. The diagram shows two *actors*[4]: the operator and the phenomenon. Both interact with the SN but are external to it. The operator may receive information about the environment through channels outside the SN, for example eyewitness report or the weather forecast. Note that the SN observes the environment but does not interact with it in any other way. For example, a SN with the task of bushfire tracking has to know where the fire is but does not have to put it out.

[3]`UML` A *use case* is a UML modelling element that describes how a user of the proposed system interacts with the system to perform a discrete unit of work. A *Use Case diagram* captures use cases and actor interactions. It describes the functional requirements of the system, the manner that outside things (actors) interact at the system boundary and the response of the system [223].

[4]`UML` An *actor* is anything that interacts with the system from the outside of the system boundary. Actors are not always human, the stick figure symbol notwithstanding.

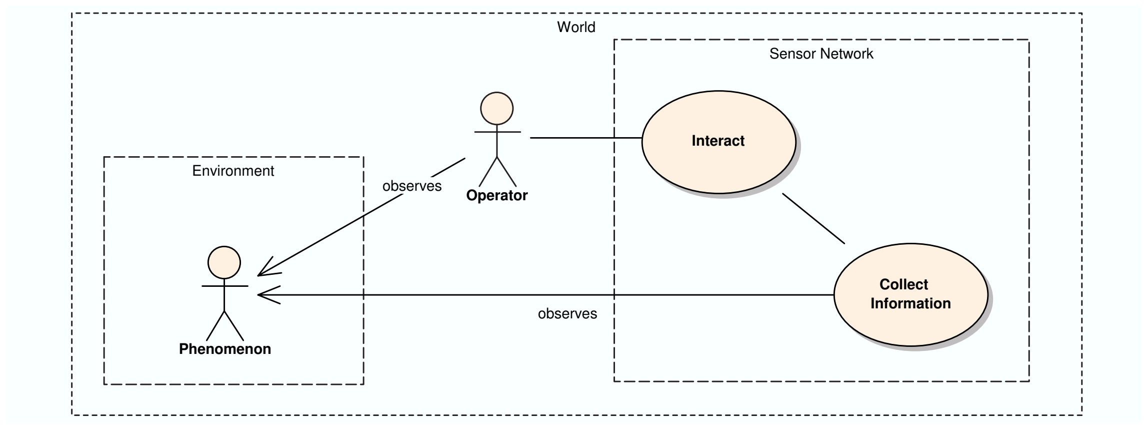


Figure 2.3: High level use case diagram for the DIG problem. The world is partitioned into the Sensor Network (SN), the environment which contains the phenomenon, and the rest. From the point of view of the SN, human operators and the environment are external entities.

A more detailed view of use cases is shown in Figure 2.4. The two large use cases are broken down into sub-tasks with the UML *include* connection[5]. The *Interact* use case is an umbrella case describing all operator-accessible functionality of the SN. It includes four subcases representing types of interactions. The *Request and Receive Info* use case is the core function envisioned by virtually all implemented and proposed SN's. Operators enter their interests and the network responds with information. The communication pattern is not specified, but the primary information flow in this case is from the network to the operator. The information flow is reversed in the *Enter Information* use case: independently-obtained information is supplied by the operator and the system combines it with the network's belief. On the decision making side, two modes of human-network interaction are specified: indirect control by specifying relative worth of various aspects of environment information (*Set Team Priorities* use case) and direct control by means of teleoperation (*Tele-operate* use case).

The *Collect Information* use case incorporates the core SN functionality. Perception is very important because, ultimately, all information available to the team is derived from observations. Fusion refers to the process of combining information obtained from different sources. Finally, the information must be delivered to consumers, typically to the decision makers, human or robotic. Robotic platforms make decisions based on the available information

[5]`UML` *Include* connections are used in use case models to show that one use case includes the behavior of another. [223]

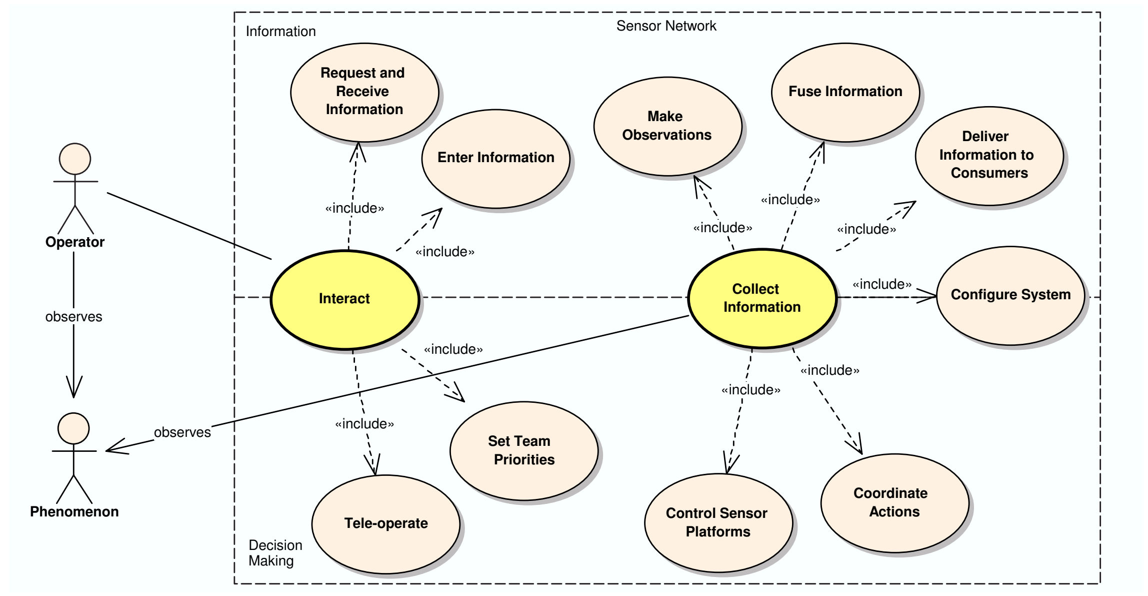


Figure 2.4: Detailed use case diagram for the DIG problem. The two large use cases are shown with the included sub-cases. Tasks related to information are shown on top, to decision making at the bottom.

and, in the team environment, performance can be improved if the actions of individual platforms are coordinated. Many, particulary large systems, need to be able to configure themselves on startup and reconfigure at runtime in response to the changing environmental conditions, changes within the team, etc.

The two use cases and ten subcases define the scope of the problem addressed by this thesis. Many systems described in the literature partially solve this problem or address only a particular aspect of the problem. For example, stationary platforms do not make decisions, while small-scale systems with reliable communications and stable team structure do not need to self-configure.

2.2.2 Design Constraints

The taxonomic divisions (and the corresponding metrics) discussed in this section are considered to be part of the problem statement. As such they lie outside the design space and are, therefore, *constraints* on the solution. Many of these are associated with the infrastructure, e.g. size, capability, and the number of individual units.

Physical unit size. Unit size refers to the dimensions and weight of the sensing device. The task and the environment usually have a strong influence on the unit size. The drive for miniaturization can be a byproduct of certain design goals, such as cost, low power consumption, but can be an objective in itself. The most common cited reason is unobtrusiveness useful for stealthy military operations [45, 91], ubiquitous computing [121, 122, 238], or natural environment monitoring [28]. Operation in small-scale environment can also impose this constraint, for example to perform data collection inside the human body [38]. Some design concepts rely on the nature of physical interaction with the environment, for example devices floating on air currents [106, 143].

Small size is a powerful constraint with many consequences [38]. Small unit size leads to limited energy storage, which in turn constrains the available processing power, favors passive sensors, limits actuation options (including locomotion), and, especially in the case of radio links, limits communication range. All these constraints, in turn, have their own consequences, which makes physical unit size a very influential driver in the overall hardware and software design.

System size. In distributed systems, size refers to the number of connected elements. In the DIG problem, platforms are the fundamental *unit of deployment*, i.e. the size of the system is increased by deploying additional platforms. Therefore, in this work, network size will refer to the number of platforms. This metric is easy to measure and is often reported.

Node density. The nature of interactions between the nodes of a SN depends on the average distance between them. Node density can be defined as the number of platforms in the world, or per unit area, or within communication reach [220]. In the case of single-hop broadcast communication, the last metric describes the size of a local "neighborhood" within which all interactions between nodes occur.

Sensor overlap. The overlap in sensor footprints (static or dynamic) is important because it provides an opportunity for information fusion. A quantitative metric can be defined as the ratio of redundant sensor coverage to total sensor coverage. Similarly, feature coverage overlap is the ratio of the number of features observed by several sensors simultaneously to the total number of observed features.

Sensor coverage. This metric refers to the fraction of the environment which can be theoretically observed by the sensors of the team. Typically, part of the environment is considered *covered* if it falls within the observation range of at least one sensing platform. This is related to the concept of collective observability [178] with four categories: Collective Partial Observability, Collective Observability, Individual Observability, and Non-Observability. A quantitative metric can be defined as the fraction of the environment observed by the team. If the environment is largely two-dimensional, the space is characterized by the area. For three-dimensional sampling problems, volumetric measure is more appropriate. Similarly, feature coverage is the ratio of all features observed by the sensors of the team at any given time. The coverage metric is important because when this metric is low there may be a need for platform mobility.

Platform mobility. Physical mobility of platforms is typically measured as platform speed. From the sensing point of view, it can be characterized as the ratio of dynamic space coverage to static sensor footprint. Dynamic properties of both platforms and features is an important aspect of the problem statement. Latencies must be considered in relation to how fast the environment is changing.

Mission duration. The overall longevity of the system is finite, unless resources are renewable. Long missions make the requirements for robustness and reconfiguration more important.

2.2.3 Desired System Qualities

This section lists common SN evaluation criteria [3, 227].

Scalability. In the context of distributed systems, *scalability* refers to the ability of the system to retain its functional capacity despite increase in its size. Several aspects of operation are of particular interest: computational load, communication bandwidth, storage requirements. The main reason for adopting this objective is to allow the system to grow in size by adding more resources, potentially, indefinitely. A quantitative metric of scalability is the theoretical or experimental performance (communication throughput, floating point operations per second) as a function of the number of platforms.

Information quality. The main output of a SN is information about the environment. Information quality is an important factor and can be measured using several metrics: accuracy, coverage, latency.

Accuracy may be quantified by probability of detection, false positives and negatives, target position uncertainty [221], distinct-event delivery ratio [100]. Environment coverage refers to the instantaneous fraction of the environment observed by the team. It is different from the sensor coverage constraint because it refers to the actual coverage provided by the system compared to the theoretical possible coverage allowed by the number and type of sensors. decision making has a strong effect on this metric. Latency refers to the delays in delivery of information due to data processing and transmission, for example average delay [100].

Fault tolerance. In large distributed systems, failure of individual components is accepted as inevitable because of possible physical damage, lack of power, effects of the environment, enemy actions in military scenarios, etc. The objective is to preserve the operation of the SN in the face of individual platform failure. Consequences of this requirement lie in redundancy of system services and resources. The system must also be able to reconfigure itself as a result of component failure. A common quantitative metric is deterioration in information quality with the increase of component failure [86].

Energy efficiency. Energy is required for all aspects of SN operation: sensing, processing, communication, etc. Architecture and algorithms have a direct impact on the energy consumption. Average dissipated energy [100] is an example metric of energy efficiency. For a system with non-renewable resources, mean run time is an energy-related metric.

Production cost. Cost per unit is readily available and generally scales with size. Other factors influencing costs are the number and type of sensors and actuators. At the level of individual units, the cost objective drives the design towards smaller, simpler, less mobile platforms. In many cases, particularly for large systems, it is more appropriate to consider the complete system cost [143].

Build-time qualities. Examples include code reusability and composability, i.e. being able to compose a system from plug-and-play components. Large systems are created by large teams of designers and build-time qualities become increasingly important.

The desired system qualities described here are not independent and are often conflicting. For example, a fundamental trade-off exists between frequency of inter-platform information exchange (latency) and communication bandwidth (scalability). In order to design a practical system, a priority of qualities must be established. This will be done in Section 2.4, after casting the DIG problem into a probabilistic framework.

2.3 Probabilistic Reference Model

This section describes the DIG problem more formally and in probabilistic terms. This description will be used as the *reference model*[6] for the DIG problem. The probabilistic framework is chosen to account for the uncertainty inherent in all interactions of a robotic system with the real world. Sensing, actuation, and dynamic environments, all contribute uncertainty which cannot be fully eliminated or accurately predicted. Probabilistic models provide a powerful and consistent means of describing stochastic properties of systems and interactions. Compared to other approaches, the probabilistic framework provides an *explicit* measure of uncertainty, which leads naturally to the ideas of information fusion and decision making.

The reference model described in this section is based on the COM-municative Multiagent Team Decision Problem (COM-MTDP) [178]. It was introduced as a unified framework and a comparison tool for different team theories. Its scope is very general and much broader than the DIG problem. Three of its features make it particularly attractive in the current context: it is probabilistic, with strong ties to Markov decision processes [177]; distributed, having originated in the multi-agent community; and team-oriented, based on the economic team theory [147]. The minor changes introduced in this work aim primarily to *narrow* the scope and tailor the model to the DIG domain.

2.3.1 Environment

The model partitions the system into three interacting subsystems (as did Figure 2.3): the SN, the environment to be monitored, and the rest. Only the environment and the SN are represented explicitly in the model: the former has to be described and the latter has to be designed. The complete system state s can take on a variety of values all of which are contained in the set $\mathcal{S}$ of possible states. The set of system states can be factored into the states of the environment $\mathcal{X}$ and the states of the sensor network $\mathcal{P}$, so that $\mathcal{S} = \mathcal{X} \times \mathcal{P}$.

[6]Reference model architecture: "an architecture in which the entire collection of entities, relationships, and information units involved in interactions between and within subsystems are defined and modeled" [5].

As a general rule, if some aspect of the system state is important for the operation of the network but is not part of the monitored environment, it is included in the network state.

A specific environment is described by a state vector $x \in \mathcal{X}$. The state of the environment $x^k = x(t_k)$ at time t_k is generally unknown and needs to be estimated. The environment is assumed to consist of a finite set of *features*: trees, walls, roads, people, vehicles, with the state space factored accordingly $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$. Typical feature states include position, size, color. A feature may have internal inputs $u_i^k \in \mathcal{U}_i$. The feature state is assumed to have the Markov property: the current state x_i^k depends only on the previous state x_i^{k-1} and the input u_i^{k-1}. The *feature state transition* function describes the state evolution from one time step to the next: $G : \mathcal{X}_i \times \mathcal{U}_i \times \mathcal{X}_i \to \Re$.[7] If the feature state x_i^{k-1} and input u_i^{k-1} are known, the state transition function describes the probability of the feature ending up in state x_i^k

$$G(x_i^{k-1}, u_i^{k-1}, x_i^k) \rightharpoonup P(x_i^k \mid x_i^{k-1} = x, u_i^{k-1} = u). \tag{2.1}$$

This formulation makes an important assumption that features do not influence each other, i.e. states of any two features are decoupled $x_i^k \neq f(x_j^k)$. In situations where this assumption is not justified and $x_i^k = f(x_j^k)$, a super-feature may be defined with the state vector combining the original two $x_s^k = \{x_i^k, x_j^k\}$.

Without loss of generality, internal inputs will not be considered in feature models. A feature, especially mobile, may well have internal inputs but, typically, they are unknown to the observer. If the inputs are important for solving the problem, they must be *estimated.* In this case they become part of the feature state and the state transition function $G : \mathcal{X}_i \times \mathcal{X}_i \to \Re$ is simply

$$G(x_i^{k-1}, x_i^k) \rightharpoonup P(x_i^k \mid x_i^{k-1} = x). \tag{2.2}$$

2.3.2 Sensor Network

The SN consists of a number of platforms. A specific SN α is described by a state $p \in \mathcal{P}_\alpha$ which is factored into states of individual platforms $\mathcal{P}_\alpha = \mathcal{P}_1 \times \ldots \times \mathcal{P}_n$. Typical platform states are pose, velocity, fuel level. Subsystems of a typical sensing platform are shown in Figure 2.5. All platforms include a power source, a computer, and means of communication.

[7]For a discrete representation of a probability density function, an interval $[0, 1] \in \Re$ is sufficient.

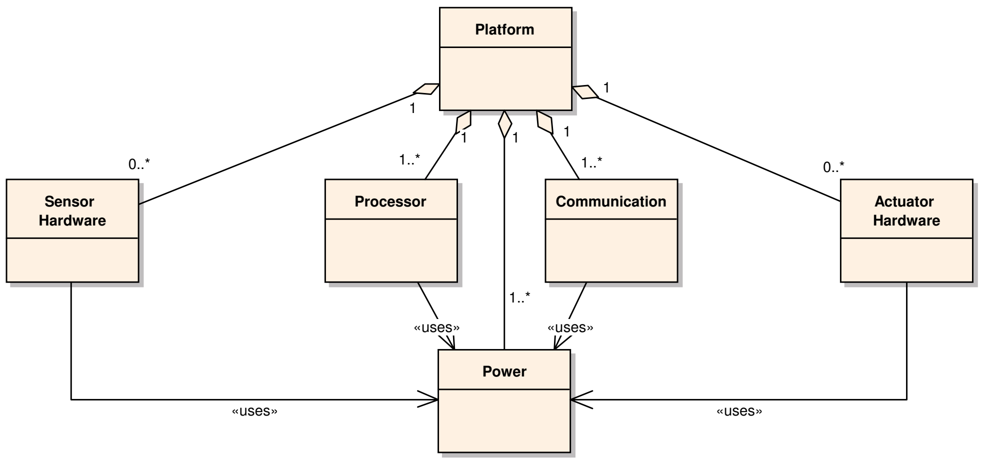


Figure 2.5: Subsystems of a typical sensing platform. Each platform is equipped with a power source, processor, and means of communication. Sensing and actuation equipment is optional. All subsystems use power.

Optionally, a platform may also be equipped with sensors and actuators.[8] Most platform properties strongly depend on the physical size and available power [151].

Platforms are an important concept in SN for two reasons. First, the platform is a unit of deployment and, therefore, the number of platforms is the characteristic size of SN. Second, the platform is the unit of decision making.

Actuation. If platform i has an actuator, then it can perform an action a_i^k from a set of all possible actions $\{\mathcal{A}_i\}_{i\in\alpha}$. A history of actions up to time t_k may be stored in a sequence $A_i^k = \{a_i^1, a_i^2 \dots a_i^k\}$. A combined set $\mathcal{A}_\alpha = \prod_{i\in\alpha} \mathcal{A}_i$ defines possible actions of the entire team. The most common actuators are means of locomotion. Other actuators include means of pointing sensors such as pan-tilt-zoom cameras. A mode switch (even a simple on-off) for a sensor can be considered an action.

In general, a platform's action may have an effect on the states of environment features and the states of other platforms. The fully-coupled state transition function is in the form $F_\alpha : \mathcal{S} \times \mathcal{A}_\alpha \times \mathcal{S} \rightarrow \Re$. For the DIG problem, certain simplifying (and limiting) assumptions can be made. In the discussion of feature models, it was assumed that a platform's actions do not influence the feature states. This assumption precludes direct

[8]`UML` The possible number of platform subsystems in indicated in Figure 2.5. In UML, the number of entities in a set is indicated by *multiplicity*. For example, each Platform has one or more Power units, shown as [1..*], but each Power unit must belong to strictly one Platform, shown as [1]. The multiplicity of Physical Sensors includes zero, which makes it optional equipment.

manipulation of the environment (e.g. no “couch pushing”) and indirect strategic influence (e.g. no pursuit-evasion games). The joint team state transition function describes the probability of transition from one team state to another after a certain team action was performed $F : \mathcal{P}_\alpha \times \mathcal{A}_\alpha \times \mathcal{P}_\alpha \rightarrow \Re$. In other words, it defines the probability of transition from team state p_α^{k-1} at time t_{k-1} to state p_α^k at time t_k after performing a team action a_α^{k-1}

$$F_\alpha(p_\alpha^{k-1}, a_\alpha^{k-1}, p_\alpha^k) \rightharpoonup P(p_\alpha^k \mid p_\alpha^{k-1} = p, a_\alpha^{k-1} = a). \tag{2.3}$$

This model allows coupling between actions of individual platforms. For example, two platforms may be precluded from performing actions which place them in the same physical location. In many cases the team state transition function can be decoupled into the functions describing the states and actions of individual platforms

$$F_i(p_i^{k-1}, a_i^{k-1}, p_i^k) \rightharpoonup P(p_i^k \mid p_i^{k-1} = p, a_i^{k-1} = a). \tag{2.4}$$

Platform models are used by platforms themselves for planning future actions. The platforms themselves may be very different, and so the models used for describing them are different too, $F_i \neq F_j$. Feature models are used by platforms for predicting state changes of the environment in the absence of observations. Note, that the models of the same feature used by two different platforms do not need to be the same, $G_i \neq G_j$. (G_i refers to the model of a feature used by platform i.)

Sensing. If platform i has a sensor, then it can make an observation z_i^k of the environment drawn from a set of all possible observations $\{\mathcal{Z}_i\}_{i\in\alpha}$. A history of observations up to time t_k may be stored as a sequence $Z_i^k = \{z_i^1, z_i^2 \dots z_i^k\}$. A combined set $\mathcal{Z}_\alpha = \prod_{i\in\alpha} \mathcal{Z}_i$ defines observations of the entire team. Any physical sensor can be used by networked sensing platforms. The typical ones include acoustic microphones, video and IR cameras, lasers, and thermocouples. Sensor hardware is listed as optional equipment for reasons of generality. Although unlikely to have no sensors at all, a platform may choose not to contribute information from locally obtained observations to the network. It may still participate in team activities by performing computations, actions, or message routing.

Observations are the only source of original information about the environment. The relationship between the state of the world and observations is captured in the *sensor model*. The joint team observation model probabilistically links the world state, team actions and observations, $H_\alpha : \mathcal{S} \times \mathcal{A}_\alpha \times \mathcal{Z}_\alpha \rightarrow \Re$. A sensor model can be used in two ways: as an *observation model* to describe the probability of possible observations given a particular world state and as a *likelihood function* to describe the likelihood of possible world states given that a particular observation has been made.

The observation model defines the probability of making a particular team observation z_α^k following a team action a_α^{k-1} when the system is in a state s^k

$$H_\alpha(s^k, a_\alpha^{k-1}, z_\alpha^k) \rightharpoonup P(z_i^k \mid s^k = s, a_\alpha^{k-1} = a). \tag{2.5}$$

Of particular interest is the case when the joint distribution can be separated into individual platform observation functions, $H_i : \mathcal{X} \times \mathcal{P}_i \times \mathcal{A}_i \times \mathcal{Z}_i \rightarrow \Re$, which depend on the state of the environment and the state and actions of the platform performing the observation

$$H_i(x^k, p_i^k, a_i^{k-1}, z_i^k) \rightharpoonup P(z_i^k \mid x^k = x, p_i^k = p, a_i^{k-1} = a). \tag{2.6}$$

The observation model can also be used as a likelihood function after a particular observation is made. As such, it provides information on how *likely* the actual observation was as a function of the true world state

$$H_i(x^k, p_i^k, a_i^{k-1}, z_i^k) \rightharpoonup L(z_i^k = z \mid x^k, p_i^k = p, a_i^{k-1} = a). \tag{2.7}$$

The *observation likelihood* $L(z_i^k = z \mid x^k)$ is equivalent to the actual observation $z_i^k = z$ according to the *Likelihood Principle* [16]: in making inferences or decisions about x^k after $z_i^k = z$ is observed, the observation likelihood $L(z_i^k = z \mid x^k)$ contains all relevant experimental information. Critically, the observation likelihood is a function defined on the world state space $\mathcal{X}$.

The sensor models in Equations 2.5–2.7 do not make any assumptions on the completeness of environment coverage by individual platforms as well as the team as a whole. Pynadath and Tambe refer to this as *collective partial observability* [178]. For an information gathering system this is important, because information fusion may not be a high priority for a system with complete individual observability of the environment.

2.3.3 Communication

Communication allows platforms to exchange information. Set $\{\mathcal{M}_i\}_{i\in\alpha}$ includes all possible messages that a platform i can send, implicitly defining a set of combined messages $\mathcal{M}_\alpha = \prod_{i\in\alpha}\mathcal{M}_i$. The messages are used by platforms to exchange information and influence beliefs of other team members. Messages have a sender and a receiver, so for example, platform i sends a message m_{ij}^k to platform j at time t_k. A history of communication from platform i to platform j up to time t_k may be stored in a sequence M_{ij}^k. All messages received by platform i from the team may be stored in $M_{\alpha i}^k$.

No restriction is made on the type of messages: single-cast m_{ij}^k, multi-cast $m_{i\beta}^k$, or broadcast m_{i*}^k. Note, that the model of message transmission is deterministic, a message is always assumed to be delivered to its destination. A possible extension of the model could probabilistically represent reliability of the communication channel.

2.3.4 Belief

The true state of the world s is unknown to the platforms. All information available about the world is compiled into the platforms' *beliefs*. Each platform i forms a belief state $b_i^k \in \mathcal{B}_i$, based on the information obtained through time t_k, where $\mathcal{B}_i$ describes the set of possible belief states. The set of possible combined belief states over all platforms is $\mathcal{B}_\alpha$.

A communicating platform has three sources of information which influence its belief: prior information, local observations and messages from the rest of the team. The *belief* function maps prior beliefs, a history of observations, and a history of received messages to the current belief

$$b_i^k = D_i^0(b_i^0, Z_i^k, M_{\alpha i}^k). \tag{2.8}$$

A recursive *belief update* function can be used if the state has the Markov property, $D_i : \mathcal{B}_i \times \mathcal{Z}_i \times \mathcal{M}_\alpha \rightarrow \mathcal{B}_i$. The updated belief of platform i is based on the current belief and the observations and messages since the last update

$$b_i^{k+} = D_i(b_i^k, z_i^k, m_{\alpha i}^k). \tag{2.9}$$

In planning future actions, beliefs on future states of the world may be required. These are formed based on the present belief and the state transition functions in Equations 2.2

and 2.4. The belief at time t_{k+n} based on the information available at time t_k is written as $b_i^{k+n,k}$.

For concreteness, the total belief of the team can be partitioned into beliefs about the state of the environment, the team states, observations, actions, messages, and possibly other: $b_\alpha^k \in \mathcal{B}_\alpha = \mathcal{B}_{x\alpha} \times \mathcal{B}_{p\alpha} \times \mathcal{B}_{z\alpha} \times \mathcal{B}_{a\alpha} \times \mathcal{B}_{m\alpha} \times \dots$. The corresponding beliefs for a particular platform are designated as $\mathcal{B}_i = \mathcal{B}_{xi} \times \mathcal{B}_{pi} \times \mathcal{B}_{zi} \times \mathcal{B}_{ai} \times \mathcal{B}_{mi} \times \dots$. The list is intentionally open-ended to allow other types of beliefs. Each of the listed belief types is described in turn below.

The platform's belief about the environment describes the probability of each state of the world according to the currently available information $b_{xi} : \mathcal{X} \rightarrow \Re$. The world state transition function $G(x_i^{k-1}, x_i^k)$ in Equation 2.2 may be combined with the definition of the world state belief b_{xi}^k into a world *belief prediction* function $G' : \mathcal{B}_{xi} \rightarrow \mathcal{B}_{xi}$. It *predicts* the state of the world at the next time step

$$b_{xi}^{k+1,k} = G'(b_{xi}^k). \tag{2.10}$$

The platform's current environment belief is based on the prior belief, all local observations, and all messages received from the team members

$$b_{xi}^k = P(x^k \mid x^0, Z_i^k, M_{\alpha i}^k). \tag{2.11}$$

If all information is shared within the team through communication, the team belief about the environment is

$$b_{x\alpha}^k = P(x^k \mid x^0, Z_\alpha^k). \tag{2.12}$$

A platform's belief about the states of all platforms in the team (including itself) is as $b_{pi} : \mathcal{P}_\alpha \rightarrow \Re$. It is based on the prior belief, all local observations and actions, and the messages received from the team members

$$b_{pi}^k = P(p_\alpha^k \mid p_\alpha^0, Z_i^k, A_i^k, M_{\alpha i}^k). \tag{2.13}$$

Similar to the case of the world belief, the platform *belief prediction* function may be defined as $F_i' : \mathcal{B}_{pi} \times \mathcal{A}_i \rightarrow \mathcal{B}_{pi}$. It predicts the state of the platform based on the previous belief and the known platform action

$$b_{pi}^{k+1,k} = F_i'(b_{pi}^k, a_i^k). \tag{2.14}$$

Beliefs about the combined team observations, actions and messages may also be important for decision making of an individual platform and are defined similarly

$$\begin{aligned} b_{zi}^k &= P(z_\alpha^k \mid z_\alpha^0, Z_i^k, A_i^k, M_{\alpha i}^k) && (2.15)\\ b_{ai}^k &= P(a_\alpha^k \mid a_\alpha^0, Z_i^k, A_i^k, M_{\alpha i}^k) && (2.16)\\ b_{mi}^k &= P(m_\alpha^k \mid m_\alpha^0, Z_i^k, M_{i\alpha}^k, M_{\alpha i}^k). && (2.17)\end{aligned}$$

They represent probability distributions on the sets of possible team observations $b_{zi} : \mathcal{Z}_\alpha \rightarrow \Re$, actions $b_{ai} : \mathcal{A}_\alpha \rightarrow \Re$, and messages $b_{mi} : \mathcal{M}_\alpha \rightarrow \Re$.

2.3.5 Policies

The set of possible *action policies* is defined as mappings from local belief states to actions $\pi_{iA} : \mathcal{B}_i \rightarrow \mathcal{A}_i$. It chooses an action based on the current belief[9]

$$a_i^k = \pi_{iA}(b_i^k). \tag{2.18}$$

The action policy may also be used to make decisions about actions in the future. An action $a_i^{k+n,k} \in \mathcal{A}_i$ is chosen at time t_k to be executed at time t_{k+n}. An *action plan* is a sequence of planned actions chosen at the same time $A_i^{k+n,k} = \{a_i^{k+1,k}, a_i^{k+2,k} \ldots a_i^{k+n,k}\}$. Future actions are chosen based on predicted future beliefs

$$a_i^{k+n,k} = \pi_{iA}(b_i^{k+n,k}). \tag{2.19}$$

[9]In the field of robotics there are two well established approaches to solving the action problem: *reactive* and *deliberative*. An identical division exists in the multi-agent community where agents are sometimes classified into *tropistic* and *hysteretic* [54]. Despite the different names, the issue is whether or not to maintain the "state", or more precisely, an estimate of the state of the world. Reactive systems avoid having a world model and base their actions solely on the most recent observation, $\pi_{iA} = \mathcal{Z}_i \rightarrow \mathcal{A}_i$. This feature can even be used as a test for "reactiveness": reactive systems "always react to the same sensor state with the same motor action" [166]. A large body of work is based on reactive action selection starting with pioneering work of Brooks [25], including many applications to multi-robot systems [65, 170, 239]. The reactive approaches may be fitted into the reference model by noting that an observation can always be considered a belief about the world when combined with an uninformative prior.

Planned actions may lead to future observations $z_i^{k+n,k} \in \mathcal{Z}_i$. An *observation plan* is a sequence of observations planned at time t_k to occur in the future: $Z_i^{k+n,k} = \{z_i^{k+1,k}, z_i^{k+2,k}, \ldots, z_i^{k+n,k}\}$.

Similar to the action policy, a separate policy for communication is defined[10], which determines what information is exchanged between team members and when, $\pi_{iM} : \mathcal{B}_i \rightarrow \mathcal{M}_i$. The current message is based on the current belief

$$m_{i*}^k = \pi_{iA}(b_i^k). \tag{2.20}$$

Since different platforms may have different actions and messages available to them, their policies may also be different $\pi_{iA} \neq \pi_{jA}$, $\pi_{iM} \neq \pi_{jM}$. The joint team policies $\pi_{\alpha A}$ and $\pi_{\alpha M}$ are the combined policies across all platforms in the team.

2.3.6 Utility

A *utility function* provides a means of evaluating different actions, allowing for direct comparison of alternative policies. A utility function depends on the true state of the system, both the environment and the platforms. It also reflects the cost of action and communication. For a team of sensing platforms, the combined team utility is defined as a mapping from system states, actions, and messages to a scalar: $U : \mathcal{X} \times \mathcal{P}_\alpha \times \mathcal{A}_\alpha \times \mathcal{M}_\alpha \rightarrow \Re$.

Note that just like in the original economic team theory, this definition of a team assumes that team members have a common goal (and utility) and that they work selflessly towards that goal. A more general notion of an *organization* removes this constraint at the expense of more complexity [147]. The notion of a team is adequate for the system under consideration because it has essentially a single task of gathering information. The true team utility at any time step t_k can be calculated as follows

$$\rho^k = U(x^k, p_\alpha^k, a_\alpha^k, m_\alpha^k), \tag{2.21}$$

with the interpretation that ρ^k is the gain obtained at time t_k if taking the team action a_α^k exchanging team communication messages m_α^k and when the true state of the system is (x^k, p_α^k). For a fixed world state, utility function U induces a preferential ordering on $\mathcal{A}_\alpha$

[10]Somewhat contrary to the arguments of Speech Acts theory [200].

and $\mathcal{M}_\alpha$, leading to the choice of appropriate a_α^k and m_α^k.

Further assumptions on the structure of the utility function can be made. Pynadath and Tambe suggest that the costs of communication and actions are independent of each other [178]. The utility function can then be decomposed into two additive components: a domain-level utility U_A and a communication-level utility U_M

$$\rho^k = U_A(x^k, p_\alpha^k, a_\alpha^k) + U_M(x^k, p_\alpha^k, m_\alpha^k). \tag{2.22}$$

2.3.7 Expected Utility

A utility function in the form of Equation 2.21 is a reward obtained when all states, actions, and messages are known. In practice, not all information may be available when the utility is to be evaluated: the true state x^k may not be known with certainty, the decisions must be made about events in the future. For these reasons, the perceived or predicted utility must be used.

One natural method of predicting utility is using the *expected utility*, which takes into account the uncertainty about the world state encapsulated in the belief b_α^k [16]

$$\rho_\alpha^k = E^{b_\alpha^k}\left\{U(x^k, p_\alpha^k, a_\alpha^k, m_\alpha^k)\right\}. \tag{2.23}$$

Probabilistic belief definitions in Equations 2.11–2.17 allow the use of the concept of expectation, which is evaluated over the random variables $(x^k, p_\alpha^k, a_\alpha^k, m_\alpha^k)$

$$\rho_\alpha^k = \int U(x^k, p_\alpha^k, a_\alpha^k, m_\alpha^k) P(x^k) P(p_\alpha^k) P(a_\alpha^k) P(m_\alpha^k)\, dx\, dp_\alpha\, da_\alpha\, dm_\alpha. \tag{2.24}$$

This definition of the expected utility is based on the complete information available to the team and can be calculated for any time step: past, present, or future. As a shortcut notation, the expected utility will referred to as a function of beliefs

$$\rho_\alpha^k = J(b_{x\alpha}^k, b_{p\alpha}^k, b_{a\alpha}^k, b_{m\alpha}^k). \tag{2.25}$$

2.3.8 Expected Value

The *expected value* of the policies $\pi_{\alpha A}$ and $\pi_{\alpha M}$ is the expected total team utility when these policies are executed. Over a period of time to a finite planning horizon t_K the expected value is defined as a sum[11] of expected utilities

$$V^K(\pi_{\alpha A}, \pi_{\alpha M}) = E^{b^k_\alpha} \left\{ \sum_{t=0}^{K} U^{k+t} \mid \pi_{\alpha A}, \pi_{\alpha M} \right\} = \sum_{t=0}^{K} \rho_\alpha^{k+t,k}, \tag{2.26}$$

where the expected utility is calculated according to Equation 2.24.

Planning horizon $t_K = 0, 1 \ldots N$ determines how far into the future expected value is evaluated. A 1-step planning horizon means that the current action is chosen by considering only the expected utility at the next time step.

2.3.9 Problem Statement

The DIG problem can now be stated in the form of four tuples[12]

$$\langle \mathcal{X}, G_\alpha \rangle, \langle \mathcal{P}_\alpha, \mathcal{A}_\alpha, F_\alpha, \mathcal{M}_\alpha, \mathcal{Z}_\alpha, H_\alpha, \mathcal{B}_\alpha, D_\alpha \rangle, \langle U, t_K \rangle, \langle \pi_{\alpha A}, \pi_{\alpha M} \rangle, \tag{2.27}$$

representing the environment, the platforms, utility function, and the team policies. The objective of the team as a whole and each of its members individually is the following: given the problem described by the first three tuples, choose the team action and communication policies $\langle \pi^*_{iA}, \pi^*_{iM} \rangle$ which maximize the expected team value

$$\langle \pi^*_{\alpha A}, \pi^*_{\alpha M} \rangle = \max V^K(\pi_{\alpha A}, \pi_{\alpha M}). \tag{2.28}$$

[11]In the field of *Reinforcement Learning* the value is commonly calculated as a *discounted* sum of expected utilities, $V^K = \sum_{t=0}^{K} \gamma^t \rho_\alpha^{k+t}$, where $\gamma(t)$ is the *discount rate*, $0 \leq \gamma \leq 1$. The discount rate is introduced for three reasons. It allows infinite time horizon, because the infinite sum in this case has a finite value as long as the individual utilities are bounded. It reflects the increasing uncertainty of future outcomes of actions. It can also be used as a design parameter to focus on short-term or long-term utility of action. When γ is close to one, the expected value takes long-term utility into account. When $\gamma = 0$, the expected value is equal to the immediate expected utility. In this work, discount rates are not used because only a finite horizon is considered and probabilistic models account for the uncertainty in the system.

[12]Compared to the original COM-MTDP [178], this formulation is specialized to the information gathering domain. The states of the world are separated into active platforms and passive features. The actions of platforms are assumed not to affect the states of the features. The beliefs are defined probabilistically. The belief update term D is brought into the problem statement. The notation is changed to bring it more in line with the conventions of the probabilistic filtering problem.

This formulation does not emphasize the fact that the belief update, decision making and policy selection is performed at the platform level. To make this more explicit, the model is reformulated from the point of view of an individual platform.

We start with the expected team utility, which is needed to evaluate alternative actions. If the platform has access only to its own belief about the world, then the expected team utility must be approximated by *partial utility* $\tilde{J}$ [70]

$$\tilde{\rho}_i^k = E^{b_i^k}\left\{U(x^k, p_\alpha^k, a_\alpha^k, m_\alpha^k)\right\} = \tilde{J}(b_{xi}^k, b_{pi}^k, b_i^a, b_i^m), \tag{2.29}$$

where the agent's partial utility is calculated based on the *local* beliefs on the state of system: the environment b_{xi}^k, all platforms b_{pi}^k, team actions b_{ai}^k, and team messages b_{mi}^k. Unlike in the case of the team utility, there is no restriction on the differences among the expected partial utilities, $\tilde{\rho}_i^k \neq \tilde{\rho}_j^k$. The partial utilities could be additive or not depending on the interaction between the decision makers [147]. In general, the team utility cannot be found as a sum of partial utilities of individual platforms

$$\rho_\alpha^k \neq \sum_{i\in\alpha} \tilde{\rho}_i^k. \tag{2.30}$$

For example, the plans of two platforms developed in isolation may interfere and the combined utility will be lower than the sum of the individual utilities. Likewise, the interaction may be beneficial and the combined utility will be higher than the predicted sum. The problem of *utility fusion* is a fundamental problem in cooperative action planning: how to obtain an estimate of team utility based on the information available locally at each platform.

The expected partial value is calculated based on the individual platform's policies

$$\tilde{V}_i^K(\pi_{iA}, \pi_{iM}) = E^{b_i^k}\left\{\sum_{t=0}^{K} U^{k+t} \mid \pi_{iA}, \pi_{iM}\right\} = \sum_{t=0}^{K} \tilde{\rho}_i^{k+t,k}, \tag{2.31}$$

where $\tilde{\rho}_i^{k+t,k}$ is the local estimate of the team's expected utility at time t_{k+t}.

The DIG problem can now be restated from the point of view of a team member. Only the terms that a team member has control over or has first-hand knowledge of are retained

$$\langle \mathcal{X}, G_i \rangle, \langle \mathcal{P}_i, \mathcal{A}_i, F_i, \mathcal{M}_{\alpha i}, \mathcal{M}_{i\alpha}, \mathcal{Z}_i, H_i, \mathcal{B}_i, D_i \rangle, \langle U, t_K \rangle, \langle \pi_{iA}, \pi_{iM} \rangle . \tag{2.32}$$

The objective for each platform is the following: based on the information available to the agent and shown in the first three tuples, choose the local action and communication policies $\langle \pi^*_{iA}, \pi^*_{iM} \rangle$ which maximize the expected team value

$$\langle \pi^*_{iA}, \pi^*_{iM} \rangle = \max \tilde{V}_i^K(\pi_{iA}, \pi_{iM}). \tag{2.33}$$

Because the agent's expected value is based on the local belief b_i^k, the objective is to make sure that b_i^k is as close as possible to b_α^k, in which case $\tilde{\rho}_i^k$ will be close to ρ_α^k and V^K close to $\tilde{V}_i^K$.

2.4 Related Work

The reference model presented in Section 2.3 established the structure of the *problem space* of the DIG problem. The literature survey in this section focuses on the *solution space.* A brief quantitative analysis demonstrates that a) there are clusters in both the problem and solution spaces and b) there is a clear correspondence between the clusters. A group of similar solutions which solve similar problems can be described as a *SN type.*

Three SN types are identified in Section 2.4.1 and the following three sections focus on each of the types. Micro Sensor Networks are discussed in Section 2.4.2, Multi-Robot Systems in Section 2.4.3, and Macro Sensor Networks in Section 2.4.4. Some of the examples from the literature cannot be easily categorized, while others are designed to cross the category boundaries. The latter are termed Hybrid Sensor Networks and discussed separately in Section 2.4.5.

The formulation of Equation 2.32 is intentionally very general. Many choices have to be made when designing a system described by the reference model. The following list of key choices will be used as a guide in describing related work.

1. System architecture

2. Belief update algorithm D
3. Utility function definition U
4. Policy selection algorithm
5. Implementation

Because the DIG problem is very broad in scope, the amount of related work even in the last few years is immense. A comprehensive review therefore is hardly possible. In particular, the review of the fields of Micro Sensor Networks and Multi-Robot Systems should be viewed as mere starting points. The main outcome of this section is in clear statement of the goals for this thesis. This is done by aligning this work with Macro Sensor Networks and adopting its objectives and priorities.

2.4.1 Sensor Network Types

Taxonomic studies of SN's are quite common [50, 53, 71, 220, 227]. This presentation is different in that it tries to obtain quantifiable metrics and ties them to a probabilistic model covering the entire problem space. Table 2.1 shows a representative list of projects and approaches which address either the entire DIG problem or some of its parts. The columns of the table list three main aspects of any solution approach: architecture, algorithms, and implementation. The entries are checked for certain solution properties: whether the architecture is distributed; whether information fusion, cooperative decision making, and self-configuration is present; and whether the system implementation is component-based. Because, the approach advocated in this thesis employs all of the above, this table is essentially a distance metric between this thesis and the related work.

The entries are listed in order of broadening scope and increasing overlap with the approach advocated in this thesis. The bottom ("high-overlap") part of the table is dominated by recent work. This fact reflects relative maturity of the algorithms and commercial availability of capable hardware, wireless networks and proper software tools. It seems likely that the trend will continue towards large-scale multi-faceted distributed applications.

The approach adopted to classify the related work is as follows. From over forty entries in Table 2.1, seventeen were chosen which have enough reported details concerning experiments

Project /	Architecture		Algorithms		Implement.
Approach	*distributed*	*info fusion*	*coop. dec. making*	*self-config.*	*component-based*
Architecture styles [8]	✓				
Nodestar [218, 219]		✓			
Freiburg RoboCup [43]		✓			
FireMapper [87]		✓			
System config [112]				✓	
Orocos framework [169]					✓
Gala meta OS [189]					✓
Oxford Strips [184]	✓	✓			
Oxford process plant [68]	✓	✓			
29 Palms CSP [45, 151]	✓	✓			
CMU patrol [42, 196, 216]	✓		✓		
USC deploy [93]	✓		✓		
Berkeley [212]	✓		✓		
μ**SN** [214, 220]	✓			✓	
Directed diffusion [99]	✓			✓	
SINA [204]	✓			✓	
RAP [133]	✓			✓	
SP, U. of MD [97]	✓			✓	
UMass tracking [90]	✓			✓	
CMU millibots [162]		✓	✓		
CMU explore [211]		✓	✓		
Bayes programming [120]		✓	✓		
Multi-sensor OG [236]		✓			✓
UMN Scouts [194]			✓		✓
GEMS study [143]	✓	✓		✓	
Sensor fusion [102]	✓	✓			✓
Multi-layer fusion [6]	✓	✓			✓
Generic fusion arch [18, 27]	✓	✓			✓
ANSER project [221]	✓	✓			✓
Hive project [155]	✓			✓	✓
LIME [158]	✓			✓	✓
μSN [28, 58, 201]	✓			✓	✓
Logical sensor sys. [78, 79]		✓	✓		✓
Xerox μ**SN** [247, 248]	✓	✓	✓	✓	
Info control [70, 71]	✓	✓	✓		✓
Info search [21]	✓	✓	✓		✓
MIRO RoboCup [235]	✓	✓	✓		✓
Em* framework [64]	✓	✓		✓	✓
TCN system [36]	✓	✓	✓	✓	✓
Centibots [94, 113, 114]	✓	✓	✓	✓	✓
MiLAN middleware [77]	✓	✓	✓	✓	✓
ASN	✓	✓	✓	✓	✓

Table 2.1: Summary of related work. The table shows a degree of overlap between ASN and selected examples from the literature. Horizontal lines divide entries into groups based on the number of check marks. Projects with detailed information on experiments or simulations are highlighted. They are used for quantitative characterization of the field.

or simulation. These selected entries are highlighted in the table. System properties are described in quantitative terms using the metrics defined in Section 2.2. The raw "data" and the details of calculating the derived metrics are presented in Appendix A. The solutions are also classified, however, because of the wide variety of possible approaches, this part of the analysis is necessarily more qualitative.

Experimental immaturity of the SN field does not allow a clustering analysis which relies on the data alone. Rather, the clusters are named based partly on the data in Appendix A and partly on the engineering insight. Three types of SN's were thus identified:

1. Micro Sensor Networks (μSN),
2. Macro Sensor Networks (MSN), and
3. Multi-Robot Systems (MRS).

SN Constraints and Qualities	μSN	MSN	MRS
Small unit size	✓		
Large team size	✓	✓	
Heterogenous team		✓	✓
Active platforms		✓	✓
Information gathering main task	✓	✓	
Long mission duration	✓	✓	
Information used within sensor field		✓	✓
High information accuracy		✓	✓
Low information latency			✓
Relatively large development effort		✓	

Table 2.2: Importance of performance metrics by sensor network type. Constraints related to the infrastructure are listed on top. Run-time qualities are listed in the middle. Build-time properties are listed at the bottom. The highlighted entries can be easily compared quantitatively.

The distinguishing features of the three SN categories are listed in Table 2.2. The top part of the table lists constraints related to the infrastructure. The middle part deals with mission description and the priority assigned to different objectives. The bottom part of the table lists build-time properties of the system.

Some of the categories are correlated, for example the objective of *large scale coverage* usually leads to *large team size* or the use of *active platforms*, or both. Without getting into the specifics of the individual design process, it is useful to list all features which are easy to quantify and which help differentiate network types. Classification of the sample projects into SN types is shown in Table A.5 and is used in Figures 2.6–2.8.

Differences in functional requirements are considered first. Information gathering is the main objective of systems in the μSN and MSN categories. MRS are engaged in network-wide information gathering inasmuch as it is beneficial for performing domain-specific tasks. For example, in the RoboCup domain, robots on the same team are often programmed to exchange information about the position of the ball. It is a task of information gathering but the main objective of the system is not to track the ball but to score a goal.

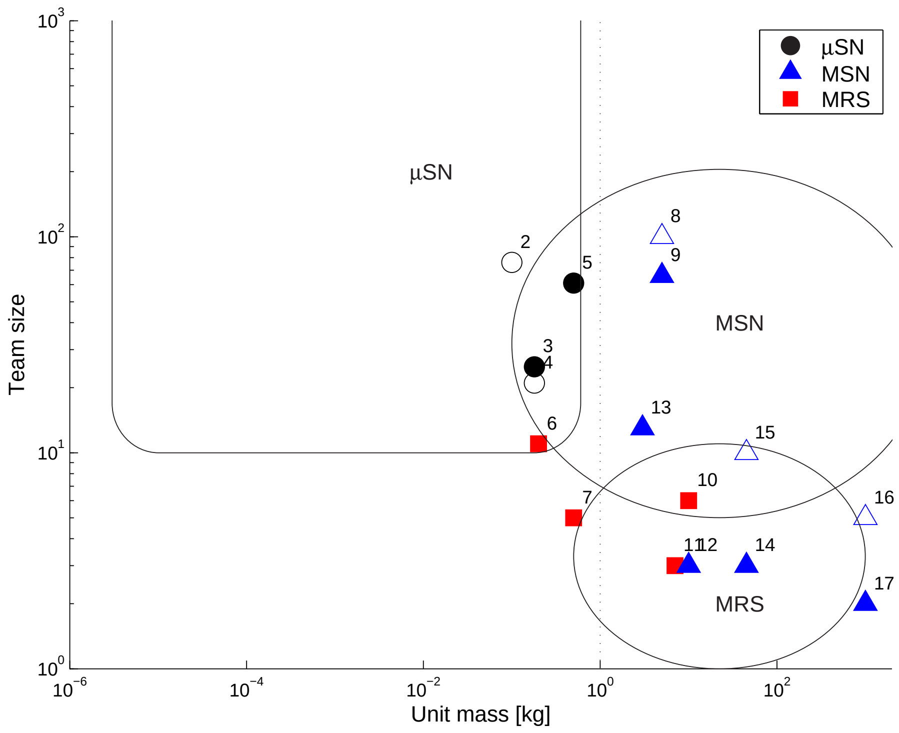


Figure 2.6: Sensor network examples: unit mass vs. team size. The GEMS study is off the scale (2.5 mg, 10^{10} units). Both experimental (filled markers) and simulation (outline markers) are shown. The numbers next to the markers correspond to the case numbers in Appendix A.

Infrastructure constraints are considered next. Individual unit mass and team size are most often cited in the literature and are plotted in Figure 2.6. Approximate *design* boundaries for the SN types are shown. μSN is the only type explicitly constrained by the micro-scale requirement. Its target region extends towards the region of large teams of small units. The intended team sizes for μSN systems are often quoted in the thousands [50, 99], tens of thousands [1], and even billions [143]. The state of the art in actual team sizes across all SN types is reaching the level of one hundred platforms. This order of magnitude is the intended design point of most MSN applications. Practical MRS designs have been mostly for indoor applications with typical mass of 5–10 kg and teams size of 3–6.

Physical size is important because it has very strong influence on energy storage. μSN platforms are battery powered and the design of all subsystems is extremely energy-conscious. The main energy consumer in stationary micro platforms is the communication subsystem because the energy required to communicate using radio waves is not reduced with miniaturization of the device [179]. The relative energy budget is different, however, for mobile platforms. The power necessary for locomotion of any kind (with a possible exception of dust-sized motes floating in the air [106, 143]) typically exceeds the power needed for communication. For example, Robotmote, one of the smallest commercially available mobile robots, spends 72% of its power on locomotion [207]. Even if locomotion is not involved, significant amount of energy may be required if active sensors are used, such as sonar, laser range finders, and radars. Narrow field-of-view sensors which require repositioning can also be large consumers of power, for example cameras with pan-tilt mounts .

These considerations make platform mobility an important factor. Figure 2.7 shows two related infrastructure metrics: sensor area coverage and sensor mobility. μSN's are predominantly static and require sensor and communication overlap in order to guarantee reasonable performance. In the figure, the μSN systems appear on the zero mobility line and the sensor coverage metric is greater or equal to one. MSN and MRS have a wide range of options in this coordinate frame.

Mission duration is shown in Figure 2.8. Here again, the hardware state of the art has not caught up with design targets. Lifespan of future μSN systems is cited from weeks [143] to years [179] with today's commonly used devices just breaking the one-week barrier if any useful measurements are to be performed. MRS tend to be short-mission oriented compared to MSN. Today's robotic missions are typically supervised by humans and last on the order of minutes and hours. Higher levels of autonomy and self-configuration are needed for missions of longer duration.

Functional requirements and constraints in Table 2.2 lead to technical drivers which di-

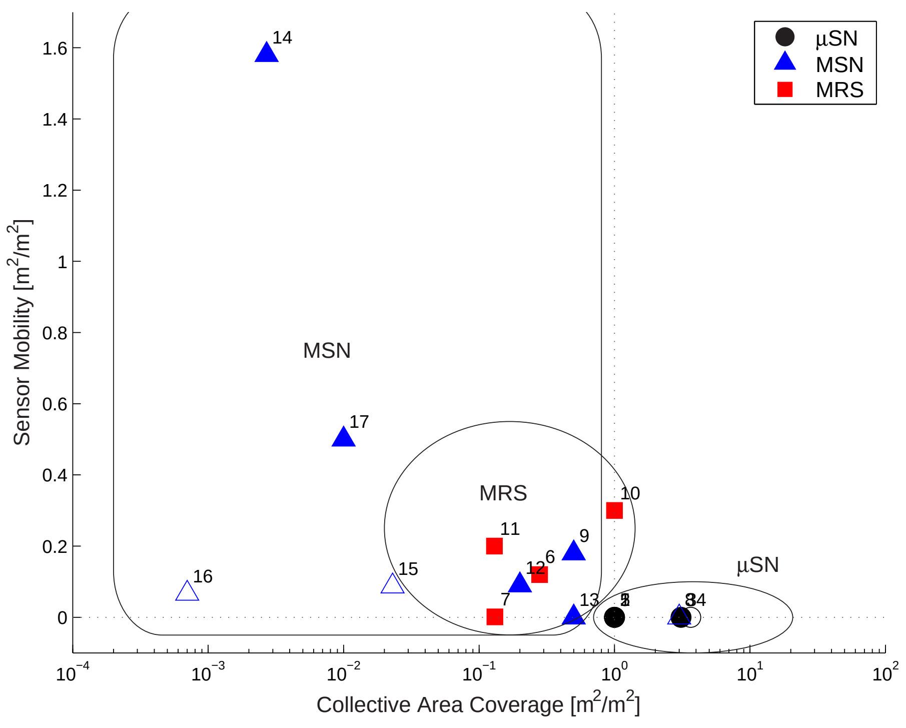


Figure 2.7: Sensor network examples: sensor coverage vs. sensor mobility. Both experimental (filled markers) and simulation (outline markers) are shown.

rectly shape solution methods. The relationship between the requirements and the drivers is summarized in Figure 2.9. Energy efficiency is an influential driver prompted by the requirement of small unit size and long mission duration. This driver has far-reaching consequences because energy is needed for all aspects of system operation: sensing, actuation, communication, and computation.

Communication and computational scalability is a must for systems requiring large team size. A larger team has more information sources producing more data which must be processed and shared. The need to use environment information within the sensor field exacerbate the issue because the number of information consumers also grows with the size of the network. The need for higher information accuracy may lead to more computation. In large systems, this requires computational scalability. The need for lower latency often demands more communication. In large systems, this translates into the need for communication scalability.

Large systems operating over long period of times must be fault tolerant. Reliance on small

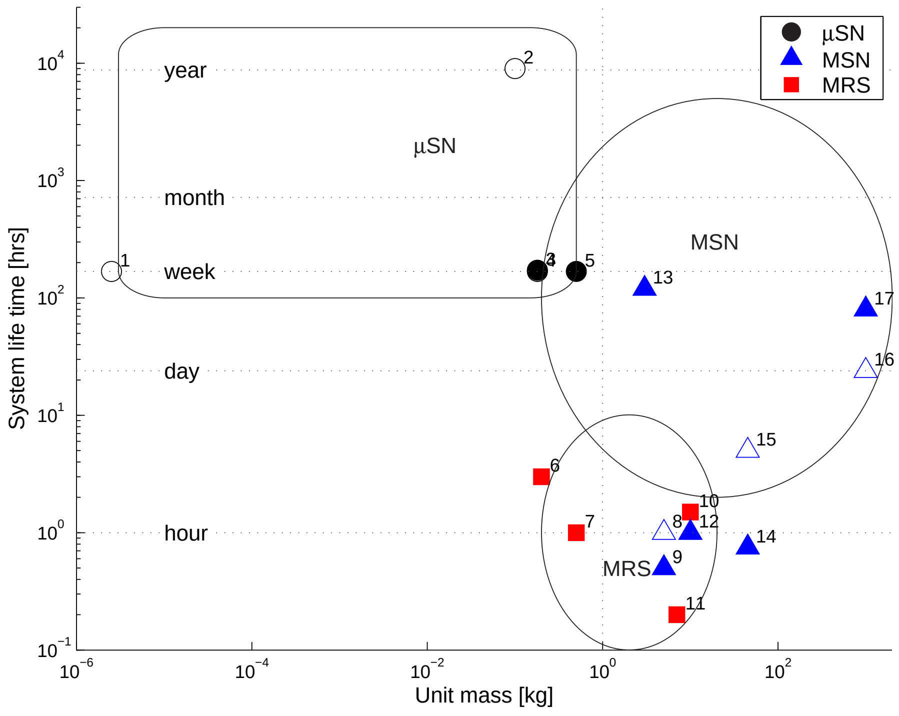


Figure 2.8: Sensor network examples: unit mass vs. mission time. Both experimental (filled markers) and simulation (outline markers) are shown.

and cheap devices may make this issue more pressing. The need for reconfiguration is somewhat related. A system must be able to reconfigure after parts of it fail, but large systems need to be able to self-configure to start functioning even before any of the components fail. Another reason to reconfigure is a change in the operating conditions — a common occurrence in long-lived systems.

Heterogeneous systems impose the requirement for interoperability at the platform level. In extreme cases, the team makeup is not even known at the time of the system design, yet cooperative operation of the individual elements is required. Unless created by a single designer, large systems are more likely to require inter-operation.

Information accuracy is harder to achieve in domains with low signal-to-noise ratio. This leads to the need for effective use of all available information. When low information latency is required, communication delays becomes an issue which imposes constraints on the acceptable network topology.

Lastly, a cautious statement can be made on the build-time properties of the SN types.

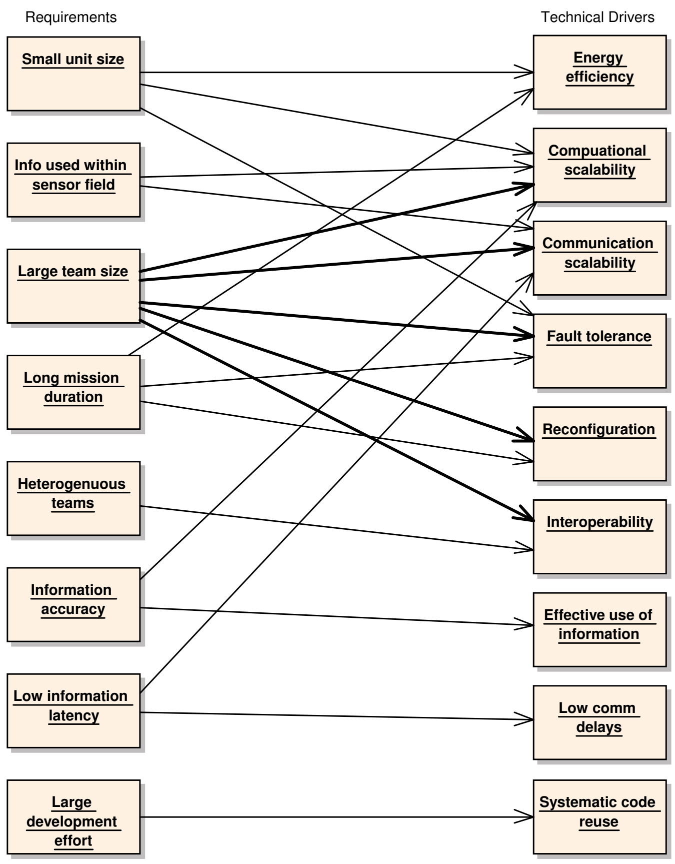


Figure 2.9: System requirements and technical drivers for a broad range of sensor networks.

It seems that, compared to other two types, MSN's entail an exceptionally high level of development effort[13] and a correspondingly high need for *systematic code reuse* [197]. The μSN severe computing constraints do not leave any room for fancy object-oriented abstractions.[14] Also, with very large-scale homogeneous installations, the economy of scale permits custom, low-level solutions. The MRS teams are heterogenous but small, so the effort to field a system is correspondingly lower. It is too early to tell if these implementation differences persists.

Some general observations about the trends in solution approaches are listed in Table A.3. Projects in the μSN and MSN categories tend to use decentralized architectures because they are intended to be eventually used with very large teams. μSN systems have limited, if

[13] "Never again!" was the comment by one of the lead developers of the Centibot project on the prospects of continuing this research. The project simultaneously fielded up to 66 indoor robots [114] (Heard by the author during a conference presentation).

[14] For example, a very common MICA-2 platform has 128 KB of memory [225].

any, information fusion, prompted by low available power. MRS approaches are centralized or hierarchical in both information fusion and decision making. This is explained by small team sizes and strict latency requirements imposed by the application domains.

The following sections describe related work in each SN category. In addition to describing representative applications and solutions, some cautious generalization are made.

2.4.2 Micro Sensor Networks

Information gathering is the main task of μSN. Small unit size, long network life, and large-scale environment coverage are critical performance metrics for this SN type. Typical applications are environment sampling, feature detection, surveillance, and monitoring. Other applications have been suggested, such as cooperative navigation aid [79] and cooperative aggregation [51].

Numerous indoor and outdoor detection and monitoring applications using wireless μSN are described in the literature [1, 28, 45, 80, 151]. The size of the individual hardware modules range from matchbox to shoebox size. An extreme in both miniaturization and the proposed extent of coverage is described in a study of a global weather observation system [143]. The network is comprised of billions MEMS-fabricated sensor nodes dispersed at high altitudes and capable of floating in the air. The system is designed to route observation to a collection station and does not perform any information fusion itself.

Traditional AI distributed problem solving techniques have been applied to the problem of vehicle detection and tracking [90, 125]. A hierarchical system passes the target from one region manager to the next. Self-configuration of nodes into regions is performed on-line. The information fusion role is “lightweight”, i.e. essentially deterministic.

The Xerox Palo Alto SN group has advocated the use of decentralized algorithms [129, 247, 248]. In dense static networks, decision making is limited to dynamically defining and forming sensor groups based on task requirements and resource availability. The Information Driven Sensor Querying (IDSQ) algorithm [30, 247] formulates tracking as a distributed constrained optimization problem that aims to maximize information gain while minimizing communication and resource usage. The task is to select an optimal subset of sensors and an optimal order of incorporating these measurements into a belief. In order to reduce communication, this selection is made based upon sensor position, observation model and predictions of contributions given the current belief.

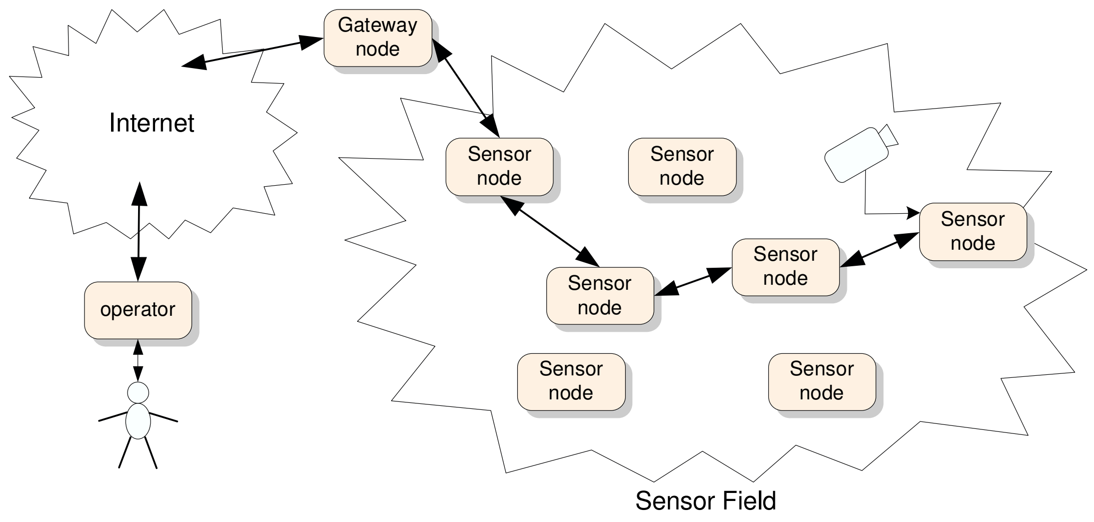


Figure 2.10: Information flow in a typical μSN (adapted from [3].)

Figure 2.10 illustrates the typical μSN approach to solving the DIG problem: routing information from producers to interested consumers. The consumer may be a mobile node or a "gateway node" capable of extracting information from the SN [214]. It is often assumed that consumers have relatively high processing capability and are connected with wired infrastructure [220]. Belief maintenance is typically not performed within the network and is left up to the information consumer located outside of the sensor field. When in-network aggregation is performed, it usually amounts to discarding duplicate observations (see [115] and a survey in [3]). Decision making is limited to node (de)activation and is usually not cooperative.

The most influential driver in the design of μSN networks is power [2, 103, 173]. This is emphasized by the abundance of energy-aware routing and topology control algorithms [51, 179, 201, 205, 228, 241]. Communication is typically multi-hop to exploit the significant reduction in energy requirements with reduction in range [151]. High node density follows logically from the fact that the platforms are stationary and both communication and sensing ranges are short. In typical deployment scenarios a dense network is required to ensure adequate coverage for both the sensing and routing functionality [179].

Two solutions to the DIG problem in static μSN are described below: Directed Diffusion, a routing protocol developed at UCLA, and MiLAN, sensor network middleware under development at University of Rochester.

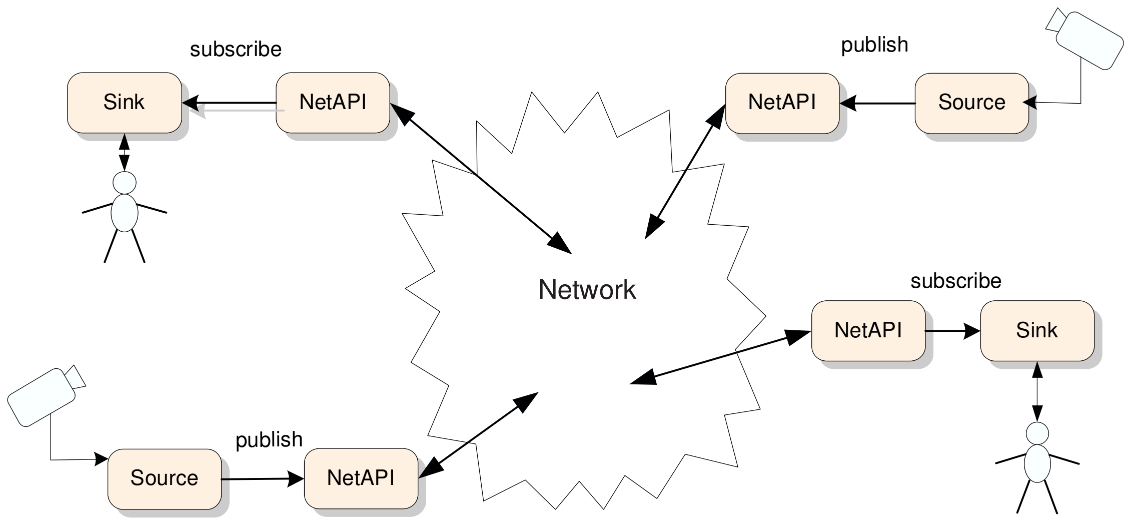


Figure 2.11: Information flow in a system using Network API for Directed Diffusion algorithm.

Directed Diffusion (DD) [99, 100] is a data-centric routing protocol, where sensor nodes are not addressed by their addresses but by the data they sense. It uses a publish and subscribe mechanism to match information sources with information sinks. Information sources advertise the data attributes of observations made while information sinks describe attributes of the information they are interested in. The information source attributes and information sink interests are propagated and matched throughout the network. Routing pathways between the sources and sinks are established as shortest paths in the network connectivity graph. Limited information fusion ("aggregation") is possible. In practice this usually amounts to discarding duplicate observations. Software tools allow easy integration of sensor platforms into the network. Figure 2.11 shows information flow in a system implementing Network API [208]. The localized interactions allow the protocol to scale to large networks. DD scales as a function of the number of active interests present in the network [227]. Efficiency of the DD algorithm for systems with approximately equal number of belief sources and sinks (e.g. each platform has a source and a sink) has not been investigated.

MiLAN [27, 77] is sensor network middleware. Figure 2.12 shows a block diagram of the

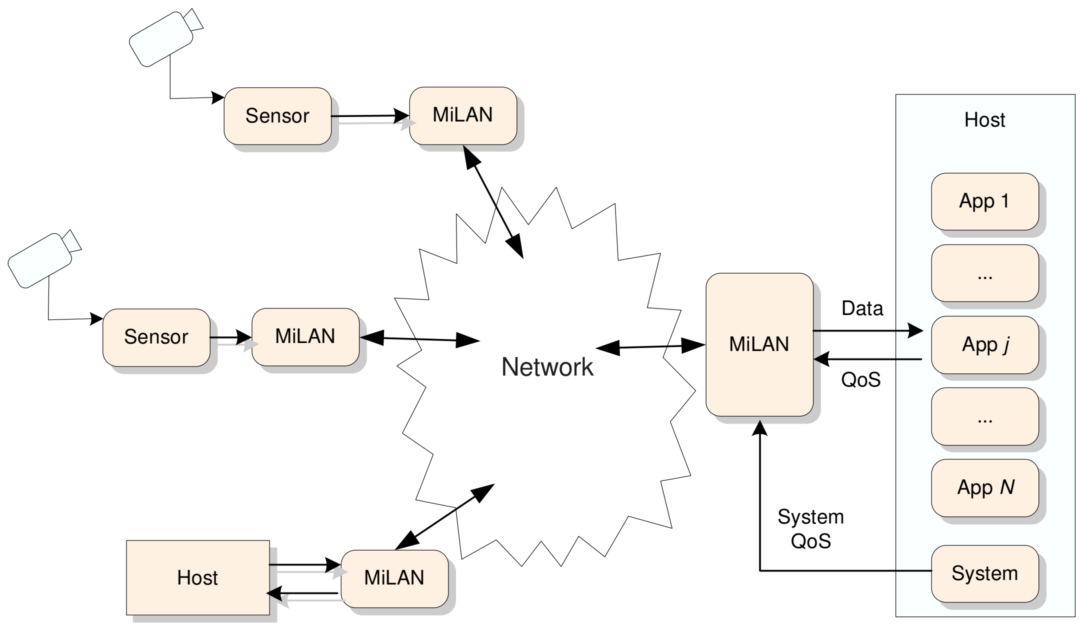


Figure 2.12: Information flow in a system using MiLAN middleware (adapted from [77].)

system. The sensors (service suppliers) and the application (service consumer) are nodes in the network. The middleware is the software that connects the sensors to the application through the network. The application has two integrated modules, the data fusion module and the decision module. The application sends its quality-of-service requirements to the middleware. The middleware forwards the sensor observations to the application. At the same time, MiLAN decides how best to configure the network to support the applications. The implementation of the system is under development.

In summary, typical infrastructure consists of a large number of small homogeneous devices with limited energy resources and computing power, and no actuators. Architecture is typically decentralized or hierarchical. Few systems utilize active platforms so there is limited need for cooperative decision making. All routing and configuration algorithms are decentralized. Some of the systems are built using a set of heterogeneous software components which self-assemble at run-time [28, 58, 201]. Implementations are not component-based, constrained primarily by limited on-board computing resources and development tools. The main performance-limiting factor is the amount of available power.

2.4.3 Multi-Robot Systems

In Multi-Robot Systems, establishing a SN dedicated to information gathering is not a goal in itself but a prerequisite for performing a domain task. Table 2.2 lists information accuracy and latency as critical performance metrics of this SN type. Typical applications are short-duration robotic missions, such as small area exploration, RoboCup games, formation following, and robotic construction.

The RoboCup domain is a good example of a MRS. Robotic soccer players can benefit from exchanging information about the position of the ball, the team mates, and the opponents. Mission duration is fixed and relatively short (10 minutes). The team is homogeneous, its size and make-up is predetermined and static. At the same time, the action is very dynamic, with strict requirements for low latency and high information accuracy. All of this leads to centralized or broadcast solutions to information fusion [43].

The task of multi-robot mapping has received substantial attention over the years [34, 132, 243]. Typical in many ways is a three-robot heterogeneous team[15] used to explore a large vacant building [211]. Both information fusion and decision making aspects of the operation are fully centralized. A typical mission duration is reported to be 8 minutes. The composition of the team does not change during the mission or from run to run.

Somewhat related is the application of indoor robotic reconnaissance missions [66, 194]. Both information fusion and control architectures are centralized or hierarchical. Forms of Bayesian information fusion are often used. Teams are typically static, so there is no need for run-time reconfiguration.

The field of Intelligent Spaces [14, 122, 238] is primarily concerned with micro-sensors but some of the applications are closer to traditional robotics. For example, an interesting application of environment sensing to direct navigation and control of indoor mobile robots is reported in [89]. Tight latency constraints (closing navigation loops around off-board sensors) lead to a fully centralized solution: all off-board cameras are connected to a single computer processing all video feeds. A similar application, but without any information fusion, is described in [121].

In distributed decision making, most efforts in robotics have focused on coordinating actions

[15]Systems with much higher number of robots have been fielded recently. These are covered in the next section.

of individual robots without explicit information fusion, for example in path planning [4]. Tighter coupling is needed in transportation and formation following [7, 65, 96]. In these domains, control-theoretic approaches are often applied to either sensor-actuator pairs or to strict platform hierarchies. A conceptual multi-layer architecture for the task of robotic construction is described in [209, 210]. It combines multi-agent planning and tightly-coupled motion control in a structure not dissimilar to the popular three-tier single-vehicle architectures [62].

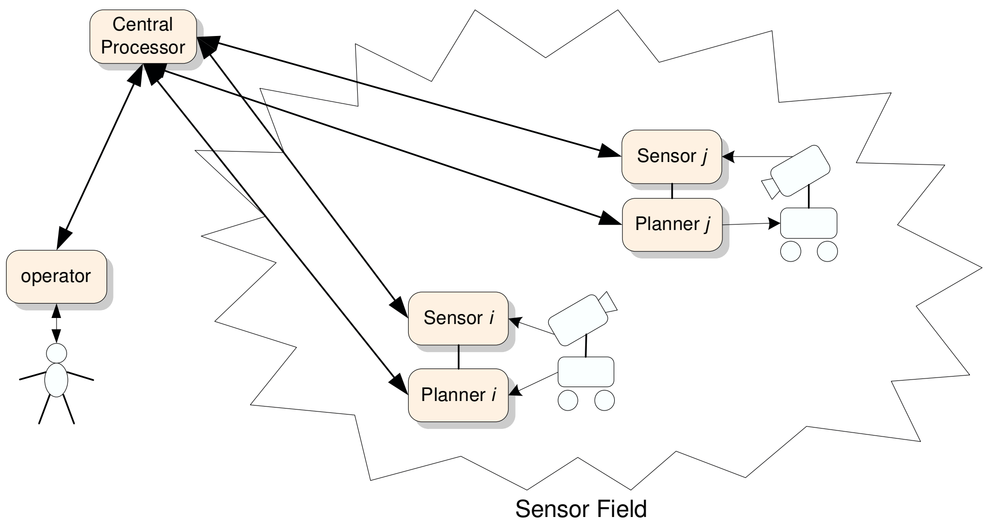


Figure 2.13: Information flow in a typical MRS. For larger team sizes, the centralized architecture may be replaced by hierarchical in which case one of the platforms takes on the role of information fusion and planning coordination.

In summary, a typical MRS infrastructure consists of a small number of mobile heterogeneous robots. Organization of a typical system is illustrated in Figure 2.13. The environment information is used within the sensor field by decision making algorithms which are executed on the platforms themselves. Low latency is possible with the use of a centralized architecture. The centralized approach also simplifies operator's access to the global view of the state of the system. Both information fusion and decision making algorithms are probabilistic. Self-configuration is rudimentary because of the small team size. Possibly for the same reason, software implementation is typically not component-based. Performance of the system is usually limited by domain-specific algorithms and not by the task of information gathering.

2.4.4 Macro Sensor Networks

Information gathering is the main task of MSN. Table 2.2 lists long mission duration and high information accuracy as the critical performance metrics for this SN type. Typical applications are mapping, patrol and surveillance, and search and rescue.

A multi-agent approach to distributed outdoor surveillance is described in [42, 196, 216]. Environment representation is probabilistic but maps are not shared between platforms. Mission planning is performed individually by each platform and a decentralized plan merging algorithm is used to detects and resolve possible collisions.

The scale of indoor multi-robot systems has increased significantly in recent years [94, 114]. An unusually high number of robots, at some stages of the operation up to 66, requires solutions typical of Macro Sensor Networks: hierarchical implementation of both information fusion and decision making, and self-configuration algorithms. Sophisticated probabilistic algorithms for localization, mapping, and exploration were executed in a distributed environment.

The approach to information fusion described in this thesis is based on the Decentralized Data Fusion (DDF) algorithm [68, 181, 184, 186]. The original implementations predate off-the-shelf wireless networking and utilized *Transputers* for communication (high performance microprocessors which could be connected into a network with serial links). More recently, a practical DDF system was developed as part of the ANSER project [163, 221]. A team of up to three of custom-built Unmanned Aerial Vehicles (UAV) performed localization of ground features. Environment representation was limited to points with Gaussian uncertainty. The motion of the platforms was preplanned. The team makeup was static and communication topology was pre-configured.

The approach to decision making described in this thesis traces back to the information-theoretic sensor management algorithms implemented on OxNav, an indoor mobile robot with a fully decentralized architecture [15, 26, 161]. Several fully decentralized algorithms for assigning sensors to beacons were developed as part of that project [145]. Grocholsky extended this approach to sparsely connected control architecture and introduced the notion of *negotiation filters* [70, 71] which are used for cooperative information-based decision making. At the same time, the approach was extended to continuous problems like path control of sensing platforms.

A similar approach is taken in [21] where decentralized Bayesian algorithms are applied to the problem of coordinated search. A simulated system uses a fully connected architecture whereby all sensing platforms broadcast their observations. Self-configuration is not part of this work.

The objectives and some of the basic principles of the Tactical Component Network (TCN) [36] are similar to the approach in this thesis. TCN advocates modular information processing, in-network information fusion, and use of information within the sensor field. Most architectural and algorithmic details, however, are unavailable due to the commercial nature and defence application of the system.

FireMapper [87] is a commercial system developed to provide real-time updates on the spread of forest fires. The information is gathered by manned aircraft taking high-quality IR images of the burning forest. The system architecture is centralized, with a single ground "fusion center", a human operator visually overlaying the images onto a digital terrain map. The tasking and control of the sensing platform is manual.

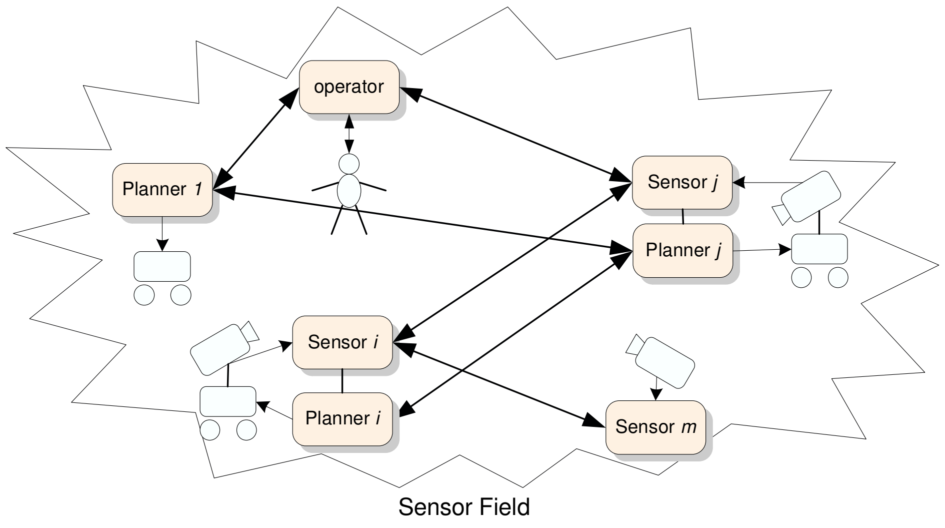


Figure 2.14: Information flow in a typical MSN.

In summary, typical MSN infrastructure consists of a large number of mobile heterogeneous platforms. Figure 2.14 illustrates the information flow in a typical MSN. The environment information is used within the sensor field by the controllers and the operators. The limiting factor is often computing power and communication bandwidth.

2.4.5 Hybrid Sensor Networks

Combining strengths of different SN approaches is an obvious step but the theoretical and practical aspects remain largely unexplored.

Very heterogeneous teams have been investigated in MRS applications. In an indoor application, a 50 kg ATRV-Jr vehicle deploys and supervises a group of 11 custom-made 200 g micro robots [194]. This design leads naturally to hierarchical information fusion and decision making architecture. In practice, this means centralized algorithms because the teams work independently from each other.

There are also examples of heterogenous μSN's. Em* is a software environment created at USC for developing and deploying Wireless Sensor Network (WSN) applications [64]. Sensors run on Mica-class micro devices while processing is done on Linux-class hardware platforms called *microservers*.

Distinction between energy-rich mobile platforms and energy-poor stationary sensors is acknowledged in [214]. In this simulation, the network consists primarily of stationary platforms, with a few mobile platforms randomly distributed through the environment. The focus is on the routing algorithms. Communication between the stationary and mobile platforms is optimized to minimize energy usage on the stationary nodes, which is the limiting factor for the whole system.

The concept of mixed stationary/mobile solution is explored futher in [202]. The system consists of low-power stationary sensors typical of μSN and mobile platforms with sufficient power and capable communications called *data MULEs*. The MULEs roam the environment and "pick up" the data collected by the sensors, thus saving the energy which would have been spent on wireless communication. Detailed algorithms are not discussed and no implementations exist. In particular, the "MULE-to-MULE" communication is listed as future work. This is the area where information fusion may be particularly beneficial.

2.5 Towards a Systems Approach to Sensor Network Design

Up until now, the infrastructure has been used to categorize the problem and is therefore considered to be specified *a priori*. This is not an unreasonable assumption: the "hardware" is often chosen before the "software" due to financial, historical, or other non-technical reasons. The choice of hardware, in turn, dictates the choice of architecture, algorithms, and software implementation.

From the system-theoretic view point, the infrastructure was specified as a constraint in Section 2.2.2. More appropriately it should be considered as part of the design space by considering different architectural alternatives and evaluating them against the requirements. One approach is what is known as Architecture Trade-off Analysis Method (ATAM) [33]. Selection of the best solution generally involves some compromise, but it is best to make this explicit.

From the description of the SN types it may appear that a particular problem admits one of the three approaches. In some applications, however, the appropriate approach depends on several parameters. The example below concentrates on just two performance metrics: area coverage and cost. Notably, the issue of information quality supplied by different platforms, sensors, and algorithms is ignored (but see e.g. [40]). A complete study of this "system-of-systems" approach remains a topic of future work and it will likely lead to hybrid solutions.

Example 2.1

Consider the scenario of area surveillance drawn from the flight program of Brumby UAV as part of the ANSER program [221]. The sensing task is to monitor a rectangular area 1.2 by 1.6 km in size. The objective of this example is to evaluate alternative types of sensing platforms in terms of overall system cost. We assume that the physical size of the platform is not a factor in itself. Four types of platforms are under consideration: a 0.5 kg stationary unattended ground sensor, a 15 kg ground vehicle of Pioneer class, a 500 kg ground vehicles of Argo class, and the 45 kg Brumby UAV which was actually used to complete this task. The platforms are likely to carry sensors of different capabilities and modalities which leads to varying information quality. This factor is not considered in this example.

A simple model of system cost is based on the number of platforms required for full dynamic coverage

$$C = C_i \frac{A_{tot}}{A_{dyn}}, \tag{2.34}$$

where C_i is the platform's unit cost, A_{tot} is the total area to be patrolled, and A_{dyn} is the area covered by a platform in a certain amount of time (e.g. 6 minutes). The parameters

Parameter	μ-Sensor	Pioneer	Argo	Brumby
Unit cost ($US)	200	14,000	30,000	188,000
Operating cost ($US/hr)	0	20	300	1000
Static sensor footprint (m^2)	1256	10,000	10,000	1365
Typical speed (m/s)	0	1	5.5	50
Sensor sweep (m^2/s)	0	160	880	8600
Dyn. sensor footprint, 6 min (m^2)	1256	67,600	326,800	3,100,000

Table 2.3: Platform performance and cost parameters used in the example. The dynamic sensor footprint is the area observed by the sensors in a certain amount of time as the mobile platform is moving. For stationary platforms, the static and dynamic sensor footprints are the same. For a fast low-flying UAV, the dynamic footprint is several orders of magnitude larger than the static one. The cost estimates include sensor packages.

used in the cost model are given in table 2.3.

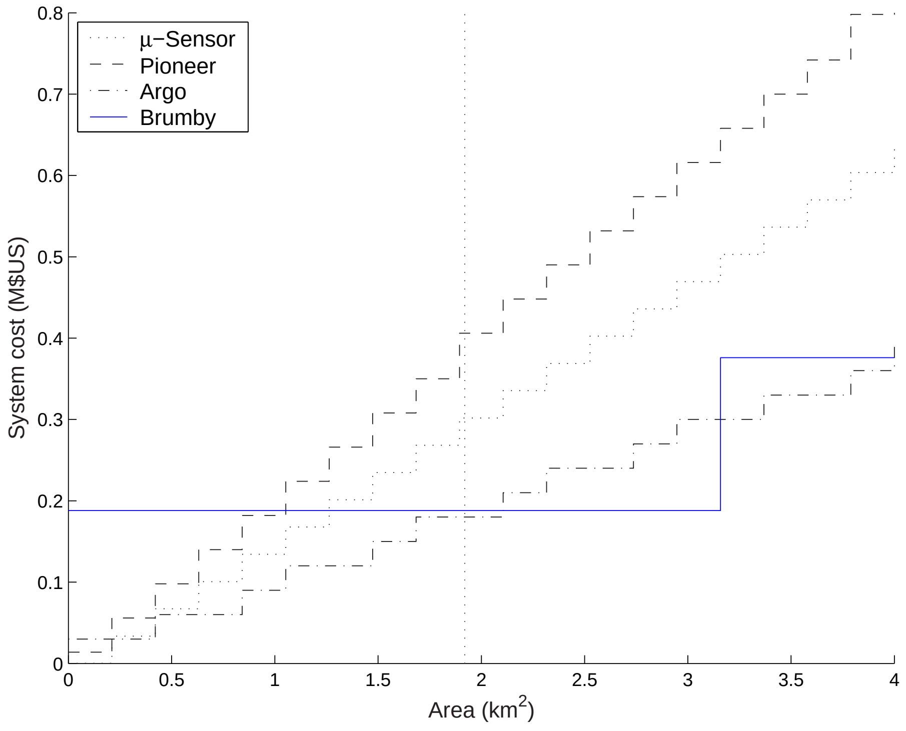


Figure 2.15: Sensor network cost as a function of infrastructure type and the area to be patrolled. The number of platforms is driven by the requirement of full area coverage every 6 minutes.

Figure 2.15 shows the estimated system acquisition cost as a function of the area to be patrolled. The nominal area of 1.92 km^2 is marked with a vertical line. For this area, a *single aircraft is sufficient to observe the entire site every 6 minutes. The cost of the aerial surveillance system compares well with other solutions, while for areas under 1 km^2, it is the most expensive option.*

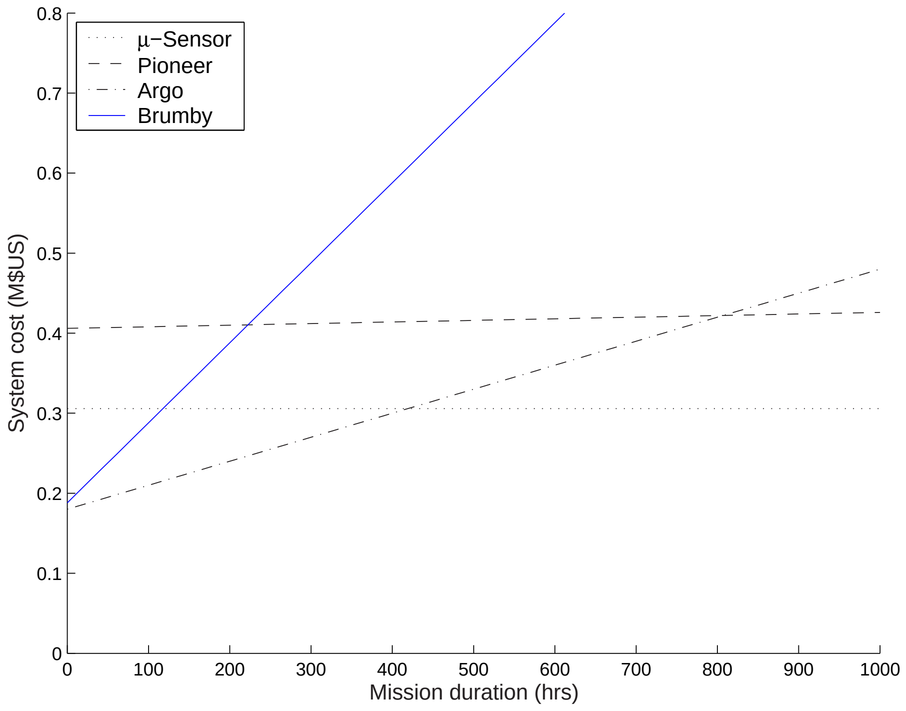


Figure 2.16: Sensor network cost as a function of infrastructure type and mission duration.

During long-term mission maintenance costs cannot be ignored. The cost model is modified to include the operating expenses

$$C = (C_i + c_i t)\frac{A_{tot}}{A_{dyn}}, \tag{2.35}$$

where c_i is the cost of operating one platform per unit time and t is mission duration. Figure 2.16 looks at the cost of different infrastructure options in terms of long-term operation. The cost from Figure 2.15 for the nominal site area are taken as the initial values and operating costs are used to plot the system cost as a function of the mission duration. Due to its high maintenance cost, the UAV-based system becomes the most expensive option for missions longer than 220 hours. This model does not take into account the possible cost of replacing ground sensors whose failure rate is likely to increase with time.

This example illustrated how the choice of SN infrastructure can be an integral part of the *design process, dictated by the task and the environment.*

2.6 Summary

This chapter stated the problem of Distributed Information Gathering, described and classified related work, and identified desired solution properties.

A probabilistic reference model established the structure of the following discussion and the notation used in the rest of the thesis. Based on this reference model, the solution space was categorized by listing a number of options available to the designer. Five key aspects of the solution were singled out: system architecture, information fusion method, utility evaluation method, decision making method, and system implementation.

Based on the survey of related work, three types of systems were identified: Micro Sensor Networks, Macro Sensor Networks and Multi-Robot Systems. The clustering is explained by both objective drivers as well as accepted practices. This thesis aims at designing a system which satisfies the requirements for Macro Sensor Networks.

The following chapter gives an overview of the proposed solution framework and describes its architectural aspect. The MSN requirements will serve as a starting point to justify the architectural choices.

Chapter 3

Active Sensor Network Architecture

3.1 Introduction

The Active Sensor Network (ASN) framework is a system solution to the DIG problem stated in Chapter 2. Three interrelated aspects of the system are illustrated in Figure 3.1: architecture, algorithms, and implementation. All three aspects must be tailored to fulfill the requirements for Macro Sensor Networks. This chapter focuses on the ASN architecture. It describes and justifies the architectural choices and shows how these choices lead to the fulfillment of the functional and non-functional requirements.

Large complex systems are difficult to design and, possibly, even more difficult to document. While the design methods tend to be domain specific, the approaches to architecture documentation are quite general and a large body of literature on the subject exists [32, 88, 117, 215]. The basic approach is to describe an architecture as a set of *Views* — representations of a whole system from the perspective of a related set of concerns [167]. The breakdown into views according to different methods varies in details but the results are largely equivalent.[1]

The architecture description in this Chapter follows the procedure developed by Bredemeyer

[1]For example, the 4+1 View Model [117] distinguishes between the logical, process, physical, and development views. Compare to the Siemens Four View model [215] with a breakdown into conceptual, module interconnection, execution, and code views.

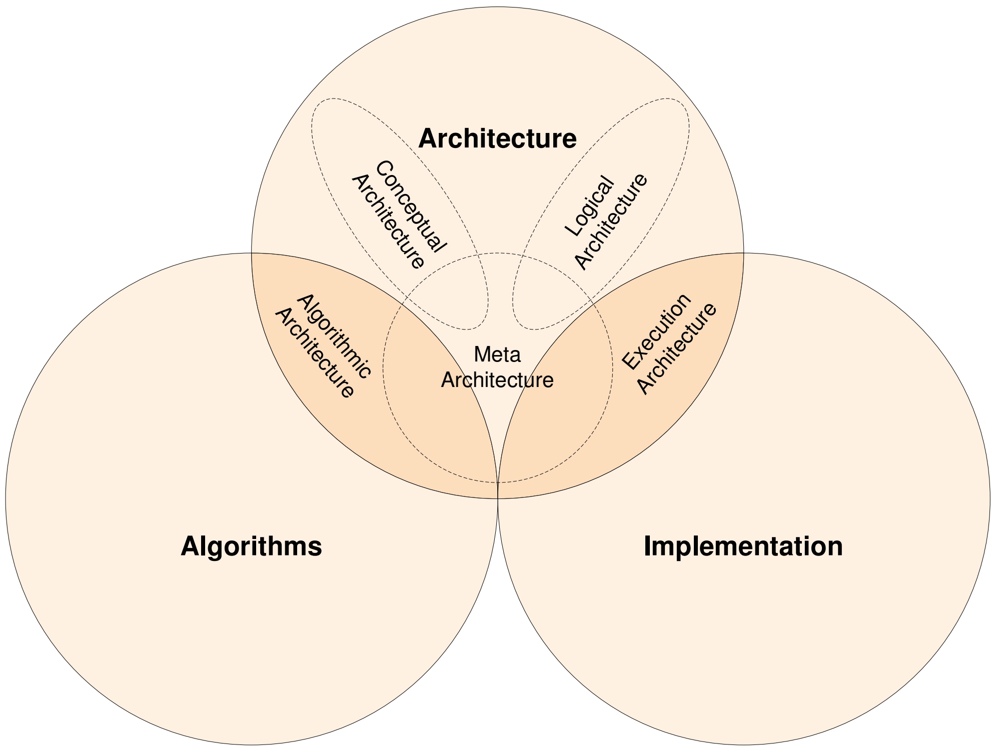


Figure 3.1: Three aspects of Active Sensor Network. The importance of meta architecture is indicated by its central position in the diagram. Note that efficient implementation of algorithms is an important area but it is not addressed in this thesis.

Consulting [139, 141, 142]. The procedure divides the architecture description into four layers: Meta, Conceptual, Logical and Execution. For a computationally-intensive application like information gathering, an additional, Algorithmic, layer is added. The relationship between the architectural layers and the three aspects of the complete solution are shown in Figure 3.1. It shows graphically that the Algorithmic and Execution architectures establish the high-level structure of the algorithms and implementation of the system.

The hierarchy of the five architectural layers, and the associated design concepts, are shown in Figure 3.2. This figure contains all concepts and UML symbols used in this thesis. The term *component* requires some clarification, despite its heavy use in the literature and a definition within UML. The following definition is adopted in this work: a software component is “a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently” [224].

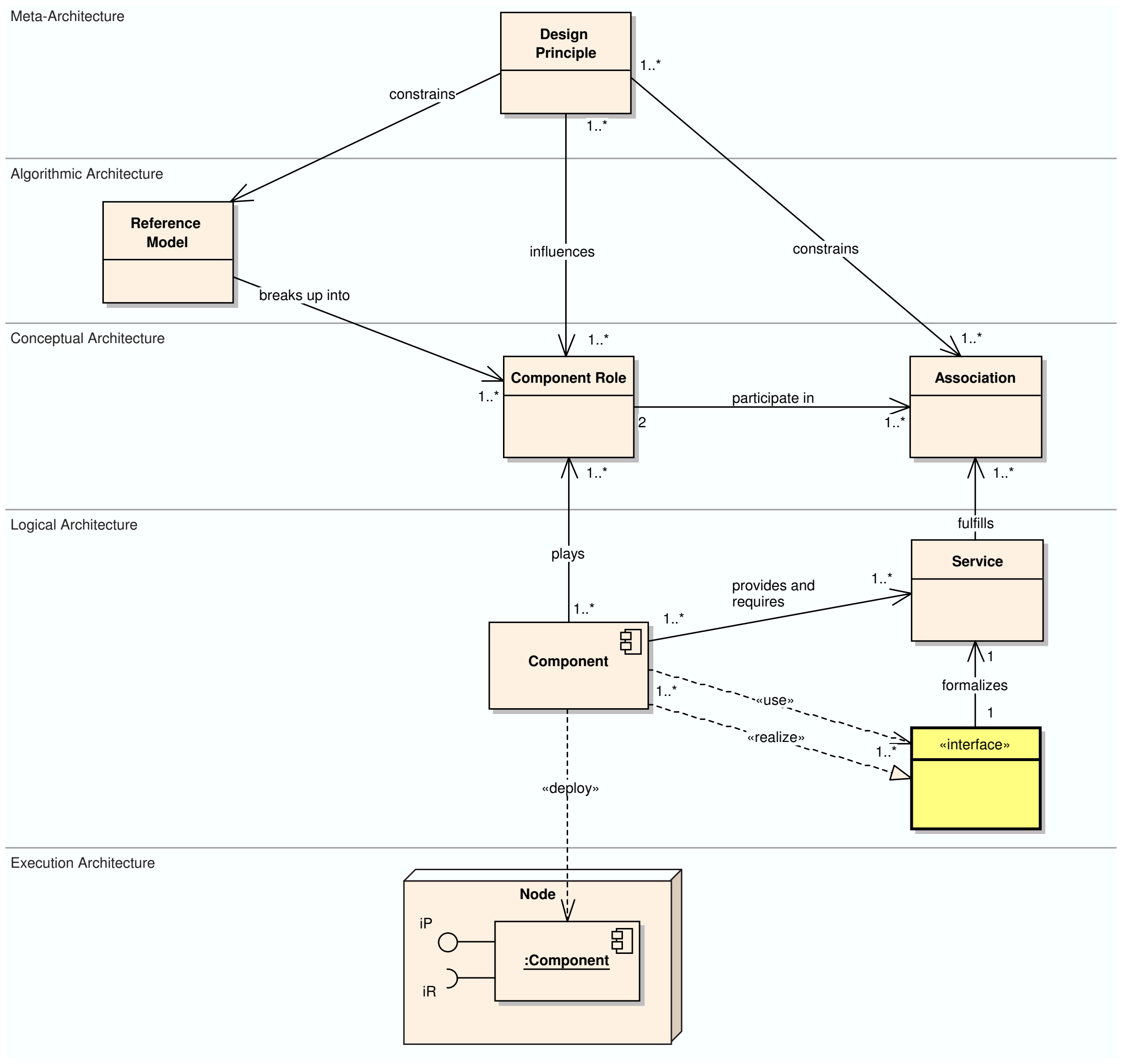


Figure 3.2: Summary of the layered architectural description with associated design concepts. The UML symbols in the diagram are applied consistently through the rest of the thesis. Starting at the top, general design principles influence and constrain the reference model, the component roles and the interactions between them. Components occupy a central position: they play one or more roles in the operation of the system and interact by exchanging services. Services are formally defined with interfaces. Interfaces are highlighted as the most versatile method of specifying the architecture. Components may provide (*realize*) or use interfaces. Finally, components are implemented as runtime software elements (executables, libraries) and deployed onto physical hardware. Provided interfaces (iP) are shown as circles, required interfaces (iR) as half-circles.

In UML it is represented by a *component* element.[2]

Note that the *description* of the architecture in this chapter and the rest of the thesis is strictly top-down: from guiding principles to implementation. The actual *design process* was, of course, not quite so linear. There is some indication, however, that for large frameworks, the top-down approach is more natural than bottom-up: proceeding from what is known to what is unknown [224]. That is, the description of the target domain codified in UML use cases is typically well understood, whereas the structure, interaction rules and roles of the framework domain are not. Exposing this structure is precisely the goal of this chapter.

The architecture description begins in Section 3.2 with *Meta-architecture*: a set of high-level decisions that strongly influence the structure of the system. The *style* of the ASN architecture is established by listing its distinguishing features: decentralization, modularity, and the use of strictly-local interactions. The rationale for these choices is explained in terms of their contribution to the fulfillment of the MSN requirements, namely large team size, active heterogenous platforms, and long mission duration. The influence of the meta-architecture is far-reaching: the communication and structural patterns defined here are repeatedly applied across the architecture ensuring a consistent approach.

Section 3.3 focuses on *Algorithmic architecture* which is obtained by applying the Meta-architecture items to the DIG reference model. This results in a substantial narrowing of the design space. The main emphasis is on ensuring that the principles of decentralization, modularity, and locality are adhered to on the algorithmic level. The resulting algorithms follow the work of Grime [67, 68], Nettleton [163, 164], Manyika and Durrant-Whyte [145] and Grocholsky [70, 71]. Unlike in the existing work, the system is described in a general Bayesian form. The details of the two main algorithms, for information fusion and decision making, are described separately, in Chapters 4 and 5 respectively.

Section 3.4 describes *Conceptual architecture* which begins the process of structuring the so far monolithic system into a network of interchangeable collaborating components. The first step is to determine the *granularity* appropriate for this problem, i.e. the size of the

[2]`UML` Prior to UML version 2.0 [75] there was a considerable ambiguity between the *logical* and *physical* components. The former were usually represented with UML packages stereotyped as ≪subsystem≫. UML 2.0 offers an improved support for component-based design by extending the use of the proper *component* element to all stages of a components life cycle: from conceptual design to deployment. The new notation is used throughout this thesis.

atomic elements. The approach taken in this work is to select *functional roles* to form the fundamental composition level. Interactions between the roles are constrained by the three meta-architectural principles. The roles are further assigned to three largely-independent groups: information fusion, decision making, and system configuration. Algorithmic and functional structures are matched by mapping the algorithms from Section 3.3 to the functional roles.

Logical architecture described in Section 3.5 is the detailed architecture specification. Six representative component types, termed *canonical*, are introduced, each playing one or several of the previously defined roles. Components interact by providing or requiring services. A set of *interfaces* is defined to formalize inter-component services. Interfaces are contractual obligations between service providers and service consumers and they define the ASN framework. Component and interface definition completes the process of decomposing the system into its building blocks. Next, the components and the interfaces are put back together to demonstrate how the use cases of the DIG problem are realized. The important question of the degree of decentralization in information fusion and decision making is revisited and illustrated with canonical components.

Section 3.6 is dedicated to describing *canonical* components. The mapping of roles to components is not unique, but practical experience has shown that the six components are sufficient for implementing a wide range of systems. All simulation and experiments described in this thesis are implemented with these canonical components. The description of components makes a clear distinction between component specification and realization: a single component specified by the interfaces it provides and requires may have multiple realizations. This property combined with interface invariance makes components *substitutable*: any realization compliant with a component specification may be "plugged" into the system.

Section 3.7 considers the case when the freedom of realization is insufficient and there is a need to create *custom components*. This is done by modifying the existing or creating entirely different specifications. Three common approaches to creating custom components are discussed: functional integration of several canonical components, addition of extra functionality, and grouping of canonical components into *super-components*. In all three cases, the list of provided and required interfaces is modified. What remains unchanged are the interfaces themselves. Among other benefits, it preserves inter-operability between the canonical and custom components.

Execution architecture described in Section 3.8 specifies how components are mapped to execution elements and how the execution elements are mapped to the physical hardware. The overall code structure is also considered. In all aspects of implementation, the emphasis is on ensuring that the three design principles are followed. This leads to the choices of decentralized infrastructure, arbitrary deployment, and a code structure with minimal inter-dependencies.

3.2 Meta-Architecture

This section establishes the guiding principles of the architecture. The influence of the meta-architectural items is far-reaching as they are applied to every aspect of the system: architectural, algorithmic, and implementation. The principles are chosen in response to the architectural requirements of the MSN version of the DIG problem listed in Table 2.2: large team size, active heterogenous platforms, and long mission duration. The following meta-architectural items have been identified

1. Decentralization
2. Modularity
3. Locality of interactions

In general, these principles are quite common.[3] The novelty lies in applying them systematically to the sensor network domain. Each principle addresses several of the technical drivers as shown in Figure 3.3. The three principles are discussed in turn below. The rationale for adoption of a particular principle is explained in terms of its contribution to the fulfillment of the MSN requirements.

[3]In software engineering terms, an *architectural style* consistent with these design constraints is called Communicating Processes (CP) [8, 203]. This style employs independent processes with asynchronous control and sporadic data flow. Process independence implies modularity. Arbitrary topology is allowed including decentralized. On-line configuration can be achieved by specifying *run-time bindings* between processes. This style classification predates the modern definition of components but can be easily extended to accommodate it.

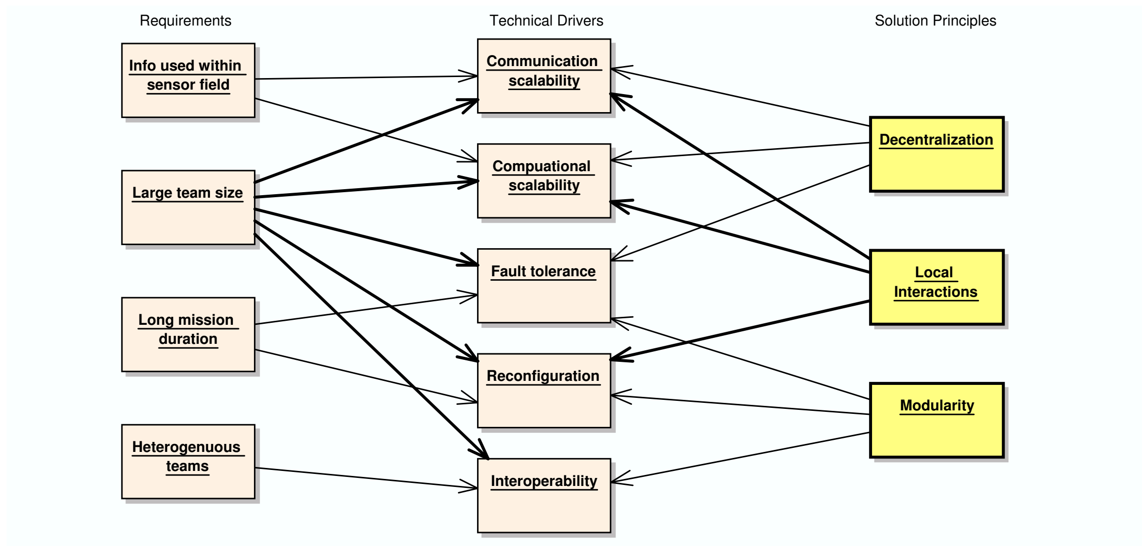


Figure 3.3: Requirements, drivers, and solution principles for an active sensor network. The requirements and drivers are a subset of those shown in Figure 2.9 corresponding to the MSN problem. The solution principles are the ASN meta-architecture items. Different line thickness is used to enhance readability.

3.2.1 Principle of Decentralization

Compared to centralized and distributed systems, a decentralized system is characterized by the following constraints [145]:

1. No component is central to the successful operation of the network. In other words, all components have equal status and the connections between them are peer-to-peer.
2. The infrastructure does not provide or rely on central facilities or services. This must apply to all aspects of the system: communication, user interaction, name and service lookup, timing, etc.

The resulting system offers a number of advantages over other architecture types in the area of scalability, fault tolerance, and reconfiguration.

Decentralization leads to scalability of system resources. In centralized systems, growth is limited by the capability of the centralized service providers. The server limit is inevitably reached at a certain system size and further growth is impossible without an upgrade. In decentralized systems, each incremental increase in size is accompanied by a corresponding increase in resources. From the point of view of architecture, this approach is scalable: it allows the system to retain its functional capacity despite increase in size.

Decentralization leads to fault tolerance. Since no element of the system is mission critical, the system can survive run-time loss of individual components. In the limit of a large self-sufficient installation without any centralized infrastructure to rely on, a decentralized system appears to “live in the network”. When this is achieved, the entire system can exist without a support center and is virtually impossible to shutdown.[4]

Building systems which are decentralized in every respect is difficult and compromises must often be made in practice.[5] Different functions of the system may use different topologies. For example, the information fusion part of a system may be fully decentralized but the time synchronization service might remain centralized (via GPS) or hierarchical (via NTP protocol). Different topology may be present at different levels of abstraction. Networks, for example, are designed in layers [57] and what appears to be a peer-to-peer link at the session layer, may in fact be (efficiently) implemented using broadcast on the transport layer.

3.2.2 Principle of Local Interactions

The benefits of sparsely coupled systems are exploited in many domains, from Finite Element Analysis, to Bayesian networks [172] and graphical games [109]. The fundamental premise is that an element of the system interacts with only a small subset of all elements of the system, thus simplifying design and analysis. The principle of local interaction addresses three of the five technical drivers: communication and computational scalability, and reconfiguration.

Decentralization is not effective if the services are blindly duplicated on each platform but the interaction pattern with other platforms remains essentially centralized. To guarantee scalability, direct interactions of each platform must be limited to a subset of the team in its immediate neighborhood. If the number of communication links a platform must maintain does not change with the network size, the goal of communication scalability may be achieved.

[4]This is a boon to the military, file-swapper, and free-speech activists alike, who seek survivable [157], “un-suable” [174], and “un-censorable” [31] systems respectively.

[5]Examples of compromises can be found outside of sensor networks as well: the internet’s Domain Name Service (DNS) is peer-to-peer in protocol design but, in practice, it uses a strong sense of hierarchy [154].

In message-oriented interactions an extra step must be taken to ensure communication scalability. Even if the number of communication links does not change with network growth, the number of messages will if each platform forwards all incoming messages to its neighbors. To ensure scalability, messages arriving from one of the platform's neighbors must be combined before being passed on to the other neighbors. In this case, the number of messages and the computations required to process them does not grow with the size of the network.

Locality has a positive affect on non-decentralized interactions as well. Centralized services may be acceptable in practice if they are made sufficiently local. Nevertheless, all examples of (centralized) client-server relationships will be carefully examined, to ensure that they do not become obstacles to scalability.

Locality of interaction makes the task of reconfiguration easier. Local connections do not require the knowledge of the global network topology. This has a positive effect on the naming and routing services, as the bandwidth and storage required to maintain the name look-up and routing tables remain constant.

3.2.3 Principle of Modularity

The modular design principle is used in all engineering disciplines. The basic idea is to take a large problem and decompose it into subproblems with only limited interaction. The hope is that the subproblems are easier to solve than the original problem. In particular, the modular approach has a long tradition in software design where it offers development time reduction due to parallel efforts, easier software maintenance due to module independence, and improved comprehensibility due to imposed structure [171]. When applied to the solution of the DIG problem, the modular approach addresses three of the five technical drivers: interoperability, reconfiguration, and fault tolerance.

Modularity leads to interoperability: heterogeneous hardware and software modules can communicate and interoperate when appropriate protocols are defined. A modular approach handles heterogeneous teams quite naturally. In fact, heterogeneous modules become interchangeable and even indistinguishable if they use and expose the same interfaces which hide the physical nature and identity of the individual modules.

Modularity promotes on-line reconfiguration in response to the changing environmental conditions. A modular system must be able to configure itself on start-up. A run-time reconfiguration often becomes an extension of the start-up procedure. Modularity combined with system description creates an opportunity for *system composition* — combining and recombining elements of the system to achieve the desired performance. Only functional composition is addressed in this thesis. No attempt is made to predict or control the resulting performance although the required extension seems quite natural.[6]

Modularity contributes to fault tolerance by confining failures to individual modules. Some effort is required to avoid module failures from spreading in a "domino effect". If this is achieved, the system may retain partial functionality and full recovery is possible if the system can reconfigure in response to failures.

Furthermore, modularity on the architectural and algorithmic levels allow exceptional modularity on the implementation level. In this light, there is a good match between the ASN philosophy and a software development paradigm known as Component-Based Software Engineering (CBSE) [76, 224] or Component Based Development (CBD) [35, 116]. The vision of CBSE is one of constructing software from pre-defined building blocks, called *components*. These may be custom-built or off-the-shelf and are "wired together" at run-time in arrangements unforeseen by the original component developers. The main driver behind CBSE is the promise of systematic *code reuse* [197], which addresses an important build-time system quality: time and cost to implement the system. Code reuse is not directly addressed in this thesis, other than noting that architectural and algorithmic modularity may have positive influence on the build-time qualities of the system, such as reusability and reliability.

Finally, it may appear that there is a contradiction between the roboticist's desire to modularize and the more holistic approach of AI. The main difference between the fields appears to be in the approaches to *creating* an autonomous system. Fully integrated "intelligent" systems are envisioned to evolve, learn, emerge or otherwise come about with minimum human intervention. If, on the other hand, a complex autonomous system is to be designed, engineered, programmed or otherwise handcrafted, then there are few alternatives to modularization.[7]

[6]See [40, 78] for related work.

[7]Another possible resolution of the conflict is in looking at the problem on a different level: while humans may not be modular, human organizations certainly are (each person is expected to contribute a set of resources to the team, not the least: a brain).

3.3 Algorithmic Architecture

This section revisits the DIG problem statement and applies the ASN meta-architecture items to the DIG reference model of Section 2.3. This process imposes certain constraints and defines a family of algorithms acceptable within the ASN framework. Methods for three critical aspects of the reference model remain to be specified: belief fusion, utility fusion, and policy selection. The ASN "philosophy" for each of the three aspects of the solution are outlined first.

Belief fusion philosophy. There are at least four approaches to decentralized belief fusion. An individual platform can

1. communicate all observations or beliefs to all other platforms [21, 182, 183];
2. communicate some (appropriate) observations to some (subscribing) platforms [99, 100];
3. communicate all beliefs to some (neighboring) platforms [67, 68, 163]; or
4. communicate some (appropriate) beliefs to some (querying) platforms [190, 244].

The first approach is not suitable for ASN because the interactions between platforms are not local. The second approach is not modular. Scalability properties of the fourth approach have not been fully investigated. Thus in terms of belief fusion this thesis follows the third approach of Grime and Nettleton called Decentralised Data Fusion [67, 68, 163, 164] because it is decentralized, modular, and all interactions are local.

Utility fusion philosophy. In terms of utility fusion, this thesis follows the approach of Grocholsky in separating the platform's partial utility into the *team* utility of belief quality and *local* utilities of action and communication [70, 71]. The advantage of this approach is that the expected future beliefs, just like in the current beliefs, are easy to combine through belief fusion. Thus the difficult problem of utility fusion is circumvented by fusing future expected beliefs and not the partial utilities themselves.

As a result of this approximation, the main objective is achieved: this approach allows decentralized, modular, strictly-local cooperative decision making. The downside is that the potential coupling between individual actions and messages is ignored because the utilities of action and communication remain local. Even without this coupling, the approximation produces suboptimal results in heterogeneous teams because the cost of local action and communication is not accounted for on the team level.

It is worth emphasizing that inherent in the reference model definition is the notion that utility coupling is the only source of action coordination. Coupled dynamics and constraints are two common alternatives. Coupled dynamics give rise to tight cooperation typical for formation following and manipulation [7, 65, 96, 210] as well as (non-robotic) decentralized control [168, 195]. Distributed constraint satisfaction techniques [245] have been applied to modular robots as well [60, 160, 161]. Tightly coupled cooperation is not part of the DIG problem but some constraints can be incorporated into the utility formulation by means of penalty functions.

Policy selection philosophy. The communication and action policies are chosen by maximizing expected values. The field of distributed decision making is very broad and a complete survey is beyond the scope of this work. Two basic approaches to solving the value maximization problem are possible:

1. Function maximization of the entire policy mapping at once: from all possible belief states to all possible actions. This approach is typical for some Reinforcement Learning algorithms [222].

2. Point maximization for one particular state[8] [70, 71, 144, 145].

The first option is problematic because of the large state and action spaces typical of DIG problem. Thus in terms of policy selection, this thesis follows the second approach of Manyika and Grocholsky [70, 71, 144, 145].

Having stated the three big "philosophical" choices, the elements of the reference model are considered in turn eliminating the options not compatible with the ASN design principles. The result is a high level algorithm description, the *algorithmic architecture*, expressed in terms of the elements of the reference model.

[8]In decision theory this is known as *decision rule* [16].

3.3.1 Communication

There are three main questions to answer: with *whom* to communicate, *what* to communicate, and *when* to communicate. The first two questions are considered here. The question of *when* to communicate is addressed as part of the communication policy. When considering the question of with whom to communicate, the main concern is about which *platforms* to communicate with. This is because in solving the DIG problem, platforms are the units of deployment and the metric of scalability is defined in terms of the number of platforms.

With whom to communicate. Based on the principle of decentralization, all inter-platform communication in ASN is restricted to peer-to-peer mode. Based on the principle of locality, all inter-platform communication is limited to the platform's local neighborhood η_i. All messages from platform i must be addressed to a platform j within its neighborhood, $\{m_{ij}^k\}_{j\in\eta_i}$, where $\eta_i \subset \alpha$. The definition of neighborhood is application specific. For system using wireless communication links, distance between platforms is a critical factor, but other considerations, such as platform type, are possible.

What to communicate. Based on the principle of modularity, the messages exchanged by the platforms must be understood by all other members of the team without knowing the specifics of the communicating platform's state, internal models, etc. The only universal language understood by all platforms is the world belief. This includes observation likelihoods and current and future world beliefs. These are defined as probability distributions on the world state space $\mathcal{X}$ and can be shared between platforms. Raw observations $z_i^k \in \mathcal{Z}_i$, on the other hand, cannot be easily shared between platforms because each platform has a potentially different observation state space.

Combining the two principles together, an ASN platform is limited to transmitting its local world belief b_{xi}^k to other platforms in its local neighborhood.[9] The set of messages sent by platform i to platform $j \in \eta_i$ consists of local world beliefs, current and future, and a null message

$$\mathcal{M}_{ij} = \{b_{xi}^k, b_{xi}^{k+n,k}, \varnothing\}. \tag{3.1}$$

The null message is included to allow a platform a choice not to communicate at any given time. The sets of available communication messages are identical between platforms because

[9]Thereby, communication in ASN is used only for expressive functions (e.g. my state, my beliefs) as apposed to conative (e.g. do this, answer my question), referential (e.g. third party's states) or others. [54].

the messages are defined on the environment state space, so $\mathcal{M}_{ij} = \mathcal{M}_{ji}$.

3.3.2 Belief

In terms of belief maintenance, two questions are of importance: what type of beliefs a platform maintains and which ones are synchronized with the rest of the team. The second question has already been addressed in Section 3.3.1: only beliefs on the world state are communicated and, therefore, synchronized. The belief about the platform itself is maintained locally. Each platform knows with certainty which local actions have been taken or are planned (not necessarily their results) and which messages have been sent or are planned (not necessarily received by the intended recipient).

Thus, while the reference model is open-ended in terms of the types of beliefs a platform may have, only two types of local beliefs are supported in ASN: the state of the world and the platform's own state: $b_i^k \in \mathcal{B}_i = \mathcal{B}_{xi} \times \mathcal{B}_{pi}$. And only one of them, the belief about the world, is synchronized on the team level.

Local world belief. Following the DIG problem statement, the environment states are not affected by the platforms' actions. A platform's belief about the environment is based on the prior information, the history of local observation, and the history of messages received from the team members

$$b_{xi}^k = P(x^k \mid x^0, Z_i^k, M_{\alpha i}^k). \tag{3.2}$$

When using Equation 3.2, the principle of modularity is not satisfied in the multi-sensor case because each sensor s on platform i may have a different observation space, $z_{is}^k \in \mathcal{Z}_{is}$.

From the point of view of modularity, the sensor model expressed as a likelihood function in Equation 2.7 is preferred because the distribution on the world state is independent of the observation space $\mathcal{Z}_i$. Observation likelihoods can be considered as *sensor beliefs*, $b_{li} : \mathcal{X} \to \Re$, because a likelihood can always be used to obtain a proper belief by combining it with an uninformative prior. The likelihood function is then a mapping from the state of the world, platform state and action, to the sensor belief, $L_i : \mathcal{X} \times \mathcal{P}_i \times \mathcal{A}_i \to \mathcal{B}_{li}$.

When redefined in terms of sensor beliefs, a platform's belief about the environment is now based on the prior information, the history of local observation likelihoods, and the history of messages received from the team members

$$b_{xi}^{k} = P(x^{k} \mid x^{0}, B_{li}^{k}, M_{\alpha i}^{k}). \tag{3.3}$$

The general belief update function D_i was defined in Equation 2.9 in terms of the previous belief, an observation, and a communicated message

$$b_{i}^{k+} = D_i(b_i^k, z_i^k, m_{\alpha i}^k). \tag{3.4}$$

Observations are now replaced with observation likelihoods and messages are constrained to only those containing external world beliefs. For convenience, the belief update function D_i can be separated into two parts: $D_{iL} : \mathcal{B}_{xi} \times \mathcal{L}_{xi} \to \mathcal{B}_{xi}$, which combines observation likelihoods with local beliefs, and $D_{iB} : \mathcal{B}_{xi} \times \mathcal{B}_{xj} \to \mathcal{B}_{xi}$, which combines local beliefs with beliefs communicated from another platform. When an observation likelihood b_{li}^k arrives, the local belief b_{xi}^k is updated using an iterative belief update function

$$b_{xi}^{k+} = D_{iL}(b_{xi}^k, b_{li}^k). \tag{3.5}$$

When a message from another platform arrives containing an external belief b_{xj}^k, the local belief b_{xi}^k is updated using an iterative belief update function

$$b_{xi}^{k+} = D_{iB}(b_{xi}^k, b_{xj}^k). \tag{3.6}$$

Team belief on world state. Communication between two platforms leads to fusion (synchronization) of their beliefs about the state of the world. If all platforms are connected to the network, then individual pair-wise exchanges lead to synchronization of the entire team belief. As a result, local beliefs b_{xi}^k are the same as the team belief $b_{x\alpha}^k$

$$b_{x\alpha}^{k} = \{b_{xi}^{k\pm\delta}\}_{i\in\alpha}, \tag{3.7}$$

where δ is the maximum communication delay in the system. The delay depends on many factors including the network topology. The future expected belief is synchronized in the same way as the current belief, i.e. the future belief is the same among the platforms to within a communication delay δ

$$b_{x\alpha}^{k+n,k} = \{b_{xi}^{k+n,k\pm\delta}\}_{i\in\alpha}. \tag{3.8}$$

Belief on platform state. A platform's belief about its own state is based on prior information and the history of local observations and actions (messages about platform states are not exchanged)

$$b_{pi}^{k} = P(p_i^k \mid p_i^0, Z_i^k, A_i^k). \tag{3.9}$$

The actions of platforms are assumed to be decoupled: platform actions affect only its own states, $F_i : \mathcal{P}_i \times \mathcal{A}_i \times \mathcal{P}_i \to \Re$. As in the case of the world belief, the belief about the state of the platform is updated by making observations. The corresponding observation model and the belief update function are not stated explicitly because they are confined to the individual platforms.

3.3.3 Expected Utility

In Chapter 2 it was assumed that the utilities of action U_A and communication U_M are decoupled (Equation 2.22). The utility function proposed by Grocholsky [70, 71] further distinguishes between the utility U_B, which depends only on the state of the environment, and U_A, which assigns a cost to local actions in the context of the current state of the platforms and the environment

$$\rho^k = U_B(x^k) + U_A(x^k, p_\alpha^k, a_\alpha^k) + U_M(x^k, p_\alpha^k, m_\alpha^k). \tag{3.10}$$

The partial expected utility based on local belief b_i^k is then

$$\tilde{\rho}_i^k = E^{b_i^k}\left\{U_B + U_A + U_M\right\} = \tilde{J}_B(b_{xi}^k) + \tilde{J}_A(b_{xi}^k, b_{pi}^k, a_i^k) + \tilde{J}_M(b_{xi}^k, b_{pi}^k, m_{ij}^k). \tag{3.11}$$

For the purposes of planning, the expected utility must be predicted into the future by basing its evaluation on the predicted future belief, $\tilde{\rho}_i^{k+n,k} = E^{b_i^{k+n,k}}\left\{U_B + U_A + U_M\right\}$.

The belief utility U_B depends only on the quality of the belief about the world and not on the true world state itself. As the world belief is synchronized network-wide according to

	Passive	Active
Isolated	b^k_{xi}	b^k_{xi}, $b^{k+n,k}_{xi}$
Coordinated	$b^k_{x\alpha}$	$b^k_{x\alpha}$, $b^{k+n,k}_{xi}$
Cooperative	–	$b^k_{x\alpha}$, $b^{k+n,k}_{x\alpha}$

Table 3.1: Types of ASN systems classified by synchronization mode. The types are defined based on the environment beliefs they hold. Passive systems do not make plans and, therefore, do not cooperate.

Equation 3.7, so is the belief utility

$$\rho^k_{\alpha B} = \{\tilde{\rho}^{k\pm\delta}_{iB}\}_{i\in\alpha}. \tag{3.12}$$

This utility synchronization provides a mechanism for coupling platforms' actions. Through communication, the platforms approximate expected belief utility $E^{b^k_\alpha}\{U_B\}$ with partial belief utility $E^{b^k_i}\{U_B\}$. The approach is the same for beliefs about the current and future state of the system. The advantage is that, just like the actual information, the expected future information is additive. Partial utilities $\tilde{\rho}^k_{iB}$ are not transmitted because, in general, they cannot be added together [147].

Table 3.1 lists three different types of ASN systems based on their access to the environment belief according to Grocholsky [70, 71]. Passive systems maintain current belief only, while active ones plan ahead. Isolated systems maintain local beliefs only. Coordinated systems access current team beliefs, but the plans remain local. Cooperative systems have access to both current and future team beliefs. The expression for utility is different for the three active system types listed in Table 3.1.

Isolated. A platform makes decisions in isolation from the team, relying on local information only. The team utility at time t_{k+n} is approximated based only on local observations Z^k_i made up to the current time t_k and local observations expected to be made up to time t_{k+n}. The planned observations are based on planned local actions $A^{k+n,k}_i$ which are implicitly included here,

$$\begin{aligned}\tilde{\rho}^{k+n,k}_{iB} &= \tilde{J}_B(b^{k+n,k}_{xi} \mid A^{k+n,k}_i) & (3.13)\\ \tilde{\rho}^{k+n,k}_{iA} &= \tilde{J}_A(b^{k+n,k}_{xi}, b^{k+n,k}_{pi}, a^{k+n,k}_i \mid A^{k+n,k}_i). & (3.14)\end{aligned}$$

An isolated platform does not communicate so the utility of communication is not defined.

Coordinated. A platform makes decisions based on the team belief and local plans. The team utility at time t_{k+n} is approximated based on actual observations of the entire team Z_α^k up to the current (planning) time t_k and local observations Z_i^k planned to be made up to time t_{k+n}. The heterogeneous nature of platforms is reflected in the utilities of action and communication by specific to individual platforms.

$$\begin{aligned}
\tilde{\rho}_{iB}^{k+n,k} &= \tilde{J}_B(b_{xi}^{k+n,k} \mid A_\alpha^k, A_i^{k+n,k}) && (3.15)\\
\tilde{\rho}_{iA}^{k+n,k} &= \tilde{J}_A(b_{xi}^{k+n,k}, b_{pi}^{k+n,k}, a_i^{k+n,k} \mid A_i^{k+n,k}) && (3.16)\\
\tilde{\rho}_{iM}^{k+n,k} &= \tilde{J}_M(b_{xi}^{k+n,k}, b_{pi}^{k+n,k}, m_{ij}^{k+n,k} \mid A_i^{k+n,k}, M_{\alpha i}^k, M_{ij}^{k+n-1,k}) && (3.17)
\end{aligned}$$

Cooperative. A platform makes decisions based on the team belief and plans. The team utility at time t_{k+n} is approximated based on the observation of the entire team Z_α^k up to the current (planning) time t_k and planned to be made up to time t_{k+n}.

$$\begin{aligned}
\tilde{\rho}_{iB}^{k+n,k} &= \tilde{J}_B(b_{xi}^{k+n,k} \mid A_\alpha^{k+n,k}) && (3.18)\\
\tilde{\rho}_{iA}^{k+n,k} &= \tilde{J}_A(b_{xi}^{k+n,k}, b_{pi}^{k+n,k}, a_i^{k+n,k} \mid A_i^{k+n,k}) && (3.19)\\
\tilde{\rho}_{iM}^{k+n,k} &= \tilde{J}_M(b_{xi}^{k+n,k}, b_{pi}^{k+n,k}, m_{ij}^{k+n,k} \mid A_\alpha^{k+n,k}, M_{\alpha i}^k, M_{ij}^{k+n-1,k}) && (3.20)
\end{aligned}$$

3.3.4 Expected Value

The expected value of policies π_{iA} and π_{iM} is a sum of partial belief, action, and communication utilities

$$\tilde{V}_i^K(\pi_{iA}, \pi_{iM}) = \sum_{t=0}^{K} \tilde{\rho}_{iB}^{k+t,k} + \sum_{t=0}^{K} \tilde{\rho}_{iA}^{k+t,k} + \sum_{t=0}^{K} \tilde{\rho}_{iM}^{k+t,k}. \tag{3.21}$$

Only the partial belief utility $\tilde{\rho}_{iB}^{k+t,k}$ is synchronized across platforms. This formulation requires the future belief to be maintained at multiple points from the current time to a time horizon $\{b_{x\alpha}^{k+1,k}, b_{x\alpha}^{k+2,k}, ..., b_{x\alpha}^{k+n,k}\}$. It offers maximum flexibility and allows the decision maker to give preference to the plans which anticipate better belief quality at an earlier time. The downside is an increase in complexity and communication requirements. The evaluation is simplified if the belief value is calculated only at the time horizon and the action value remains integral

$$\tilde{V}_i^K(\pi_{iA}) \approx \tilde{\rho}_{iB}^{k+K,k} + \sum_{t=0}^{K} \tilde{\rho}_{iA}^{k+t,k} + \sum_{t=0}^{K} \tilde{\rho}_{iM}^{k+t,k}. \tag{3.22}$$

3.3.5 Policies

Following the decision rule approach, the communication and action policies are chosen at every decision point.

Communication policy. The communication policy has to choose between two available options: transmit the local belief or not. The decision is confined to the part of the local belief which has changed due to local observations or messages from neighbors. The communication policy is chosen to maximize a single-step expected value

$$\pi_{iM}^* = \max \tilde{V}_i^0(\pi_{iM}). \tag{3.23}$$

The communication utility are defined in Equations 3.17 and 3.20 depending on the system type. Grocholsky proposes to use the mutual information gain associated with the communication message for evaluating the utility of transmission [70].

The decision can be made by specifying a minimum threshold ρ_{min} for the utility of transmission, set for example to zero

$$\pi_{iM} : m_{ij}^k = \begin{cases} b_i^{k+n,k} & \text{if } \tilde{\rho}_{iM}^{k+n,k} > \rho_{min} \\ \varnothing & \text{otherwise.} \end{cases} \tag{3.24}$$

Another possibility is to specify a threshold for the belief quality on the consumer side [36]. Only when the estimated belief quality drops below the threshold, the information provider transmits an update. A simpler approach is to specify the maximum communication bandwidth allocated to each channel. All transmission temporarily stops when the allocated bandwidth is exceeded. Note that for all of these possible approaches, the communication policy is the same for all platforms, $\{\pi_{iM}\}_{i\in\alpha} = \pi_M$, even though the expected communication utility $\tilde{J}_M$ is platform-specific.

Action Policy. The action policy is chosen to maximize a generally multi-step expected value

$$\pi_{iA}^{*} = \max \tilde{V}_i^K(\pi_{iA}). \tag{3.25}$$

The design space of action policy selection is very large and multi-dimensional. Two aspects of the solution are discussed briefly here: 1) length of planning horizon and 2) continuity of the utility function. Chapter 5 focuses entirely on the decision making algorithm.

The 1-step planning horizon problem can be solved by simply enumerating the alternatives, evaluating the expected value, and choosing the best one. The N-step problem is more complicated because it requires dynamic programming [17], i.e., the outcome of the first step determines the options for the second one, and so on. The nature of utility has a strong influence on the required planning horizon. Domains with *terminal* rather than *integral* utility generally require longer planning horizon.[10] RoboCup is an example of a domain with predominantly terminal utility function, i.e. scoring a goal. A combination of terminal payoff with a high degree of noise in observations and actions makes this field particularly challenging.

If the utility function is discontinuous (actions are truly discrete and not discretized), there is no alternative to evaluating the expected value corresponding to all possible actions and selecting the best one. Destination-based exploration policy in the multi-platform case becomes essentially a target assignment problem [156, 211, 243].

If the utility function is assumed to be smooth, optimization can be done numerically or, in some cases, analytically. An example of solving the DIG problem with twice-differentiable performance measure using Newton's method is discussed in [70]. Even in the cases where the utility function is locally smooth, it is of course still possible to discretize the action space, evaluate all discretized options and pick the best one [20, 52].

3.3.6 Algorithm Illustration

To illustrate the algorithm, a single cycle of its operation is considered. The operation is divided into two stages: the execution stage, with communication and action policies

[10]Different research fields use different terms to describe the same concern. The terminology adopted here, integral vs. terminal payoff, is used in differential game theory [101]. In machine learning: the utility function is non-zero for a "large" number of state-action pairs resulting in a *sparse reward function* [213]. In the theory of negotiations: application domains are classified as worth-oriented vs. task-oriented [192]. In multi-agent systems: "a goal is neither black nor white, but rather has a degree of achievement associated with it" [124].

already determined, and planning, in which the policies are selected.

Execution stage. Figure 3.4 shows the execution stage of the algorithm. A boundary separates the platform from the environment (below) and from the rest of the team (above). The true states of the world and the platform, x^k and p_i^k, are unknown. Platform's belief b_i^k is at the center of the diagram. It contains exactly two types of beliefs: belief about the world b_{xi}^k and belief about the platform's own states b_{pi}^k. Interaction with the world happens in two ways: observations z_i^k cross the boundary inward and actions a_i^k cross the boundary outward.

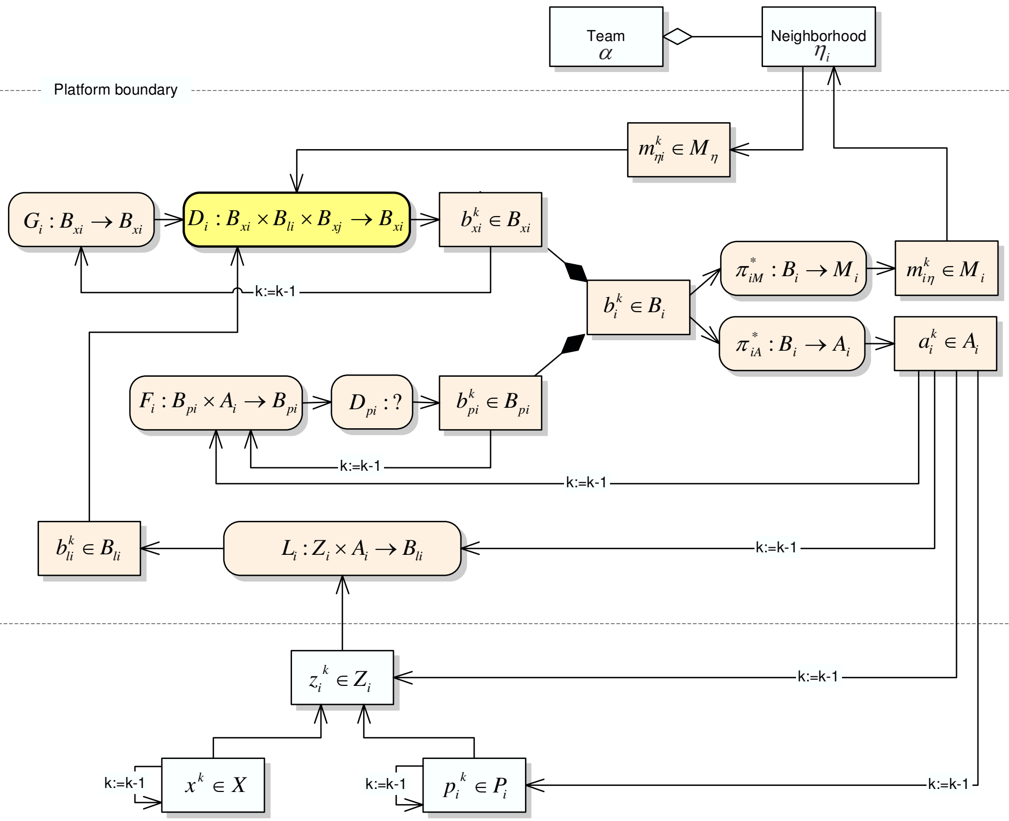


Figure 3.4: Execution stage of platform activity. The policies $\langle \pi_{iA}^*, \pi_{iM}^* \rangle$ are already selected. The asymmetric filled-diamond connector indicates containment, i.e. the belief is divided into beliefs about the world and platform states. Rectangles designate variables, rounded rectangles designate mappings.

The platform receives an observation z_i^k generated by an unknown function of the true system states and the previous action a_i^{k-1}. Based on the observation and the known previous action, the likelihood model L_i produces an observation likelihood b_{li}^k. At the same time step t_k, the platform receives a message $m^k_{\eta i}$ from a platform in the local neighborhood which contains its own world belief b^k_{xj}.

The local world belief needs to be updated. The previous belief b^{k-1}_{xi} is first predicted forward using the feature belief prediction function G'_i and then updated using the belief update function D_i. The updated belief b^{k+}_{xi} is based on the predicted belief b^k_{xi}, the latest observation likelihood b^k_{li}, and the communicated belief b^k_{xj}. The belief about the platform itself is predicted forward using the belief prediction function F'_i and possibly updated with local observations (outside of the ASN framework). The updated belief b^{k+}_{pi} is based on the predicted belief b^k_{pi}, the previous action a^{k-1}_i, and possibly on observations.

The current message $m^k_{i\eta}$ and action a^k_i are chosen based on the current belief b^k_i, using a pair of fixed policies $\langle \pi^*_{iM}, \pi^*_{iA} \rangle$. The message $m^k_{i\eta}$ is sent to one or more platforms in the neighborhood. The action a^k_i is executed locally. This completes the description of the execution stage. It remains to describe how the policies are chosen.

Planning stage. The planning process depends on the nature of the utility function, the time horizon, and synchronization mode. For concreteness, the procedure is described for the case of a discrete utility function, zero-step look-ahead ($n = 1$), and cooperative synchronization mode. Figures 3.5 and 3.6 illustrate the planning stage of platform activity. A boundary separates the platform from the rest of the team. The belief in the center of the planning process is the future expected belief $b^{k+1,k}_i$. Just like the current belief, it consists of two parts: the expected world belief $b^{k+1,k}_{xi}$ and the expected belief about state of the platform $b^{k+1,k}_{pi}$. As the belief under consideration is in the future, the current true state of the world does not directly influence the planning process, i.e. the boundary between the world and the platform is not crossed.

In cooperative mode, the solution process is iterative. All stages of negotiation occur before proceeding to the next execution step t_{k+1}. Figure 3.5 illustrates a single stage l of the negotiated cooperative solution. The current and future beliefs, b^k_i and $b^{k+1,k}_i$, are fixed and the platform has to choose its action policy π^l_{iA}. For each feasible action $a^{k+1,k}_i$, two partial utilities are evaluated, $\tilde{J}_{iB}$ and $\tilde{J}_{iA}$. The solution procedure chooses the action policy π^l_{iA} which maximizes the expected value over a single step. The chosen action is compared with the action from the previous stage and an appropriate test for convergence is applied. If the policy selection process has converged, the system enters execution stage in Figure 3.4.

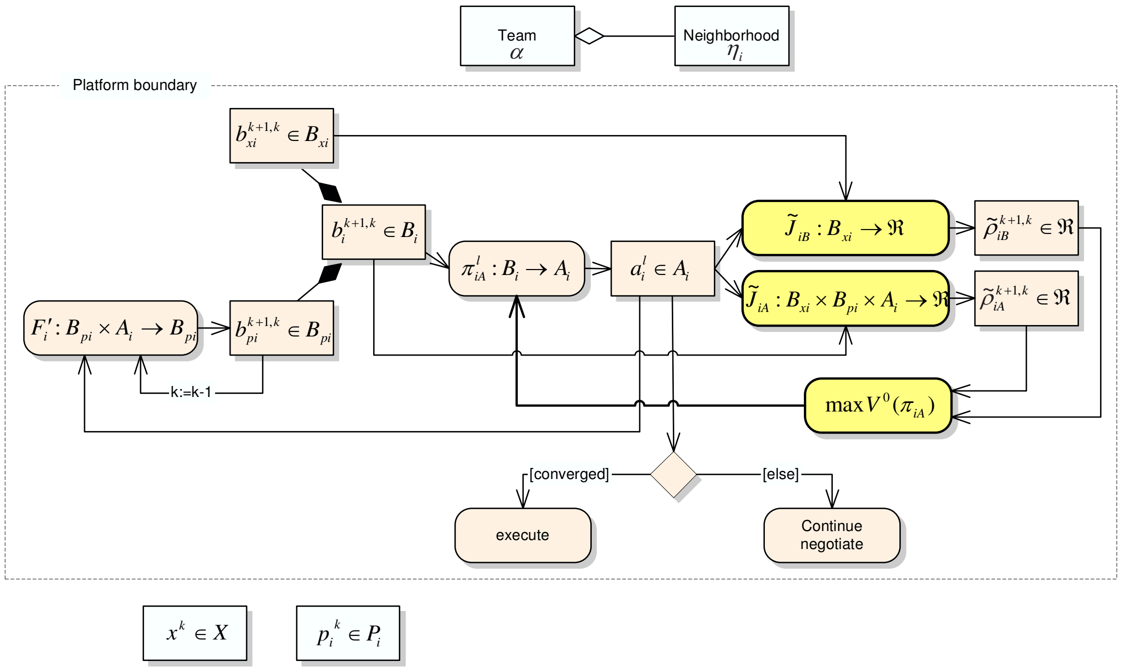

Team
α
Neighborhood
η_i
Platform boundary
$b_{xi}^{k+1,k} \in B_{xi}$
$b_i^{k+1,k} \in B_i$
$\pi_{iA}^l : B_i \to A_i$
$a_i^l \in A_i$
$\tilde{J}_{iB} : B_{xi} \to \Re$
$\tilde{J}_{iA} : B_{xi} \times B_{pi} \times A_i \to \Re$
$\tilde{\rho}_{iB}^{k+1,k} \in \Re$
$\tilde{\rho}_{iA}^{k+1,k} \in \Re$
$F_i' : B_{pi} \times A_i \to B_{pi}$
$b_{pi}^{k+1,k} \in B_{pi}$
k:=k-1
$\max V^0(\pi_{iA})$
[converged]
[else]
execute
Continue negotiate
$x^k \in X$
$p_i^k \in P_i$

Figure 3.5: Planning stage of platform activity for zero-look ahead case. A local action policy π_{iA}^l is selected to maximize the expected value.

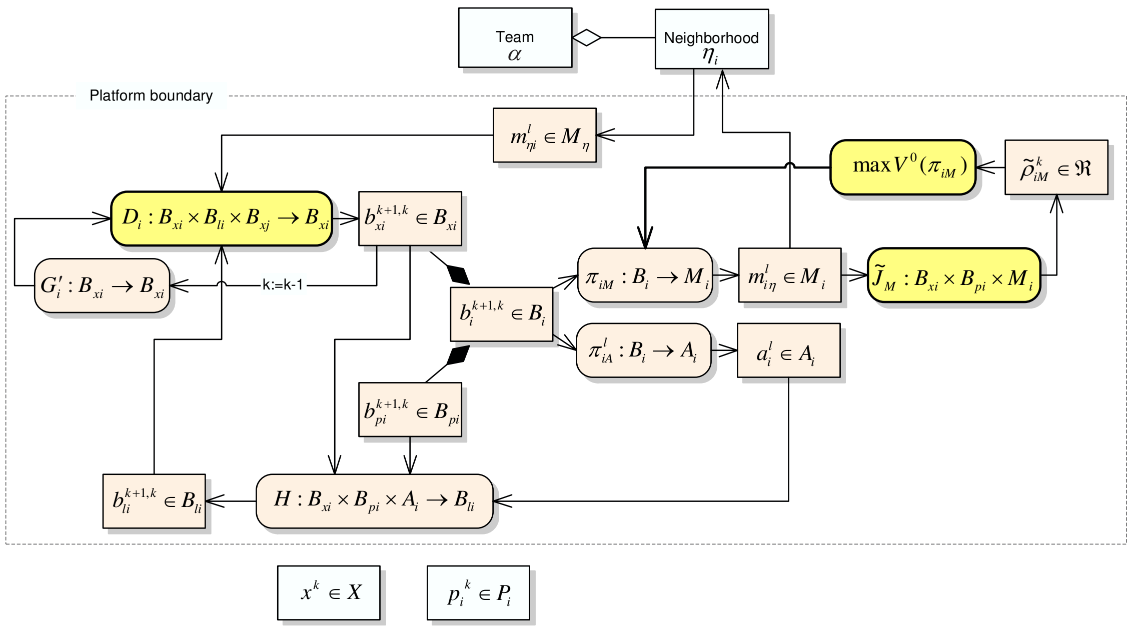

Team
α
Neighborhood
η_i
Platform boundary
$m_{\eta i}^l \in M_\eta$
$\max V^0(\pi_{iM})$
$\tilde{\rho}_{iM}^k \in \Re$
$D_i : B_{xi} \times B_{li} \times B_{xj} \to B_{xi}$
$b_{xi}^{k+1,k} \in B_{xi}$
$G_i' : B_{xi} \to B_{xi}$
k:=k-1
$b_i^{k+1,k} \in B_i$
$\pi_{iM} : B_i \to M_i$
$m_{i\eta}^l \in M_i$
$\tilde{J}_M : B_{xi} \times B_{pi} \times M_i$
$\pi_{iA}^l : B_i \to A_i$
$a_i^l \in A_i$
$b_{pi}^{k+1,k} \in B_{pi}$
$b_{li}^{k+1,k} \in B_{li}$
$H : B_{xi} \times B_{pi} \times A_i \to B_{li}$
$x^k \in X$
$p_i^k \in P_i$

Figure 3.6: Planning stage of platform activity for zero-look ahead case. A local communication policy π_{iM}^l is selected as part of the negotiation process.

If convergence is not achieved, the observation plan corresponding to the latest action choice is communicated back to the team. Figure 3.6 shows the information flow to and from the team. The world belief update function D_i is the same as in Figure 3.4. Local future likelihood $b_{li}^{k+1,k}$ is based on the action a_i^l chosen in the current negotiation stage. Messages from the team $m_{\eta i}^l$ arrive in the form of external future world beliefs $b_{xj}^{k+1,k}$. The communication policy weighs the partial utility $\tilde{J}_{iM}$ of sending a message $m_{i\eta}^l$. After completion, the system enters the next stage of negotiation in Figure 3.5.

In summary, the main differences between the ASN algorithmic architecture and the general DIG problem statement are the following

1. Inter-platform communication is strictly on a peer-to-peer basis, m_{ij}^k not m_{i*}^k (following the principle of decentralization).

2. Inter-platform communication is limited to the platform's local neighborhood η_i, $\{m_{ij}^k\}_{j\in\eta_i}$ with $\eta_i \subset \alpha$ (principle of local interactions).

3. Only beliefs on the world state are transmitted between platforms: including observation likelihoods, current beliefs, and future beliefs. In information fusion, this means that raw observations defined on $\mathcal{Z}_i$ are not transmitted. In decision making, it means that actions defined on $\mathcal{A}_i$ and platform states on $\mathcal{P}_i$ are not transmitted. Thus the difficult problem of utility fusion is circumvented by fusing planned observation likelihoods and not the utilities (principle of modularity).

3.4 Conceptual Architecture

The previous section described the ASN approach in its entirety but, despite best efforts, the description did not make it easy to grasp the system as a whole. It serves as an example, unintentionally, of how the complexity of a large framework may appear unmanageable. The problem can be traced, at least in part, to the presentation style which describes a large system as a flat non-hierarchical structure.[11] This section and the rest of the chapter are concerned with partitioning the ASN framework into components in order to make it easier

[11]Possibly approaching the limits of human short-term memory which can reportedly hold seven, give or take two, items at a time [153].

to understand, implement, and use. The conceptual architecture described here begins the process by directing attention to an appropriate decomposition of the system.

Section 3.4.1 considers module *granularity* appropriate for this system — the size of the atomic elements. The approach taken in this work is to select *functional roles* as the fundamental structural level which are identified in Section 3.4.2. Interactions between the component roles are constrained by the meta-architectural principles.

The roles are grouped into three largely independent functions: information fusion, decision making, and system configuration. The reason, again, is primarily to allow independent analysis, design, and description. Each functional group is discussed separately in Sections 3.4.3–3.4.5. In these Sections, the algorithmic and functional structures are matched by mapping the algorithms from Section 3.3 to the appropriate component roles.

3.4.1 Partitioning Criteria

Having made the decision to design a modular system, the question arises as to what the best partitioning criteria are. The classic advise, attributed to Parnas, is to choose components which are highly cohesive internally (serve a single purpose) and loosely coupled (have few external connections) [171]. Practically, the process of modularization reduces to the questions of what is contained in a module and what type, kind, and quantity of connectors are employed. The size of the modules, or component *granularity*, is an important factor for reasons of design, implementation, deployment, marketing, legal, etc.[12]

One natural level of granularity in the DIG problem is the scale of the sensing platforms themselves. The physical collocation of platform's hardware and software imposes a certain level of cohesiveness. At the same time, the physical separation between the platforms and the outside world creates a natural boundary, or a *membrane*, with a limited and defined set of inputs and outputs [54]. This boundary naturally decouples the platforms from each other.

While platforms seem to satisfy the requirements for module definitions, the ASN architecture opts for finer-grained components for two reasons: algorithmic and composition. The reference model revisited in Section 3.3 reveals considerable structure in the DIG problem at the sub-platform level of granularity. From the point of view of system composition,

[12]A variety of factors which may be considered in choosing the level of granularity are discussed in [224].

the smaller the size of the modules, the higher the total number of possible permutations, and the more freedom a designer has in recombining the modules. Thus the modules of sub-platform granularity are preferred, implying that the platform architectures themselves should be modular.

On the other hand, very fine-grained modules have disadvantages as well. Intuitively, the amount of development effort needed to integrate a set of modules grows with the number of modules. Other issues related to software development process suggest that the module size should be larger than a class [35, 149].

The optimal size of atomic elements appears to lie somewhere between the scale of compiled objects and platforms. The approach taken in this work is to select *functional roles* as the fundamental building blocks. The actual deployable components are constructed by combining these building blocks.

3.4.2 Functional Decomposition

Partitioning the system into modules begins by identifying possible *component roles* in relation to the data types exchanged within the system. The data is first grouped into three categories:

1. Current world belief, including beliefs $b^k_{xi} \in \mathcal{B}_x$ and observation likelihoods $b^k_{li} \in \mathcal{B}_x$. Both are distributions on state space of the environment $\mathcal{X}$.
2. Future (planned) world belief, including beliefs $b^{k+n,k}_{xi} \in \mathcal{B}_x$ and planned observation likelihoods $b^{k+n,k}_{li} \in \mathcal{B}_x$. Both are also distributions on $\mathcal{X}$.
3. Action plans to time horizon t_{k+n}: $A^{k+n,k}_i = \{a^{k+1,k}_i, a^{k+2,k}_i, ..., a^{k+n,k}_i\}$, where $a^k_i \in \mathcal{A}_i$.

For each of the three data categories, four component roles are considered: source (producer), sink (consumer), fuser (aggregator), and distributor (transmitter). Table 3.2 shows the matrix of possible combinations between the data types and component roles. The check-marked entries of the table correspond to the ten component roles recognized within the ASN architecture. The table highlights the difference in how the world belief and action plans are treated. Network-wide synchronization of the world beliefs requires all four

Data type		Component role			
		Source	*Fuser*	*Distributor*	*Sink*
Current world belief	("Belief")	✓	✓	✓	✓
Future planned world belief	("Plan")	✓	✓	✓	✓
Future planned actions	("Action")	✓			✓

Table 3.2: Component roles with respect to data types. Abbreviated names for each data type are shown in parentheses. Ten check marks correspond to all component roles possible within the ASN architecture. Action plans are kept local so their distribution is not needed. Fusion of action plans is typical in behavior-based robotics but is not implemented in ASN.

component roles. The action plans are local to individual source-sink pairs and, therefore, do not need to be distributed and fused.

When used as part of role names, the data types names will be abbreviated. "Belief" will stand for "current belief about the world", "plan" for "future planned belief about the world", and "action" for "future planned action". To make the reference clear, component roles will be set in italics, for example *belief source*. Figure 3.7 shows the ten roles implemented in ASN. All ten are derived from (are *specializations* of) an abstract *ASN Component* role.

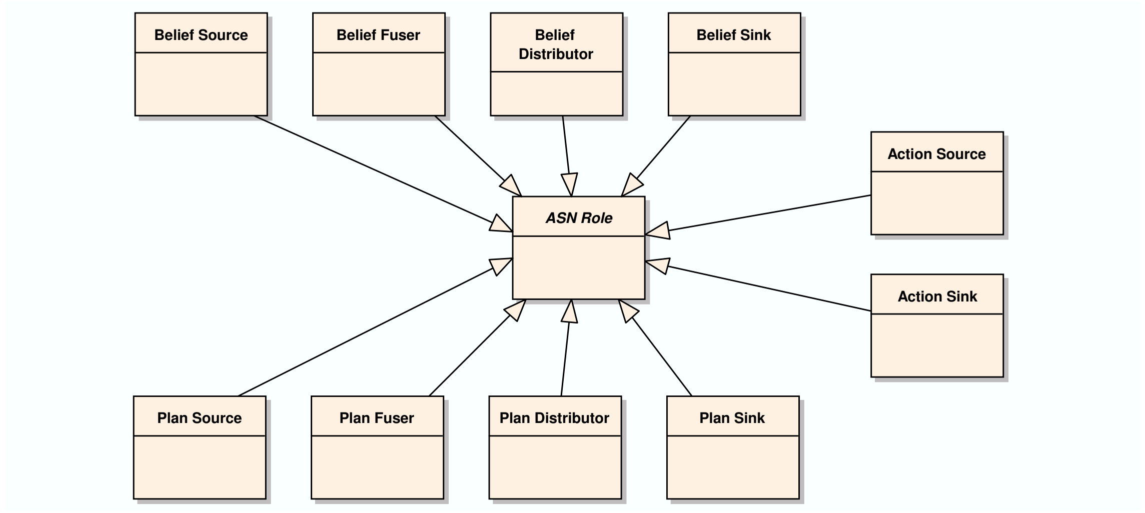


Figure 3.7: Component roles in ASN. Each role corresponds to a check mark in Table 3.2.

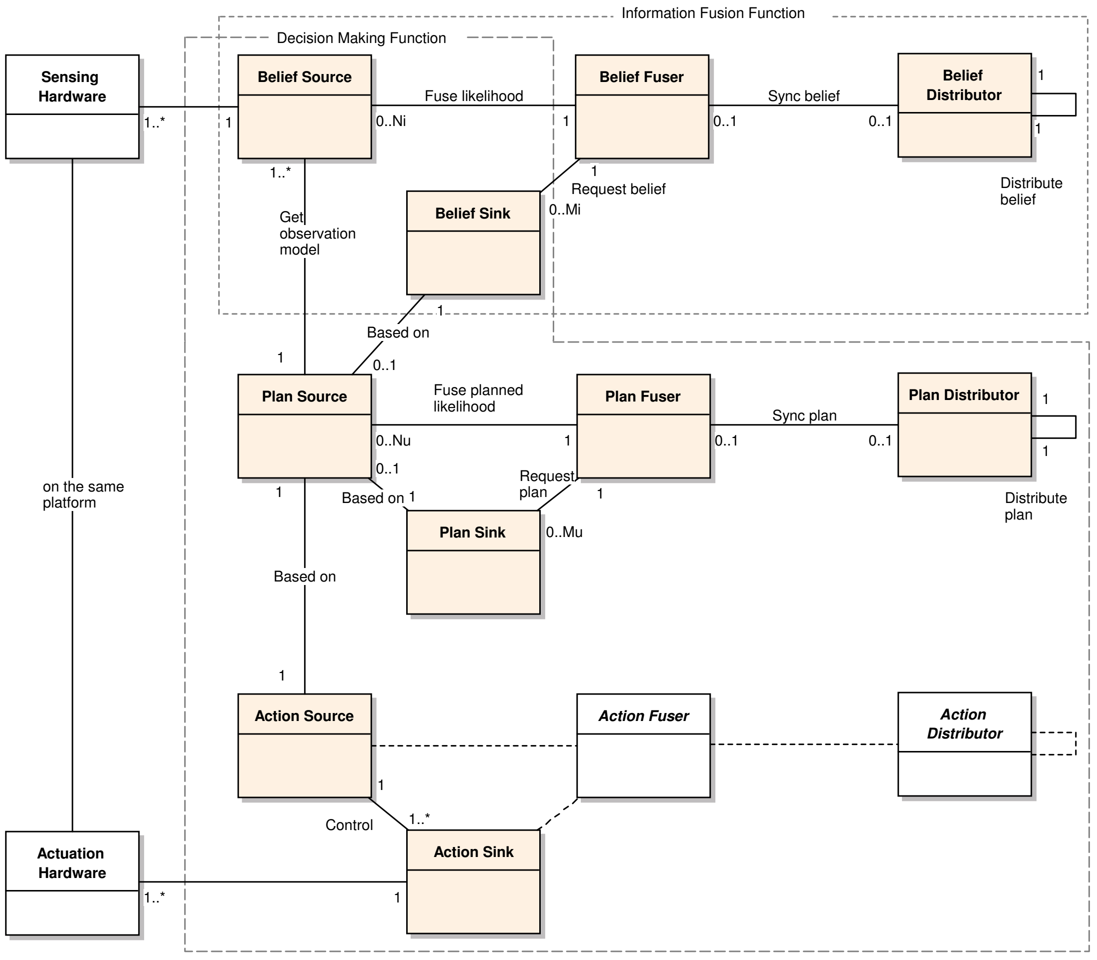


Figure 3.8: The relationship between component roles in ASN. The top four component roles related to forming beliefs are grouped into the task of information fusion. The eight roles related to action selection and execution are grouped into the task of decision making. All component roles participate in the task system configuration. *Action fuser* and *distributor* roles are not used in ASN but their place in the diagram is indicated for completeness. The sensing and actuation hardware is shown to indicate the physical coupling between the actions and observations.

Figure 3.8 shows the same ten roles and the relationships between the roles. Interaction between the roles is shown using a UML *Class diagram*.[13] It is convenient to separate the activities in which a platform is engaged into three task groups:[14] information fusion,

[13]`UML` A *Class diagram* captures the logical structure of the system – the classes and things that make up the model. It is a static model, describing what exists and what attributes and behavior it has, rather than how something is done. [223] Roles in association can have multiplicities specified.

[14]The term *layer* is intentionally not used in this context in order not to imply a sense of special structure between the tasks (e.g. similar to the Open Systems Interconnection reference model [57]).

decision making, and system configuration. Membership of component roles in the first two groups is shown as dashed boundaries. All component roles participate in the task of system configuration, so this grouping is not shown.

The three functional groups are identified because their operation is largely independent from each other. By considering the functional groups separately, the design, description, and implementation of the overall architecture is simplified.

Connectivity is the second distinguishing feature: the three groups have different reasons for interaction which may lead to different communication patterns between platforms and components. For example, several platforms observing the same part of the environment are likely to be engaged on the information fusion level. At the same time, the decision makers on the same platforms, negotiating future plans, may be communicating with a different set of team members. Meanwhile, the reconfiguration algorithm may be discussing a change in topology with yet another set of components.

In addition to the three task groups described above, it is easy to envision addition of functional groups operating at higher levels of abstraction. On the information fusion side, additional groups may correspond to the higher levels of the JDL fusion model [130], for example *situation awareness.* On the decision making side, a "task management" group is a good candidate intended for task allocation, scheduling, and cooperative execution [126]. These activities go beyond the scope of the ASN framework and the DIG problem and would lead towards more general-purpose decentralized systems.

In the rest of the section, each of the three functional groups is considered separately. The possible interactions between the component roles are further constrained by applying the ASN design principles. At the same time, the independently derived algorithmic and functional structures are matched by mapping the algorithms from Section 3.3 to the component roles from this section.

3.4.3 Information Fusion Task

The information fusion functional group includes all tasks related to the maintenance of the current belief: perception, update, storage, synchronization, and delivery to consumers. Four component roles participate in this function: *belief source, fuser, distributor* and *sink.*

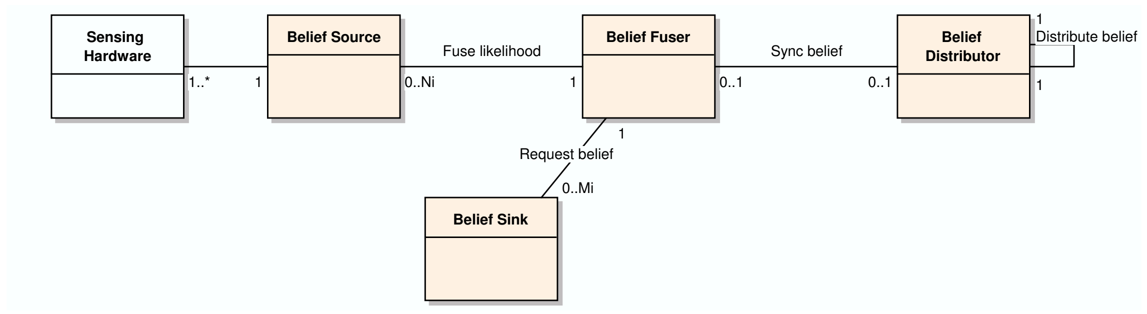


Figure 3.9: Class diagram for the task of information fusion. This is a subset of Figure 3.8.

Figure 3.9 shows a segment of Figure 3.8 focusing on the four component roles participating in information fusion. The function of extracting information from raw observations and fusing them into a local belief b^k_{xi} requires two component roles: *belief source* and *belief fuser*. Following the principle of modularity, all observations are processed locally at each *belief source*. This modular approach allows heterogeneous sensors to be incorporated easily: each *belief source* has access to its own observation model, but the *belief fuser* does not need to know it. Sensor-specific observations are translated into the form understood by everyone — a likelihood defined on the state of the environment. In this form, communicated information can be interpreted and fused without any knowledge of its source.

A strict many-to-one relationship exists between the *sources* and the *fusers* to ensure that the information from *belief sources* is not counted twice. As with any non peer-to-peer relationship, there is a danger that, with the network growth, this relationship becomes the bottleneck of system performance. To ensure that this does not happen, the number of *belief fusers* must grow with the number of *belief sources*. The ultimate decentralization of information fusion is achieved when a dedicated *belief fuser* exists for each *belief source*. This may not always be practical, but the architecture and the algorithms must be able to accommodate this option. In practice, a *fuser* may limit the maximum number of *sources*, based on its communication and computational capacity.

The function of delivering the current belief b^k_{xi} to interested consumers involves two component roles: *belief sinks* request information from the *fusers*. A *fuser* can provide information to several *sinks* implementing a one-to-many relationship. To ensure proper scaling, the number of *fusers* must grow with the number of *sinks*. The maximum number of *sinks* supported by a single *fuser* may be limited, based on the hardware limitations.

The function of belief synchronization requires two component roles: *belief fuser* and *belief distributor*. Connections between *distributors* form the backbone of the ASN framework. The information exchanged between *distributors* is in the form of their local beliefs b^k_{xi} and b^k_{xj}. From the principle of decentralization, the links between *distributors* are strictly of peer-to-peer type. From the principle of local interactions, the number of links each *distributor* maintains is limited and the maximum number is constant. The following example illustrates the ASN approach to information fusion.

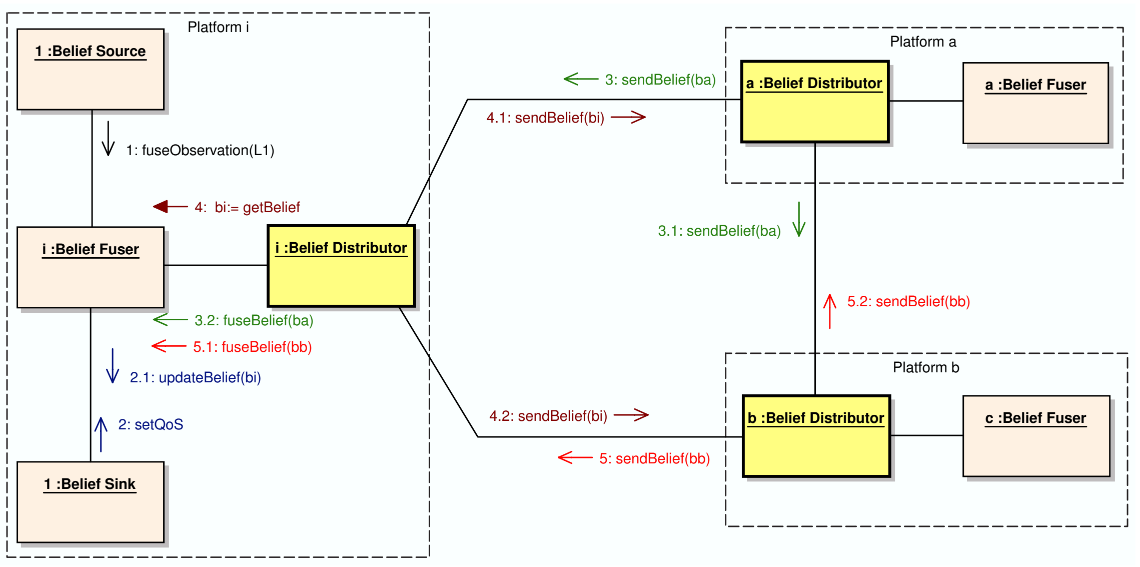


Figure 3.10: Communication diagram illustrating the task of information fusion. Three communicating platforms are shown. *Belief distributors* (highlighted) form the backbone of the system synchronizing beliefs of *fusers* on different platforms.

Example 3.1

Consider an example with three platforms. Message exchange between platforms is shown in Figure 3.10 as a UML Communication diagram.[15] *Each software object in the diagram plays one of the information fusion roles in Figure 3.9. The objects are grouped by platform boundaries. The description follows the order of messages starting with message group 1.*

Platform i contains a belief source *which sends a likelihood to a local* belief fuser *(mes-*

[15]UML A *Communication diagram* shows the interactions between elements at run-time. The inter-object relationships are visualized by showing the messages passed between the objects. Communication messages should be ordered to reflect the sequencing of the diagram. The numbering scheme should reflect the nesting of each event. A sample sequencing scheme could be 1, 2, 2.1, 2.2, 3. This would indicate events 2.1 and 2.2 occur within an operation initiated by event 2. [223] In this notation event 2 is equivalent to event 2.0.

sage 1). A belief sink *request information from the* belief fuser *(2) and starts receiving belief updates (2.1). The three* belief distributors *form an inter-platform network by establishing peer-to-peer links. The* belief distributor *a sends belief updates to* belief distributors *i and b (3, 3.1). On receipt,* belief distributor *i forwards it to its local* belief fuser *for fusion (3.2). To send its own belief update,* belief distributor *i first obtains current belief from the local* belief fuser *(4) and sends it to* belief distributors *a and b (4.1, 4.2).*

Note that in this example the distributors *formed a cyclic network. This detail is an important consideration for some belief update methods and will be further discussed in Chapter 4.*

To summarize, when expressed in terms of component roles, the scalability of information fusion in ASN stems from three design choices:

1. *Belief fusers* fuse only local observations and remote observations are not communicated. As more *belief sources* are added, the number of observations to be processed does not increase.

2. *Belief distributors* communicate with a limited number of other *distributors.* As the network grows in size, the number of communication links maintained by each *distributor* does not increase.

3. *Belief distributors* receive belief updates from neighboring *distributors.* They are fused with local beliefs and the *combined* belief is sent to the neighbors. Regardless of the number of incoming messages, each *distributor* transmits a single outgoing message which contains all new information. Consequently, as the sensor network grows in size, the amount of information communicated by a *distributor* remains constant.[16]

The last design constraint makes it necessary for every platform to contain a *belief fuser* even if no local observations are made. Without the ability to fuse incoming beliefs, a distributor is simply a router of beliefs. This situation is reminiscent of the approach taken in the Directed Diffusion algorithm [100] where *in-network* aggregation is absent or minimal and the platforms route observations to interested *belief sinks.* The following example illustrates a system without in-network aggregation. Note that while this example shows a system *not*

[16]In the field of μSN, this property is known as *in-network processing* [128].

typical of ASN it does demonstrate the use of component roles to describe a very different approach to solving the DIG problem.

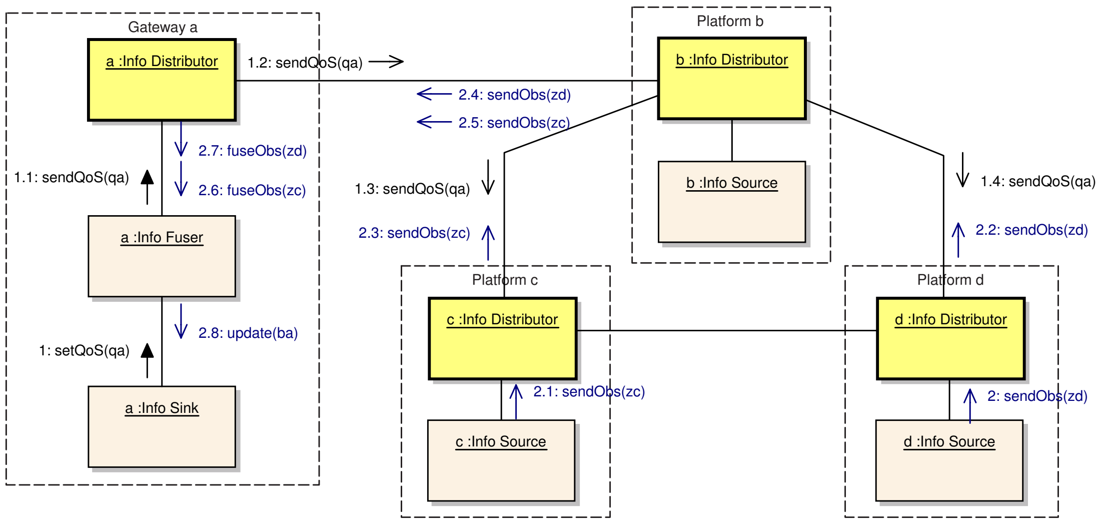


Figure 3.11: Information distribution using Direction Diffusion algorithm. *Belief distributors* (highlighted) route observations from *sources* to *sinks*. Information fusion is performed only at the gateway platform.

Example 3.2

The network consists of three sensing and one gateway platforms. Network topology and the messages exchanged are shown in Figure 3.11. A gateway platform is shown on the left. It contains a belief sink *which registers its interests with a local* belief fuser *(message 1). From here the interests are propagated through the network until reaching every platform (1.1–1.4). Some time later, the* belief sources *c and d observe an event which matches the interests of the gateway node. They transmit these observations nearly simultaneously through the network back to the gateway (2–2.5). Because there are no* belief fusers *in the network, the observations are not combined and the belief is not stored within the network. Both observations arrive at the gateway* belief fuser *(2.6, 2.7). Here they are combined into a belief b_a and forwarded to the* belief sink *(2.8).*

This example is intended as a simple illustration and the reality of μSN is more complex. For example, multi-path routing, maintained by most routing algorithms for robustness, would lead to multiple copies of the observations arriving at the gateway. Some algorithms attempt to discard multiple observations, which does cut down the communication traffic but does *not constitute fusion.*

The approach described in this section leads to important operational properties from the point of view of an individual platform. If the communication link between two *belief distributors* is suspended, the *belief fusers* on both platforms simply accumulate locally obtained information. When the communication link is reopened, the total locally accumulated belief is communicated in a single message. This allows operation with intermittent communication and provides a degree of flexibility even when the links are reliable. If a platform is disconnected from the network due to signal loss or jamming, it can be re-introduced later and information can be synchronized in a single step. Burst transmission of accumulated data can be employed on purpose in order to reduce communication bandwidth requirements.

3.4.4 Decision Making Task

The decision making functional group includes all tasks related to generating, evaluating, selecting, and synchronizing action and observation plans. A total of eight component roles participate in this function: two related to current beliefs, *belief source* and *sink*, all four associated with future beliefs, and two associated with actions, *action source* and *sink*.

Figure 3.12 is a segment of Figure 3.8 showing interactions between the eight component roles participating in decision making. The function of controlling actuators requires two component roles: *action source* and *action sink*. A single *action source* can issue commands to one or several *action sinks*.[17] To ensure proper scaling, the number of *action sources* must grow with the number of *action sinks*. The ultimate in decentralization of decision making is achieved when a dedicated *action source* exists for each *action sink*. This may be impractical in reality, but the architecture and the decision making algorithms must be able to accommodate this option.

The function of evaluating local utility associated with action plans requires five component roles: *belief*, *plan*, and *action sources*, as well as *belief* and *plan sinks*. Observations are made by *belief sources*. The proposed control actions affect observations of one or several

[17]The role of an *"action fuser"* is effectively avoided by not allowing many-to-one relationships between *action sources* and *sinks*. See question mark in Table 3.2.

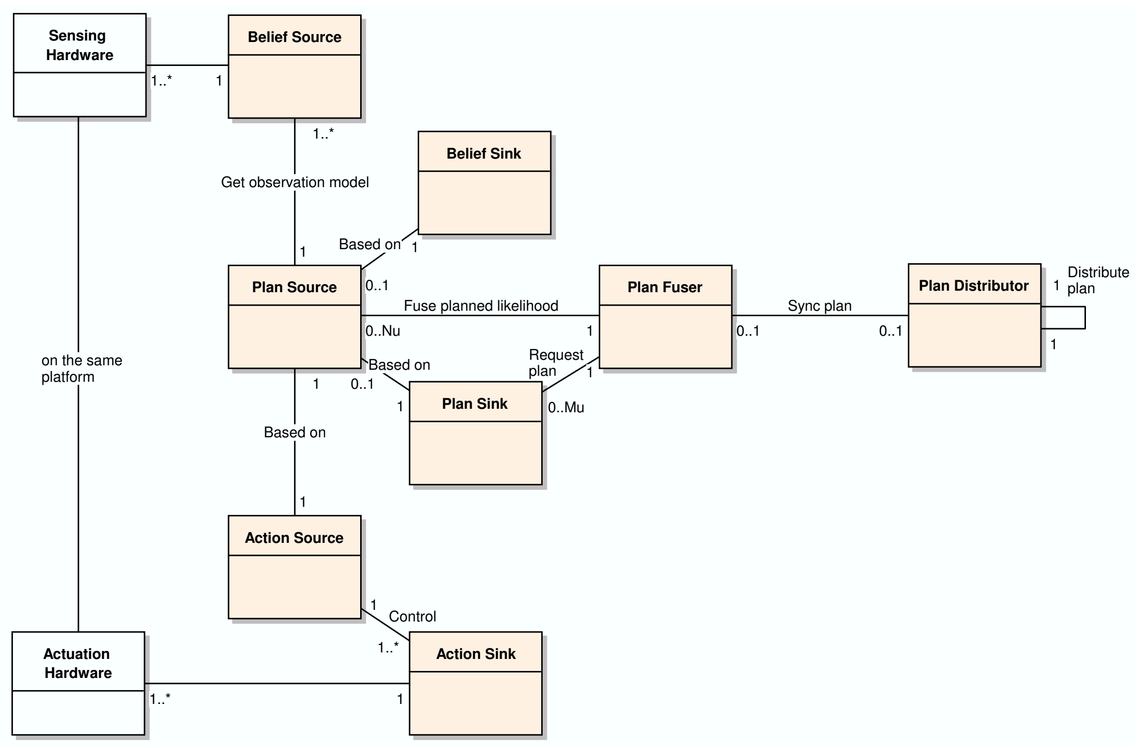


Figure 3.12: Class diagram for the task of decision making. This is a subset of Figure 3.8.

belief sources. This relationship is typically associated with the physical actuators and sensors being on the same platform. Feasibility of alternative action plans depends on the current state of the world provided by the *belief source*. Different observations lead to different expected future beliefs about the environment. The utility of observations in a team scenario depends on plans of other platforms. The combined team belief about the future state of the world is provided by the *plan sink*. In summary, *action sources* select an action plan based on the states of the *action sinks*, the properties of the *belief sources*, the relevant states of the environment maintained by the *belief sink*, and the anticipated future states of the environment maintained by the *plan sink*.

Following the principle of modularity, the utility of action plans is evaluated locally by *plan sources*. Different platforms have different sensors and actuators. A modular approach allows heterogeneous active platforms to be incorporated easily: each *plan source* knows the properties of its own hardware, but the *plan fuser* does not need to. As a result, platforms cooperate without knowing specifics of each other's capabilities. Platform-specific action and observation plans are translated into the form understood by everyone — a probability distribution on the state of the environment. In this form, communicated information can be interpreted and fused without any knowledge of its source. As with current beliefs, the relationship between *plan sources* and *plan fusers* is many-to-one and the number of *fusers* must keep up with the number of *sources* for scalability.

Delivery and distribution of future beliefs parallels that of current beliefs. The function of delivering future beliefs to interested consumers involves two component roles: *plan sinks* requesting information and *plan fusers* supplying it. As with current beliefs, the relationship between *plan fusers* and *plan sinks* is one-to-many and the number of *fusers* must keep up with the number of *sinks* for scalability. Plan synchronization is accomplished by the collaboration between *plan fusers* and *plan distributors*. Locality in information access leads to sparse network topology and peer-to-peer connections between the *distributors*.

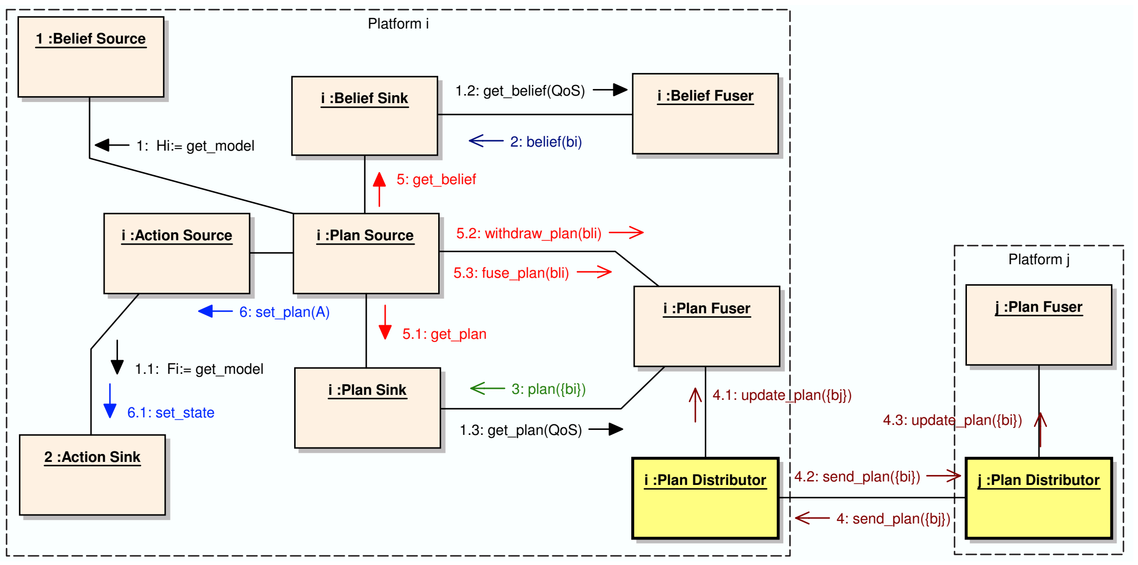


Figure 3.13: Communication diagram illustrating the task of decision making. *Plan distributors* (highlighted) synchronize observation plans between *plan fusers* on different platforms. The system makeup and topology on platform *j* may be different from platform i. The rest of component roles on platform *j* are not shown for lack of space.

Example 3.3 ———

Consider an example with two active platforms. Messages exchanged by modules within and between the platforms are shown in Figure 3.13. The plan source *on platform i requests the actuation and observation models from the local* belief source *and* action sink *(messages 1 and 1.1).* Belief *and* plan sources *request the current and planned beliefs from the corre-*

sponding fusers *(1.2, 1.3). In response to the request, the* fusers *start sending asynchronous updates to the* sources *(2, 3). The future belief updates between the* plan distributors *and* plan fusers *follow the pattern already seen in Example 3.1: each update received by the* plan distributors *(4) is fused by the* plan fusers *(4.1). At decision time, the* plan source *i obtains the current and future beliefs from the* sources *(5, 5.1) and produces the action and observation plan based on all locally available information. If the new plan is different from the previous one, the old plan is withdrawn (5.2) and the new local observation plan is submitted to the* plan fuser *(5.3). If the negotiation has converged, the platform enters execution stage by sending commands to the* action sink *(6, 6.1).*

To summarize, cooperative decision making in ASN is based on synchronizing observation plans. The algorithm and structure of this aspect of decision making closely parallels that of belief synchronization with all ensuing scalability properties.

3.4.5 System Configuration Task

System configuration is responsible for initialization and maintenance of component connections in a distributed system. All component roles participate in system configuration, either at start-time, or run-time, or both.

Typical stages of the network configuration process are discovery, selection, addressing, monitoring, and disconnection. During the discovery phase, a new component is looking for required services. Once a service is found, its availability and performance is monitored. If the service is lost or its performance is unacceptable the component returns to the service discovery phase. Common reasons for reconfiguration are addition, exit and especially failure of components; motion of platforms and environmental features; changes in operating conditions, etc.

Because all inter-platform communication in ASN is addressed (not broadcast), finding physical addresses of other components is an important part of the configuration process. The directory services are typically organized into two categories, similar to White and Yellow Pages.[18] A naming service converts a service name into a physical address. A trader

[18]CORBA middleware refers to these as *Naming* and *Trading* services respectively [72, 74].

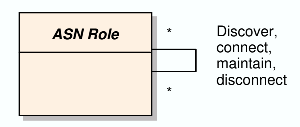


Figure 3.14: Class diagram for the task of system configuration.

service converts a service description into a physical address. Following the principle of decentralization, no centralized naming or trading services are used in ASN. In Figure 3.14 this is indicated by the absence of any external or centralized task managers, system configuration servers, etc. Components playing a particular role discover, connect, and disconnect from each other without any help from the infrastructure. Unlike with other associations, multiplicity of this relation is unspecified.

3.5 Logical Architecture

The conceptual architecture presented in the previous section began the process of partitioning the system by defining component roles. This section describes the logical architecture — a detailed architecture specification. Section 3.5.1 introduces a set of six components used in the examples and experiments. The components are considered standard for ASN are are called *canonical.* Each component type plays one or several component roles. The interaction between the components is captured in the services which they provide to each other. Interfaces which formalize the services are defined in Section 3.5.2.

Component specifications make the architecture tangible, but based on the practical experience with ASN, an architecture defined in terms of services and interfaces is more flexible. It offers a broader range of implementable systems compared to an architecture defined in terms of concrete component types. For this reason, the component listing in Section 3.5.1 should be considered *descriptive*, whereas the listing of interfaces in Section 3.5.2 *prescriptive.*[19] In other words, the interfaces become the definition of the architecture. In Section 3.6, the canonical components are formally *specified* in terms of the interfaces they provide and require.

[19]This view echoes Jonkers, who promotes *interface-centric* over *component-centric* architectural descriptions [104].

Component	Belief			Plan			Action	
Type	*Source*	*Fuse/Dist.*	*Sink*	*Source*	*Fuse/Dist.*	*Sink*	*Source*	*Sink*
SENSOR	✓							
NODE		✓			✓			
ACTUATOR								✓
PLANNER			✓	✓		✓	✓	
UI	✓		✓			✓	✓	
FRAME								

Table 3.3: Canonical components and the roles they play. Multiple check marks in the same row indicates that some inter-role relationships are internalized within a component. FRAME does not participate in information fusion or decision making but is required for localization and other platform-specific tasks.

The process of decomposition of the system is now complete and attention is turned to composition. Section 3.5.3 combines the components and the interfaces to show how the use cases of the DIG problem are realized in ASN. Section 3.5.4 returns to the question of the degree of decentralization in information fusion and decision making. This important issue was already raised in Sections 3.4.3 and 3.4.4. It is revisited because the idea is easier to illustrate using the definitions of canonical components.

3.5.1 Canonical Component Types

Ten component roles were identified in Section 3.4.2. It is possible, in principle, to build a system with ten component types using one-to-one mapping between roles and components. Some roles, however, are tightly coupled and combining several into a single component is more efficient. Six component types form a representative list: node, sensor, actuator, planner, user interface (UI), and frame. These components will be called *canonical.* Component names are set in small caps for clarity, for example SENSOR.

Table 3.3 shows the mapping of canonical components to the component roles. Multiple check marks within a row indicate that some inter-role relationships are internalized within a single component. For example, the PLANNER combines four tightly coupled roles related to generating and evaluating action plans. The same role-to-type mapping is presented graphically in Figure 3.15 using the *generalization* relationship.

A SENSOR plays the role of *belief source.* It represents one or more physical sensing devices. It performs the task of perception by processing raw measurements obtained from the

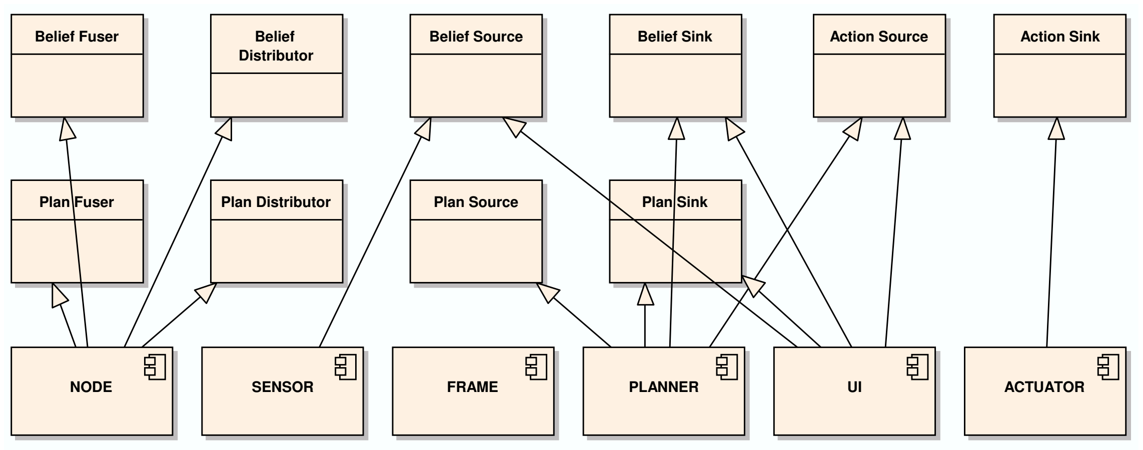


Figure 3.15: Mapping of component types to roles. Canonical components are shown using the UML *component* symbol.

physical sensors.

A NODE plays the roles of *belief fuser* and *distributor.* It is responsible for updating, storing, synchronizing, and distributing the current belief about the world. In the task of decision making it also plays the roles *plan fuser* and *distributor.* It updates, stores, synchronizes, and distributes the expected future beliefs about the environment.

An ACTUATOR plays the role of an *action sink.* It represents one or more physical active devices. It ensures that the action plans are executed.

A PLANNER is responsible for generating action plans in cooperation with other controllers. In this function it plays the roles of a *belief* and *plan sink*, as well as *action* and *plan source.*

A UI manages interactions between a human operator and the network. It plays the roles of *belief source* and *sink* and *plan sink.* It can also control ACTUATORs directly and is, therefore, an *action source.* It translates between the human inputs and requests and the probabilistic representation of the sensor network.

A FRAME represents the physical platform. Its main responsibility is to perform self-localization in the global coordinate system and to make its pose available to other components. Note that FRAME does not participate in information fusion or decision making.

The six components have different responsibilities, require access to different information to operate, and encapsulate different parts of the reference model. Table 3.4 shows the mapping

Component Type	G	$\mathcal{P}$	$\mathcal{A}$	F	$\mathcal{M}$	$\mathcal{Z}$	H	$\mathcal{B}$	D	U	T	π_M	π_A
SENSOR						✓	✓						
NODE	✓				✓			✓	✓			✓	
ACTUATOR		✓	✓	✓									
PLANNER										✓	✓		✓
UI						✓	✓						
FRAME		✓											

Table 3.4: Probabilistic reference model mapped to canonical components. Sparse entries highlight modularity of the system (just over one fifth of the cells are check-marked).

of the elements of the reference model to component types. Modularity of the architecture is apparent from the sparsity of the table. Observation models are confined to SENSOR and UI, actuation model to ACTUATOR. Fusion algorithm, feature models, messages, and communication policy are the responsibility of NODE. Utility, time horizon, and actuation policy are the domain of PLANNER. Platform state is shared between ACTUATOR and FRAME.

The mapping of roles to components is not unique. The term *canonical* is meant to remind that the list of six components is not exhaustive. A valid component can play any number of roles (from one to all ten). It would be possible, for example, to redefine PLANNER, instead of NODE, to play the roles of *plan fuser* and *distributor*.[20] Practical experience has shown that the list of six canonical components, as defined here, is sufficient for implementing a wide range of systems solving the DIG problem.

3.5.2 Service Interfaces

Component interaction is captured in the services which they provide to each other. *Interfaces* formalize inter-component services. Technically, an interface is "a set of named operations that can be invoked by clients" [224]. In other words, it is a specification of behavior that an implementer agrees to meet. The agreement is binding, i.e. an interface is a contract and by implementing an interface, a component guarantees to support a specified behavior.

In principle, every one of the thirteen associations shown in Figure 3.8 can give rise to

[20]The rationale for not doing so is the following. Because some of the platforms are likely to be passive, the density of NODEs is higher than the density of PLANNERs, so the probability of preserving network connectivity is also higher.

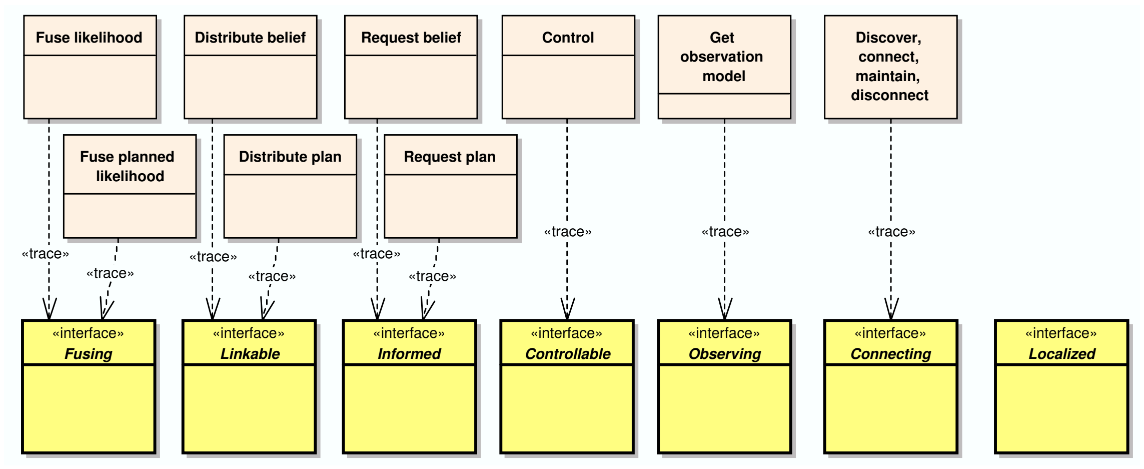


Figure 3.16: Associations mapped to service interfaces and shown as UML *interfaces*. Four associations are not shown as they are internalized within components: *sync belief, sync plan, equivalent, based on*. Three interfaces do not participate in information fusion or decision making *Localized*, *Connecting*, *Detailed* (the last one is not shown).

an inter-component service. Canonical components internalize some of the interactions for efficiency, so the actual number of services is lower. Mapping from associations to service interfaces is shown in Figure 3.16. Out of thirteen associations appearing in the structural diagram, five are implemented as services. The ubiquitous system-configuration service is also included. An additional service which provides platform pose is added. Thus the total number of services is eight.

The description of services provided below includes name, responsibilities, associated interface, assumptions, and key algorithms [92]. Interface definition in ASN is object-oriented, meaning that each interface has a data object associated with it. The mapping of interfaces to data objects is summarized in Table 3.5. Following a convention, interface names are adjectives. For clarity, they will be capitalized and set in italics, for example *Informed*.

The observation fusion service (*Fusing* interface) is used by *belief sources* to contribute their observations to the network for fusion. This service must implement a fusion algorithm which is described in detail in Chapter 4. The same interface accepts future observations, as part of decentralized decision making described in Chapter 5. The interface is identical for both robotic (SENSORs) and human (UIs) sources of information. The incoming data object is observation likelihood, b^k_{li}. As a special case, the same interface propagates deterministic information through the entire system. In particular, it can be used to adjust the utility

Interface	Data Object	
	receive	*transmit*
Fusing	b^k_{li}	–
Informed	–	b^k_{xi}
Linkable	b^k_{xj}	b^k_{xi}
Observing	–	H_i
Controlling	a^k_i	F_i, b^k_{pi}
Localized	–	b^k_{pi}
Connecting	–	–

Table 3.5: Interface definition in terms of data objects received and transmitted by the component which implements the interface. Note that only the data flow objects are shown. A separate configuration flow is possible for all interfaces. For example, the *Fusing* interface can transmit details of the global coordinate system and the *Informed* interface can receive QoS requests concerning the environment information it transmits.

weights used in decision making.

The environment information service (*Informed* interface) is used by *belief sinks*, both robotic (PLANNERs) and human (UIs). Note that a *sink* needs to connect to a single, most easily accessible component which provides this service. Information originated in other parts of the network is automatically accounted for through the process of information fusion. The outgoing data object is the local belief about the state of the environment b^k_{xi}. Quality of Service (QoS) requests may be used to specify the scope and desired quality of the information request. In particular, a component may be interested in the entire belief or an estimate based on the belief. Both current and future beliefs can be accessed through this service.

The belief synchronization service (*Linkable* interface) is used by *belief distributors* to initialize, maintain, and repair peer-to-peer communication links with each other. This service requires an algorithm for pair-wise synchronization of beliefs on the connected NODEs. The data flow is bidirectional, with local belief b^k_{xi} sent and external belief b^k_{xj} received.

The sensor access service (*Observing* interface) describes the sensing capabilities of the platform using a domain-specific model. It is used for the purposes of sensor management and platform control. The model H_i may include observation function, sensing constraints, observation costs, etc. As an example of the interface usage, PLANNERs use the sensor model to plan actions and observations.

The actuator access service (*Controllable* interface) describes the actuation domain-specific model. The actuator can be a simple motion device or a sophisticated platform. The outgoing data objects are the actuation model F_i and the current relevant state b_{pi}^k. The model F_i may include control influence functions, control and state constraints, actuation costs. The same interface is used to send commands to the actuator for execution. The incoming data object is an action a_i^k. For example, a PLANNER can request the actuation model, the current state, and, having made a decision, send back a control action for execution.

The localization service (*Localized* interface) provides components with information on the platform's position in the world. The outgoing data object is the platform's position in the global coordinate frame. Implementation of the interface requires a localization algorithm which is intentionally left unspecified. Observation preprocessing is an example of the use of this interface: SENSORs use the platform pose estimate and the known sensor offset to convert observations from a local to the global coordinate frame.

The most commonly used service is system configuration (*Connecting* interface). In a decentralized system, this functionality must be implemented by each component.[21] Key algorithms for the typical stages of network configuration include service discovery, selection, addressing, connection and disconnection.

Finally, the *Detailed* interface is used to communicate component status to the UI for the purposes of debugging, compiling global topology, performance metrics, etc. Each component implements this service. Enabling this service significantly increases communication requirements but it is invaluable during the design and validation stages of system development.

3.5.3 Use Case Realizations

After the service interfaces are defined, it is possible to describe in detail how the DIG use cases in Figure 2.4 are implemented in ASN. That is, having decomposed the system, it is time to take a look at how the different pieces fit together.

Figure 3.17 shows a realization of the *Operator* use case with three canonical components. An operator interacts with a UI component to access the information available to the net-

[21] Implementation of this service is typically encapsulated within a *component model*, which "defines specific interaction and composition standards" available to and required of component instances [76]. For a review of the three general-purpose component models in use today, see [131].

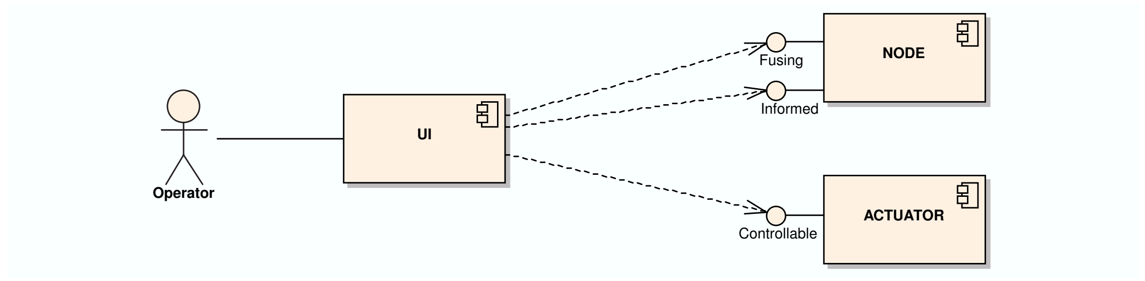


Figure 3.17: Realization of the *Operator* use case. Compare with the use case definition in Figure 2.4. In this and the following UML structural diagrams, a circle symbol is used for interfaces. Dependency relationships are shown as dashed arrows.

work through the *Informed* interface. The operator can also contribute new information in the form of observation likelihoods through the *Fusing* interface. Direct teleoperation is allowed by connecting to an ACTUATOR through the *Controllable* interface. The concept of local interactions limits the application of the teleoperation scenario to the immediate neighborhood of the human operator. To overcome this limitation, distribution of actions needs to be implemented (see Table 3.3).

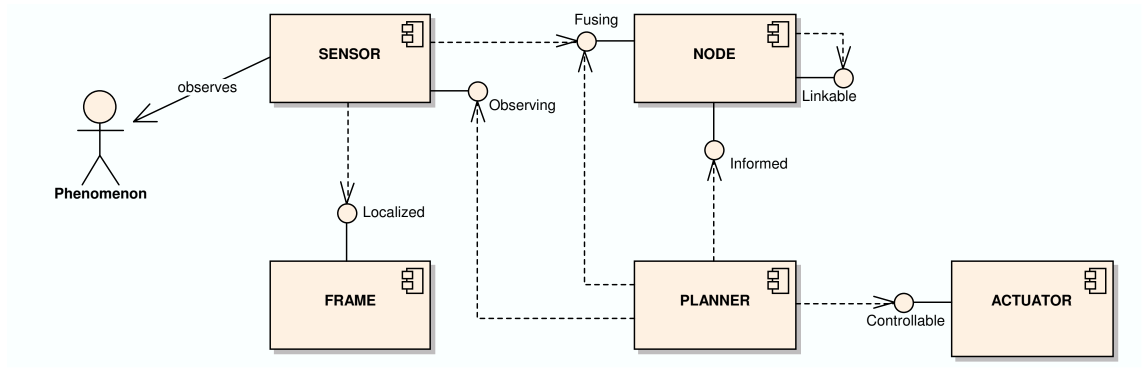


Figure 3.18: Realization of the *Collect Information* use case. Compare with the use case definition in Figure 2.4.

Figure 3.18 shows a realization of the *Collect Information* use case. As expected, all ASN interfaces are engaged in implementing this central use case. SENSORs submit their observations to NODEs using the *Fusing* interface. For observation preprocessing, SENSORs need platform position obtained through the *Localized* interface. NODEs connect to each other and exchange beliefs using the *Linkable* interface. PLANNERs get the beliefs about the current and future state of the environment from NODEs through the *Informed* interface.

Observation models are accessed by PLANNERs through the *Observing* interface. PLANNERs issue actions to ACTUATORs through the *Controllable* interface.

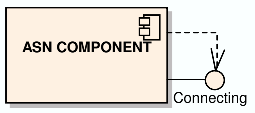


Figure 3.19: Realization of the *Configure System* sub-case. Compare with the use case definition in Figure 2.4.

Figure 3.19 shows a realization of the *Configure System* use case. Each component implements the *Connecting* interface which allows navigation through the interfaces provided by remote components.

A closer examination of Figures 3.17–3.19 reveals that the ASN implementation with canonical components satisfies the *acyclic dependency principle* (ADP) [149] . This software design principle states that the dependencies between a set of software elements should not form a cyclic graph. The original version applied only to the *build-time* dependencies between source-code packages. Collins-Cope proposes to extend the principle to *run-time* dependencies between executable components [35], arguing that if two components cannot be deployed separately because they require each other to run, then they should be combined into a single component. Although compliance with the ADP is not an explicit design goal for ASN, the acyclic property of run-time dependencies certainly simplifies the deployment procedure.[22]

3.5.4 Degree of Functional Decentralization

The concept of variable degree of decentralization was brought up in Sections 3.4.3–3.4.4 in the context of component roles and scalability of their interactions. This section revisits this important topic because the idea is easier to illustrate using the definitions of canonical components.

[22]An argument against strict compliance with ADP is that a component may retain partial functionality when not all of its dependencies are satisfied. In UML, the required interfaces are those required by a component in order to “offer its *full* set of provided functionality” [75] (author’s emphasis). This is different from the situation with source-code which cannot be “partially compiled”.

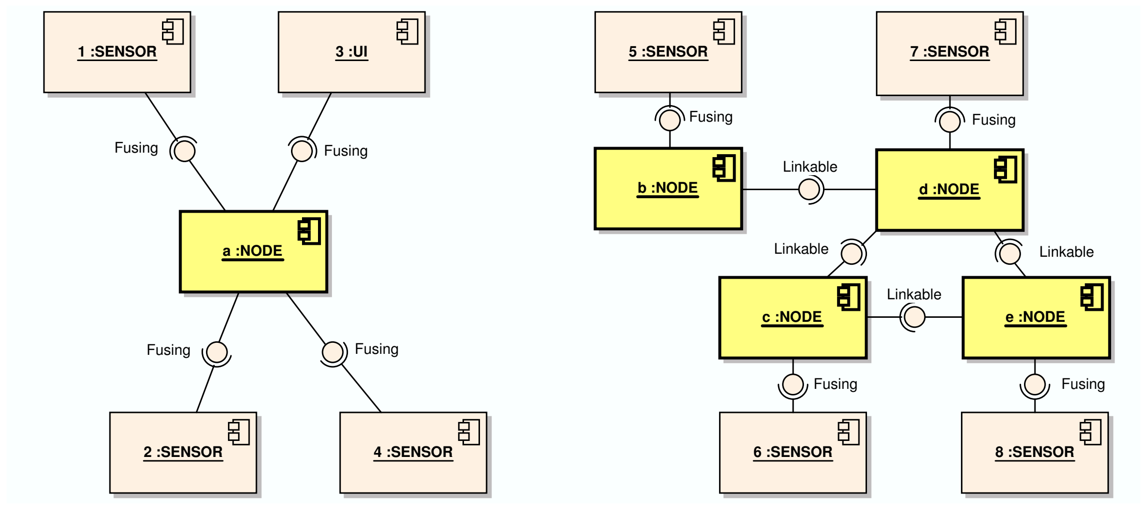


Figure 3.20: The range of decentralization options in the task of information fusion. On the left, a fully centralized system has one NODE for all *belief sources*: SENSORS and UIS. On the right, a fully decentralized system has a dedicated NODE for each *belief source*.

Figure 3.20 shows two topological extremes in structuring information fusion: fully centralized and fully decentralized. On the left, all *belief sources*, SENSORS and UIS, submit their observations to a single NODE. An information fusion system cannot be more centralized. On the right, every *belief source* connects to a dedicated NODE. The NODES, in turn, link to each other. In this case, the system cannot be more decentralized.

Figure 3.21 applies the same logic to the decision making function of the network. Fully centralized topology on the left has a single PLANNER generating commands for all *action sinks*. In the fully decentralized case on the right, each *action sink* has a dedicated PLANNER. All PLANNERS are shown connected to a single NODE for clarity.

A system in which both the information fusion and decision making tasks are fully decentralized is shown in Figures 3.22. Every platform now has its own SENSOR, NODE, PLANNER, and ACTUATOR. The NODES on different platforms connect to each other and exchange beliefs.

Optimum degree of (de)centralization depends on many factors including the state of the environment and the relative costs of computation, communication, and storage for each platform. As most of these factors are unknown *a priori* and are not constant, an architecture must be flexible enough to allow a range of different topologies depending on the task, resources, and operating conditions. Global optimality is hard to achieve and is not an

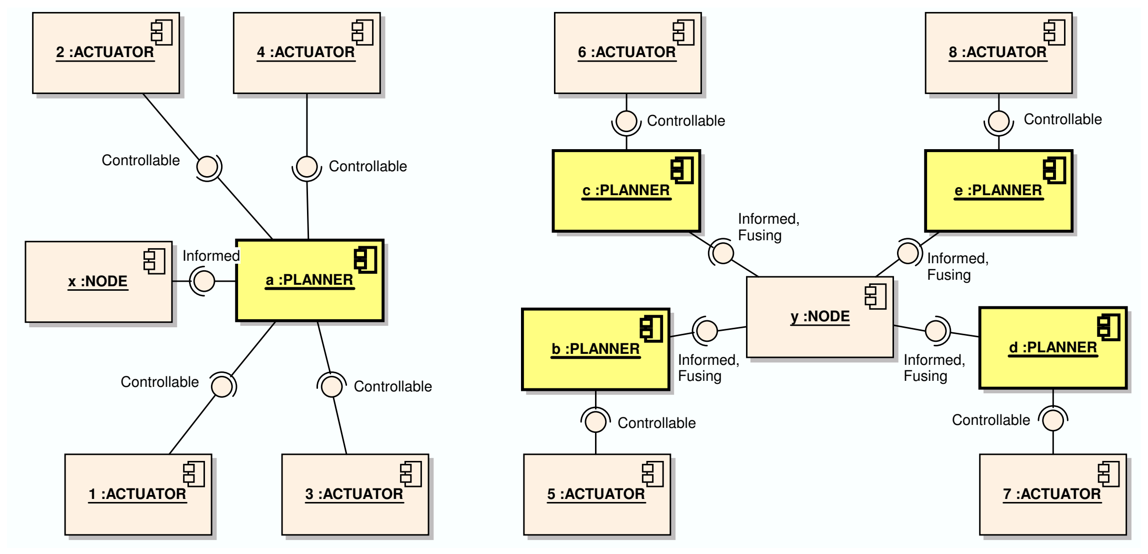


Figure 3.21: The range of decentralization options in the task of decision making. On the left, a fully centralized system has one PLANNER for all ACTUATORs. On the right, a fully decentralized system has a dedicated PLANNER for each ACTUATOR.

objective of this work. As with other large, complex networks, the structure of the sensor network arises naturally during the network growth [11]. Connection decisions are made one at a time and are based on local partial information. More capable and reliable NODEs with cheaper power, processing, and communication resources will accept more connections and become the hubs of the network.

In this context, it may also be possible to include on-demand *activation* of components as part of system's self-configuration. Consider as an example the system on the left side of Figure 3.20. If spare computing resources are available, it may be beneficial to activate a second NODE which will take on some of the processing load. This type of self-organization is likely to require new algorithms to be developed.

3.6 Canonical Components

Section 3.5.1 informally described the grouping of roles into component types. Section 3.5.3 showed how the canonical components cooperate to realize the DIG use cases. This section *defines* the canonical components by listing the interfaces which they provide and require. Table 3.6 shows a summary of the interfaces provided and required for all six canonical

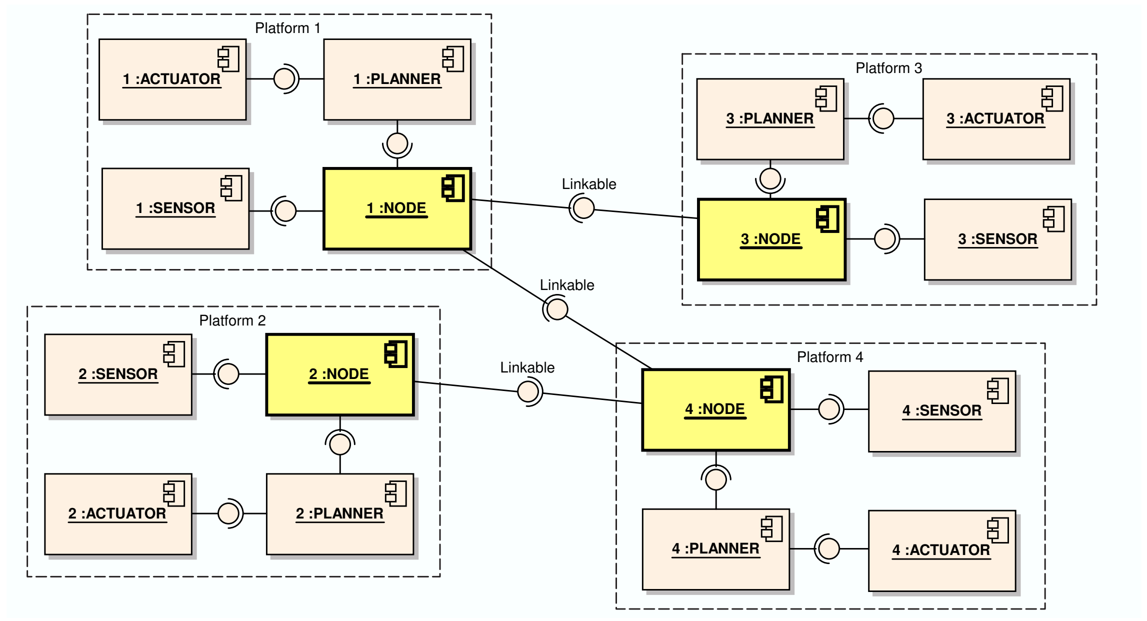


Figure 3.22: A system fully decentralized in both information fusion and decision making.

components. The *Connecting* and *Detailed* interfaces are widespread and therefore are not shown. The *Connecting* interface is provided and required by all components. *Detailed* interface is provided by all components and is required by the UI.

Component	*Localized*	*Linkable*	*Fusing*	*Informed*	*Observing*	*Controllable*
FRAME	●/○					
NODE	○	●/○	●	●		
SENSOR	○		○		●	
ACTUATOR	○					●
PLANNER			○	○	○	○
UI	○		○	○		○

Table 3.6: Canonical component types and the interfaces they provide (●) and require (○). *Connecting* and *Detailed* interfaces are not listed because they are provided by all components.

Component descriptions in this section cover functionality, specification, and realization. The *black-box* view of a component is its specification. It contains enough information to allow the component to be "wired" into the system but hides the implementation details. The *white-box* view of a component exposes the internal structure of its realization. It shows how the internal parts are connected to the external interfaces. Where applicable, additional constraints and limitations, some algorithmic detail, and other implementation details are listed. Note that by virtue of being a replaceable element, a component may have multiple realizations [75], all compliant with its specification. For this reason, the realizations described in this section are shown for illustration only.

3.6.1 Frame

The FRAME component is responsible for platform localization, communication, power monitoring, and safety. The black-box specification of FRAME is shown in Figure 3.23. A FRAME interacts with the rest of the system through two ports[23]: it requires the *Hardware* interface and provides localization services through the *Localized* interface.

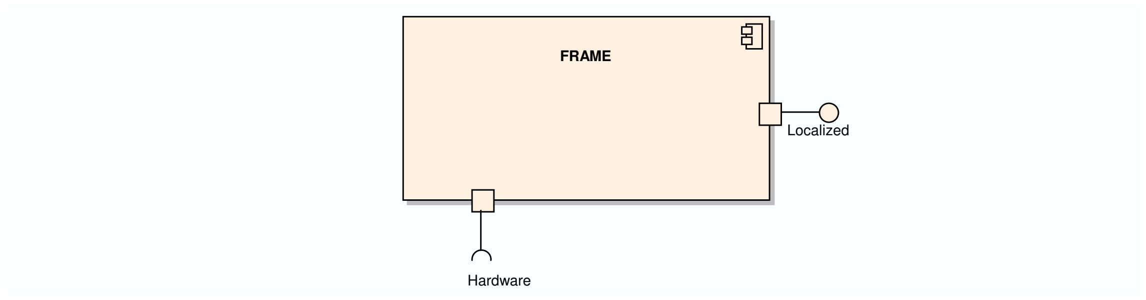


Figure 3.23: Specification of the FRAME component. Squares correspond to ports which contain required or provided interfaces. Circle notation is used for provided interfaces, half-circles (sockets) for required.

A FRAME's primary responsibility is to maintain the belief about platform's location measured in the global coordinate system. The localization mechanism is intentionally left unspecified. The *Localized* interface shields the consumers of the data from the details of localization algorithm.

The hardware port connects to sensors, actuators, etc. As a minimum, a FRAME needs to access platform's hardware to perform localization. For a particular implementation of FRAME component, the *Hardware* interface describes the hardware which is assumed to be present. This interface is not listed as part of the ASN definition because it is strictly local, i.e. all hardware is accessed within the same platform.

[23]UML A component's *port* specifies a distinct interaction point between [the component] and its environment [75]. Port's type is determined by the set of provided and/or required interfaces associated with it. Any connector to a port must have the matching interfaces.

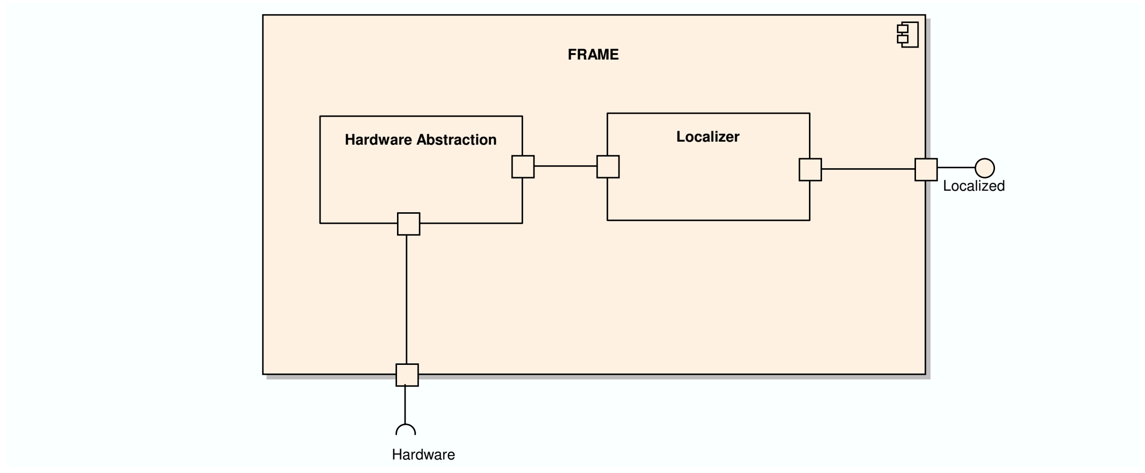


Figure 3.24: A possible realization of the FRAME component.

A possible FRAME realization is shown in Figure 3.24. Examples of localization algorithms are GPS, beacon-based, sensor-based, SLAM, stationary pre-configured, etc. Hardware is typically accessed through a hardware abstraction layer.

3.6.2 Node

The NODE component is responsible for fusing information supplied by *belief sources* and for distributing it throughout the network. As part of this task, it performs belief prediction, data association, and belief update. The NODEs do not generate or use the information they store, rather they make it available to others.

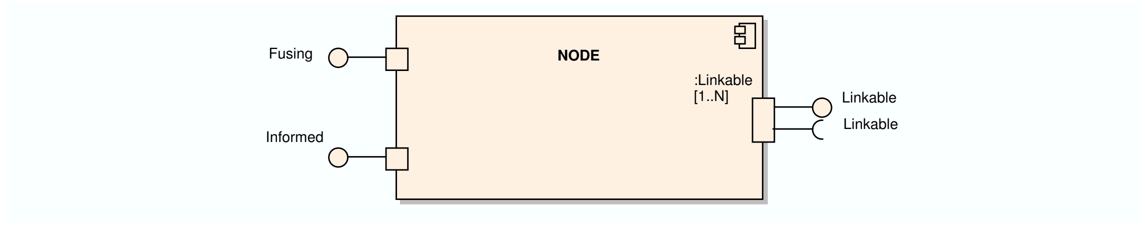


Figure 3.25: Specification of the NODE component. A NODE can be connected to several other NODEs through the *Linkable* interface. Communication through the *Linkable* interface is peer-to-peer and the information flow is bidirectional, i.e. each *Linkable* port provides and requires the *Linkable* interface.

The black-box specification of the NODE component is shown in Figure 3.25. A NODE interacts with other components through four types of ports which provide the *Fusing*, *Informed* and *Linkable* interfaces and require the *Localized* interface. Multiple *Linkable* ports are possible.

Both the *Fusing* and *Informed* ports implement the server function in a client-server (many-to-one) connection. The *Fusing* port can accept multiple clients each submitting observation likelihoods. The main flow of information is incoming. The *Informed* port can accept multiple clients each receiving updates on the state of the environment. The main flow of information is outgoing. The maximum number of clients for both client-server links can be configured.

Each of the multiple *Linkable* ports implements one side in a peer-to-peer connection. The communication traffic is bidirectional and symmetric. The connection through each port can be initiated internally (the interface is required in this case) or from the outside (the interface is then provided). The maximum number of *Linkable* ports can be configured.

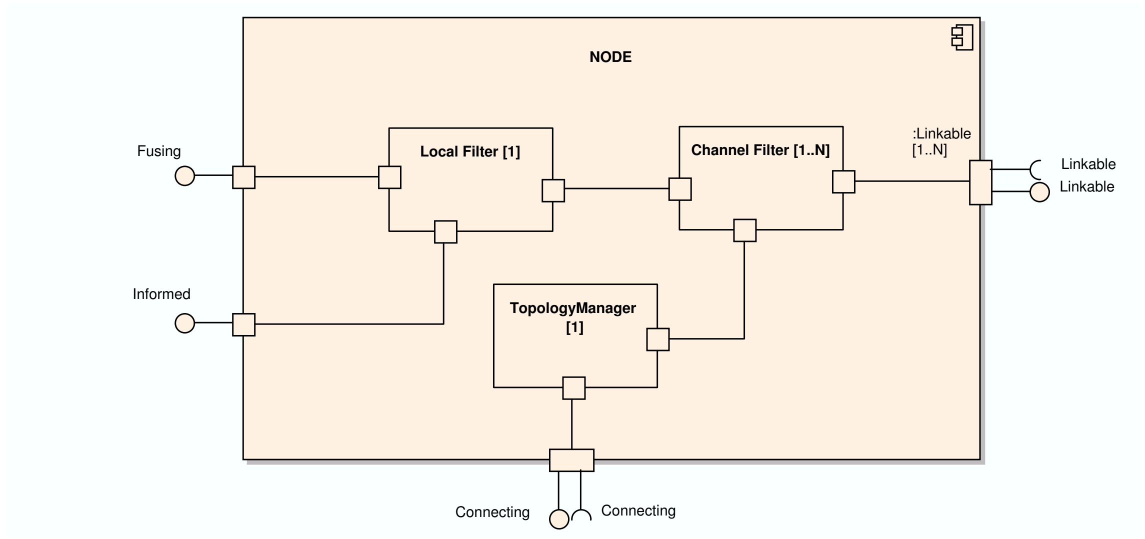


Figure 3.26: A possible realization of the NODE component. Multiplicity of the *Linkable* port and the corresponding Channel Filter part is N.

The main internal subsystems of the NODE are shown in Figure 3.26. The responsibilities of the subsystems are indicated by the interfaces they realize.

The *Local Filter* is responsible for fusing observations submitted by *belief sources* and maintaining a local belief on the current state of the environment. It realizes two interfaces: *Fusing* and *Informed*, used by *belief sources* and *sinks* respectively. The Local Filter generates state estimates on the basis of observed, predicted and communicated information. Other infrastructure such as Channel Filters and the topology manager exist only to support the proper functioning of the Local Filter.

The *Channel Filter* is responsible for maintaining a link (or *channel*) to other NODEs. It implements the *Linkable* interface. The Channel Filter serves one purpose: it maintains the state of the link between two directly linked NODEs. As will be shown later, common information shared by two NODEs is a key aspect of the channel state as it is critical to fusing beliefs originating on different sensing platforms. Together, the Local and Channel Filters implement the central idea of ASN: local aggregation of information. It is important for efficient inter-NODE communication and ultimately leads to scalability.

The *Topology Manager* selects NODEs in the neighborhood to connect to using the *Linkable* interface. While every component must have basic self-configuration functionality, the links between NODEs however are most important. That is why it is separated into a dedicated module. The local connection decisions made by individual Topology Managers shape the global topology of inter-NODE links.

A more detailed description of the structure and the algorithms of the NODE is given in Chapter 4.

3.6.3 Sensor

The SENSOR component is responsible for gathering information from the environment by processing raw measurements from physical sensors.

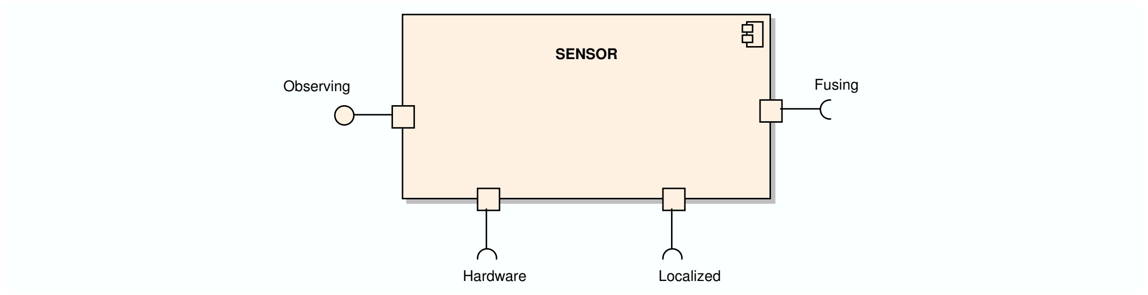


Figure 3.27: Specification of the SENSOR component.

The black-box specification of the SENSOR component is shown in Figure 3.27. A SENSOR interacts with the rest of the system through four ports. It requires three interfaces: *Localized* for platform pose, *Fusing* to process its observations, and *Hardware* for access to physical sensors. A SENSOR provides one interface: *Observing* to expose its sensor model.

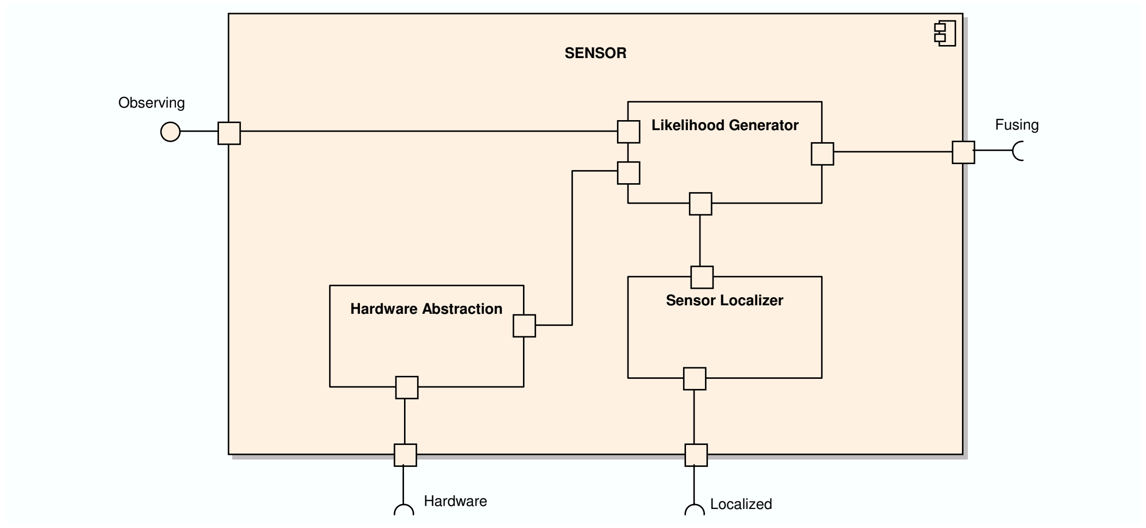


Figure 3.28: A possible realization of the SENSOR component.

Figure 3.28 shows a possible SENSOR realization. Likelihood Generator is a key module which converts raw measurements in $z_i^k \in \mathcal{Z}_i$ to observation likelihoods in $b_{li}^k \in \mathcal{B}$ defined on $\mathcal{X}$. Platform position obtained through the *Localized* interface are used for spatial alignment. Once generated, the observation likelihood is submitted for fusion using the *Fusing* interface. The *Fusing* interface expects observation likelihoods for environment features in global coordinate system. The SENSOR reads measurements from the sensing hardware in local (often non-cartesian) coordinate system.

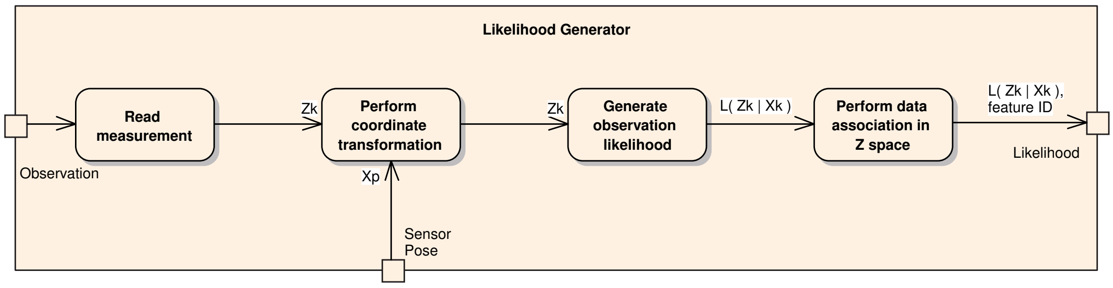


Figure 3.29: Data transformation steps inside the SENSOR component.

The data processing steps inside the SENSOR component are shown in Figure 3.29. The preprocessing stage includes coordinate transformation and spatial alignment. Physical sensor measurements are often made in polar coordinated system and the global coordinate frame is invariably cartesian. Regardless of the type of the coordinate frame, most, if not all, sensors report measurements relative to its mounting point, i.e. in local or relative coordinate system. To perform the transformation to the global coordinate frame, the SENSOR needs to know its current pose. It is calculated by the Sensor Localizer based on the current FRAME pose and the sensor's pose in FRAMEs local coordinate system — the sensor's offset.

The next step is to calculate observation likelihood based on the likelihood function L_i. Building probabilistic sensor models is a well-studied topic covered in most standard text on data fusion and tracking [150, 219]. Optionally, initial data association may be performed at the SENSOR. Often data association is easier to perform in the measurement space $\mathcal{Z}_i$ than in the state space $\mathcal{X}$. The measurement itself and the measurement space remain unavailable to the NODE. Finally, the observation likelihood (and the results of data association) is submitted to the NODE through the *Fusing* interface.

Communication between the SENSOR component and the physical sensors is performed through the *Hardware* interface. For each particular pair of SENSOR implementation and a physical device the interface definition is very specific. There can be one-to-one, one-to-many, many-to-one, and many-to-many types of relationship between the two. Figure 3.30 illustrates the options with multiple SENSORs using a laser, a camera, or both. The architecture emphasizes maximum flexibility: any arrangement is allowed as long the provided and required interfaces match.

The case of one-to-many (c) represents centralized information fusion at the measurement level (in $\mathcal{Z}$ space). The sensor component receives data streams from two or more physical sensors and forms observation likelihoods based on all the data available. Compare case (c) with two instances of case (a), one for the laser and the camera. The advantage of (c) over (a) is that potentially more information can be recovered by using a combined sensor model which takes the correlation of the observations into account. Case (a), on the other hand, offers more modularity because each SENSOR component may be designed independently. From the algorithmic point of view, this choice corresponds to *data* and *decision fusion* [39, 111]. With data fusion it is impossible to modularize information fusion

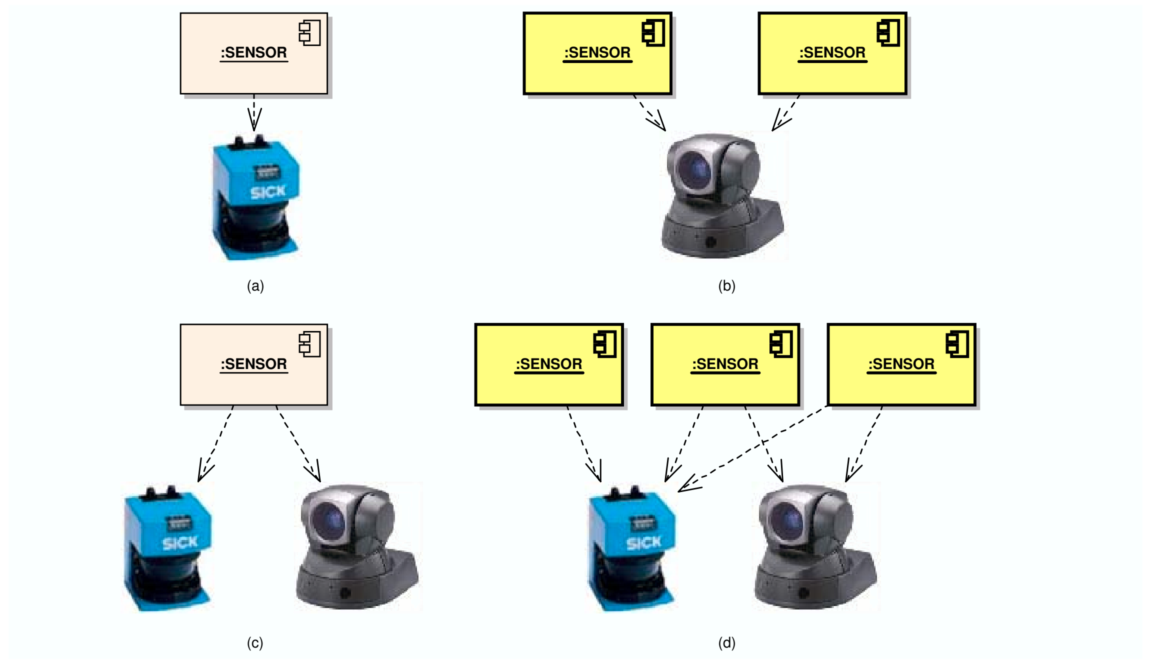


Figure 3.30: Possible relationships between SENSOR components and sensing hardware: (a) one-to-one, (b) many-to-one, (c) one-to-many, and (d) many-to-many. Two types of sensing hardware are shown: a laser range finder and a video camera. Highlighted SENSORs (b, d) produce correlated observations and must be treated with caution.

when heterogeneous sensors are employed. This precludes collaborative signal processing which requires exchanging raw observations.

The cases of many-to-one and many-to-many (b, d) must be treated with caution because the likelihoods produced by the logical SENSORs are correlated if the same raw measurements were used. It is possible to use this setup if the correlations are explicitly accounted for (outside of the ASN framework), the different SENSORs produce likelihoods on different types of features whose independence is not assumed, or the SENSORs simply use alternating measurements thereby eliminating correlations altogether.

3.6.4 Actuator

The ACTUATOR component is responsible for providing an interface to a controllable device. The device in this case can be as simple as a 1-DOF physical actuator or as complicated as a complete robot.

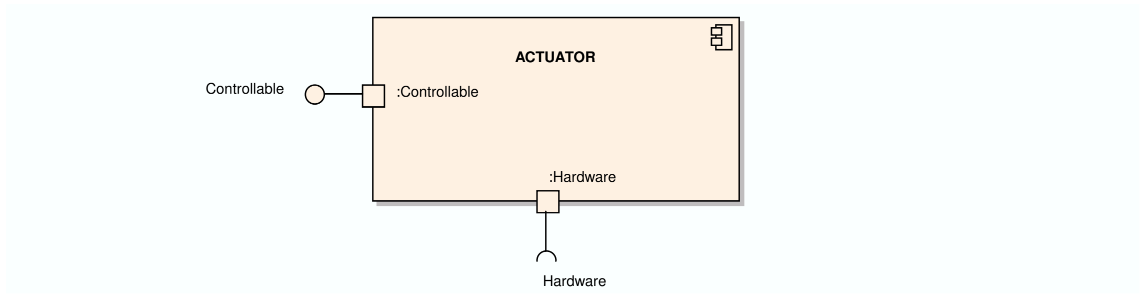


Figure 3.31: Specification of the ACTUATOR component.

The black-box specification of the ACTUATOR component is shown in Figure 3.31. An ACTUATOR interacts with the rest of the system through two ports. It provides one interface: *Controlling*, which exposes its actuation model. An ACTUATOR requires one interface: *Hardware* to access the physical actuators.

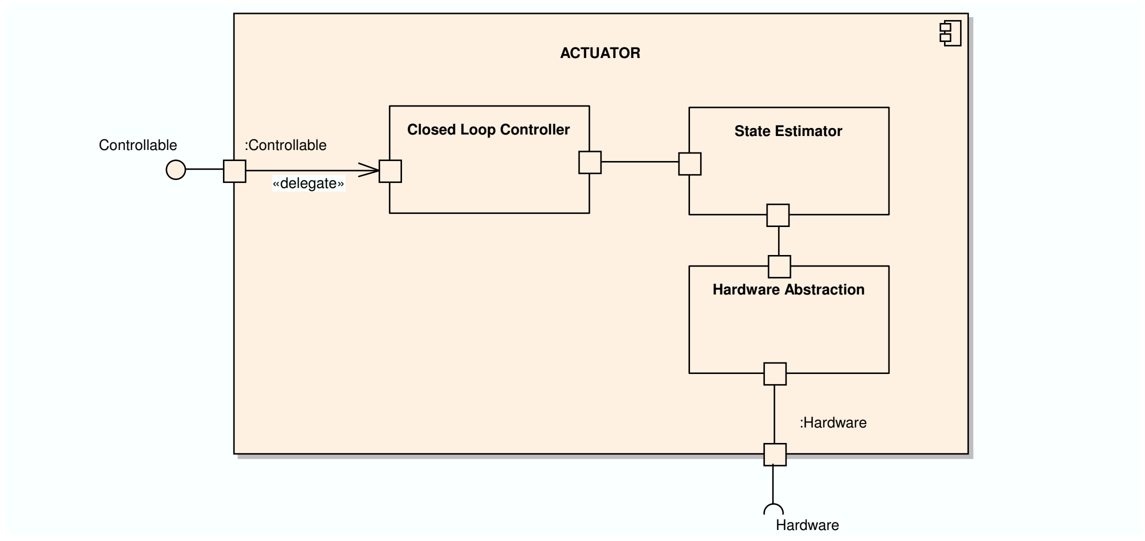


Figure 3.32: A possible realization of the ACTUATOR component.

Figure 3.32 shows a possible realization of the ACTUATOR component. Typically, there is a Closed-Loop Controller connected to a State Estimator and which receives reference input through the *Controllable* interface. If the controlled property of the ACTUATOR is spatial, it must be converted to the global coordinate system.

Interaction between the ACTUATOR component and the physical actuators is governed by the *Hardware* interface. There can be one-to-one and one-to-many relationships between the

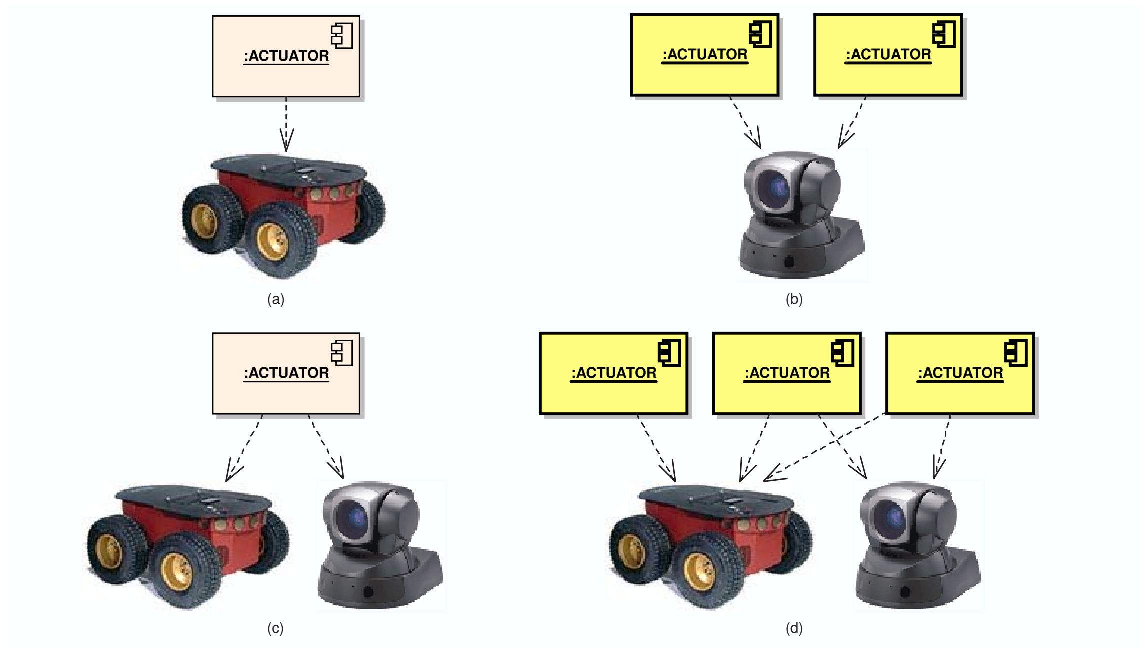


Figure 3.33: Possible relationships between ACTUATOR components and actuation hardware: (a) one-to-one, (b) many-to-one, (c) one-to-many, and (d) many-to-many. Two types of actuation hardware are shown: an indoor robot and a pan-tilt-zoom camera. Highlighted ACTUATORs in (b, d) may receive conflicting control inputs and must be treated with caution.

two types of actuators. An example in Figure 3.33 shows multiple ACTUATORs controlling a vehicle, a pan-tilt camera, or both. The architecture emphasizes flexibility: any arrangement is allowed as long the provided and required interfaces match.

The one-to-one case (a) is the most typical. An example of one-to-many scenario (c) is an articulated sensor mounted on a vehicle. This scenario may be preferred if the joint definition of multiple actuators captures an important coupling leading to better performance. An alternative of having two ACTUATORs with one-to-one relationship to the devices is more modular.

The cases of many-to-one (b) and many-to-many (d) must be treated with caution because the commands received from different PLANNERs may be in conflict. If for some reason this setup is preferred, the conflicts must be detected and resolved outside of the ASN framework.[24].

[24]By possibly implementing the role of *action fuser* which was left blank in Table 3.3

3.6.5 Planner

The PLANNER component is responsible for generating commands for the actuator which will maximize a certain utility function. The choice of commands is typically made based on the current state of the world, state of the actuator under control, together with the observation and actuation models.

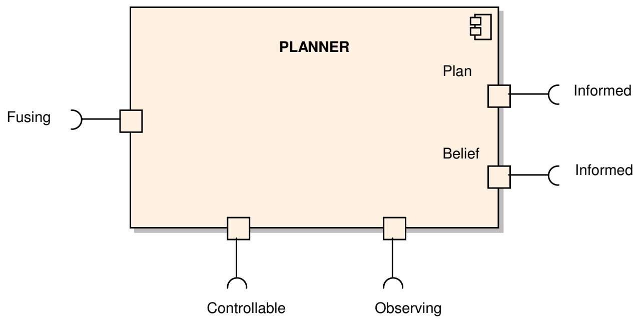


Figure 3.34: Specification of the PLANNER component.

The black-box specification of the PLANNER component is shown in Figure 3.34. The PLANNER interacts with other components through five ports. It provides no services of its own. It requires several interfaces: *Informed* for the current and future planned belief about the state of the world, *Controlling* for the state and description of the actuators, and *Observing* for the description of the sensors. In order to make cooperative plans, a PLANNER needs the *Fusing* interface to submit future planned likelihoods.

Figure 3.35 shows a possible realization of the PLANNER component. Two main modules are identified: Solver and Negotiator. The Negotiator iterates through possible action plans and chooses an appropriate one based on the preferred ordering of actions supplied by the Solver. The division of the PLANNER into two modules is for illustration only, since some solution approaches do not make this distinction. A more detailed description of the structure and the algorithms of the PLANNER is given in Chapter 5.

3.6.6 User Interface

The UI is a single point of interaction between an operator and the sensor network. It displays network information to the human operator. It also allows the operator to submit

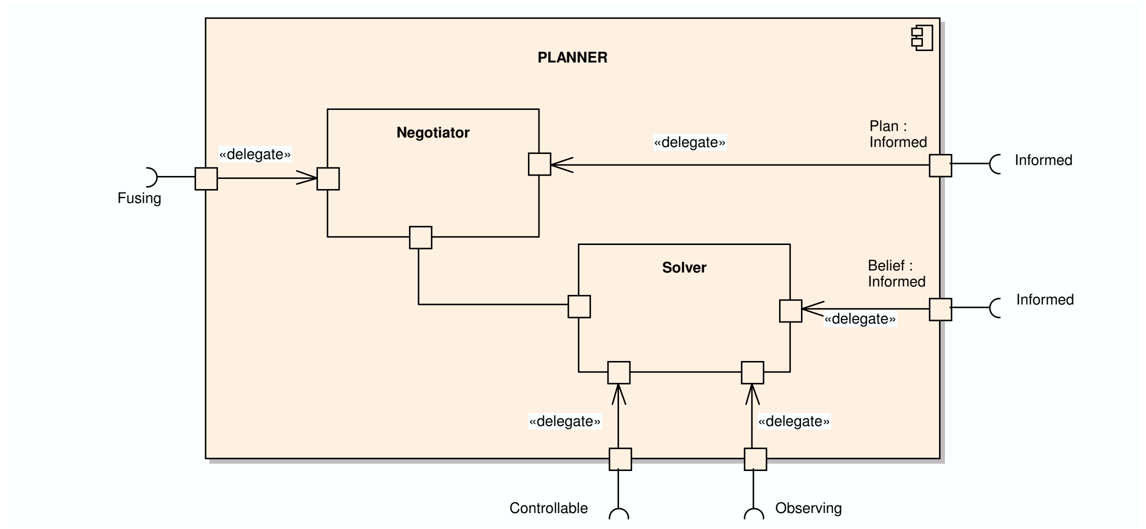


Figure 3.35: A possible realization of the PLANNER component.

human observations for fusion with the network's belief. An operator can also submit a change in utility weights which will affect the decisions made by individual platforms. A UI also allows direct teleoperation of platforms. Finally, the UI may be used to compile global information about the state of the network. This functionality is primarily intended for debugging purposes.

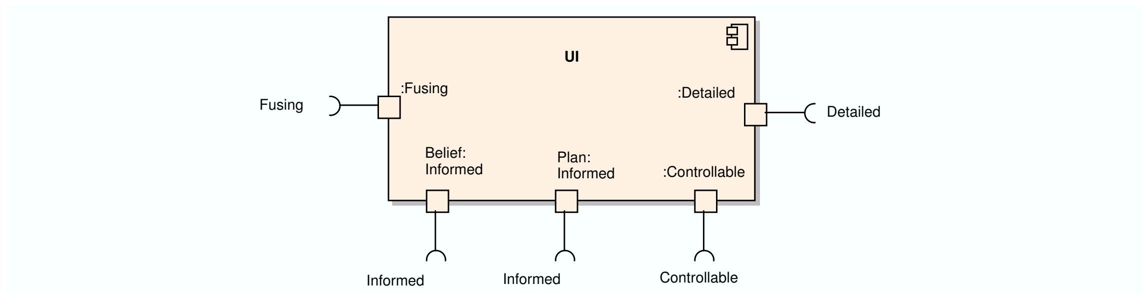


Figure 3.36: Specification of the UI component.

The black-box specification of the UI component is shown in Figure 3.36. The UI interacts with other components through up to five ports. It provides no services and requires five interfaces: two *Informed* interfaces for current and future beliefs, *Fusing* for submitting human observations, *Controlling* for teleoperation, and *Detailed* for debugging.

Figure 3.37 shows the internal structural view of the UI component. The primary goal of

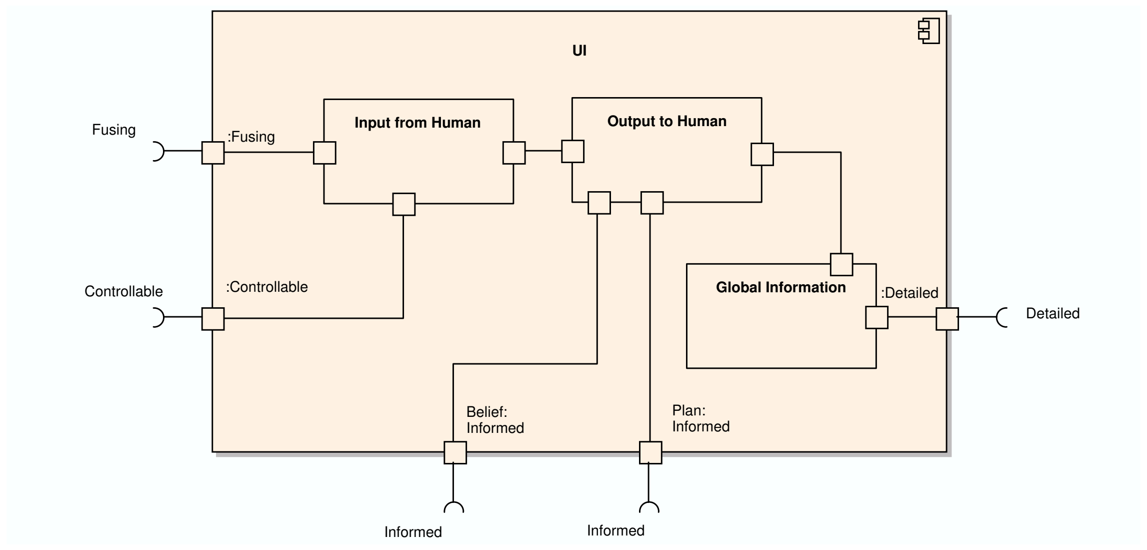


Figure 3.37: A possible realization of the UI component.

the UI is to present the information about the environment collected by the network to the human operator. This task is carried out by the Output to Human module. The network belief about the world state is accessed through the *Informed* interface. Because of this feature, the display of environment information in ASN is as scalable as the network itself. The user interface can be graphical, textual, or another modality appropriate for the type of information being gathered.

The information about the network itself may be useful especially for debugging and status monitoring purposes. For this reason, the optional Global View module can aggregate component information obtained through the *Detailed* interface. Based on individual reports, the Global View can compile the global topology of the network (within a direct communication range). While convenient, it is important to realize that relying on this information makes the UI a computational or communication bottleneck. In addition to increasing the bandwidth requirements, the task of keeping track of component information also necessitates either the use of a broadcast medium or a direct link between every component and every GUI station. Both options are undesirable as they violate the constraints of decentralization and locality.

The operator input is handled by the Input from Human module. Others have suggested several classification systems for the roles that a human operator could play while interacting with robots. For example, Scholtz identifies mechanic, supervisory, and peer-to-peer levels of human-robot interaction [198]. In ASN, two types of operator controls are distinguished: environment-centric and component-centric. The former is scalable but relies on fully autonomous platform operation. The latter gives more authority to the operator but scales poorly with the number of platforms.

Environment-Centric Control. In the *information based* control mode, the information about the environment which has become available from outside of the network is entered into the network by a human operator. It is then fused into the system and will affect actions of one or more of the network components. This is a *peer-to-peer* mode in which the operators and the network components act on the same level of authority. The information is entered into the network as an observation likelihood b^k_{li} through the *Fusing* interface. The operator acts as a HUMAN SENSOR. Influencing the action of the network components by providing it with additional information is a natural extension of the underlying information fusion and decision making algorithms. The information entered by the operator only needs to be transmitted to a single NODE and the network itself propagates it through the rest of the system, potentially leading to actions by several platforms.

Component-Centric Control. Under *direct* type of control an action a^k_i or a sequence of actions is sent through the network addressed to the components which possess the attributes specified. These commands are to be executed directly and no autonomy is required on the part of the platform. This approach is a direct extension of the teleoperation or *mechanic* mode of operation. It may be invaluable in certain situations, but cannot be considered a primary control tool in large decentralized sensor networks. The actions are submitted through the *Controllable* interface and the UI acts as a HUMAN PLANNER. In ASN, teleoperation is limited within a direct communication range. For teleoperation outside of the platform's neighborhood, service lookup and action routing must be performed outside of ASN framework.[25]

The following two control methods can be classified as *supervisory* modes. Under the *mode switch* type of control, the platforms operate autonomously most of the time, but are periodically commanded to switch from one pre-programmed action policy to another. Under the *utility change* type of control, a change in the utility function U is sent to all or a subset of the network components addressed by appropriate attributes.

[25]By possibly implementing the role *action distributor* which was left blank in Table 3.3.

3.7 Custom Components

Components, by definition, are substitutable. This is achieved by simultaneously requiring strict compliance with interface definition and allowing full freedom in implementation. There are many reasons, why freedom in implementation may be insufficient and the component specification must be modified. This leads to the creation of *custom components* by modifying the existing or creating entirely different specifications.

This section shows three approaches to building custom components: by integrating functions of two standard ones, by adding functionality while changing the list of provided/required interfaces, and by defining super-components. Other approaches are certainly possible. What remains unchanged in the process are the ASN interfaces. This allows inter-operability between the canonical and custom components.

3.7.1 Function Integration

A variety of component types can be defined each implementing different combinations of component roles identified in Section 3.4.2. Recombination of roles compared to the canonical set may be prompted by algorithmic, implementation, or non-technical reasons.[26] Two examples in this section illustrate how a custom component can be created by combining functionality of canonical components.

As an example, consider a mobile platform. Figure 3.38(a) shows a partial design using two canonical components: a FRAME and an ACTUATOR representing the means of locomotion of the platform. The ACTUATOR depends on the *Localized* interface provided by the FRAME and exposes the *Controllable* interface to be used by *action sources*. Figure 3.38(b) shows an integrated custom component VEHICLE which couples the localization and actuator-handling functions internally but exposes the same two interfaces for external consumption. Thereby, efficiency is achieved while preserving inter-operation with the rest of the system.

As another example consider the storage of platform and world belief. Figure 3.39(a) shows the standard ASN approach whereby FRAME and NODE components are independent. A different configuration would result if a Simultaneous Localization and Mapping algorithm

[26]Houston and Norris [92] make a similar statement noting that multiple *logical* subsystems may be mapped to the same *physical* component to satisfy the needs of the overall system. The point made here is slightly different and is concerned with the recombination of roles at the level of logical components.

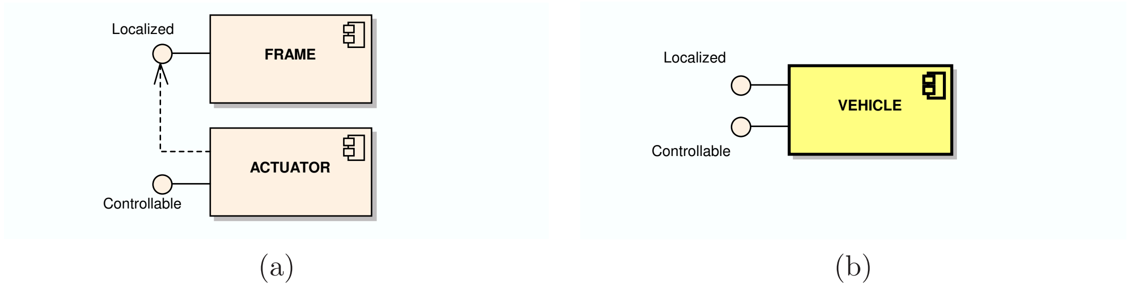


Figure 3.38: A custom VEHICLE component: (a) canonical implementation using a FRAME and a ACTUATOR and (b) custom implementation. The localization service provided by *Localized* interface to ACTUATOR is internalized inside the custom component but both interfaces must still be provided for the benefit of other components in the system. Notice that the interfaces remain unchanged.

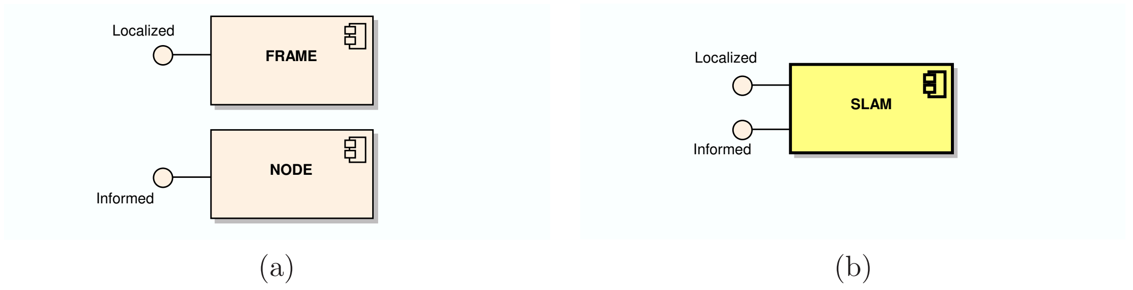


Figure 3.39: A custom SLAM component: (a) canonical implementation with a FRAME and a NODE and (b) a custom implementation. The custom component is responsible for both building the map of the environment and localizing the platform. Both the *Localized* and *Informed* interfaces must still be provided for the benefit of other components in the system.

(SLAM) is used which couples the estimation of the platform pose and the environment features. Transmitting the information between the two components would be impractical. As an alternative, SLAM component would have realized both the *Localized* and the *Informed* interface as shown in Figure 3.39(b).

3.7.2 Additional Functionality

Functionality of canonical components is limited. Nearly every new system will require some new capability and new algorithms. Extra functionality can be added to canonical components by changing the list of provided and required interfaces.

An example is shown in Figure 3.40. A canonical frame in panel (a) estimates its pose in the

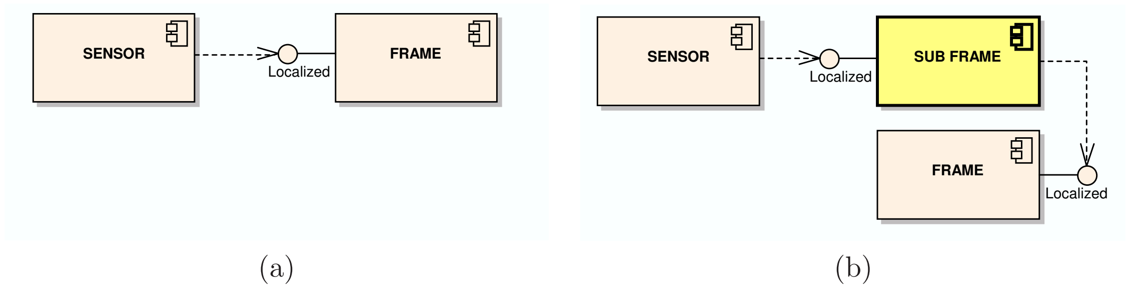


Figure 3.40: A custom SUB-FRAME component: (a) canonical implementation with a FRAME component which localizes itself in the global coordinate system and (b) a custom SUB-FRAME connected to another FRAME.

global coordinate system. A custom FRAME can be defined as in panel (b) which estimates its global position by measuring an offset relative to another FRAME, which acts as a base. The location of the base FRAME is accessed through its *Localized* interface. By connecting multiple FRAMEs, an arbitrary hierarchy of coordinate systems may be constructed to reflect the physical connectivity of sub-assemblies.

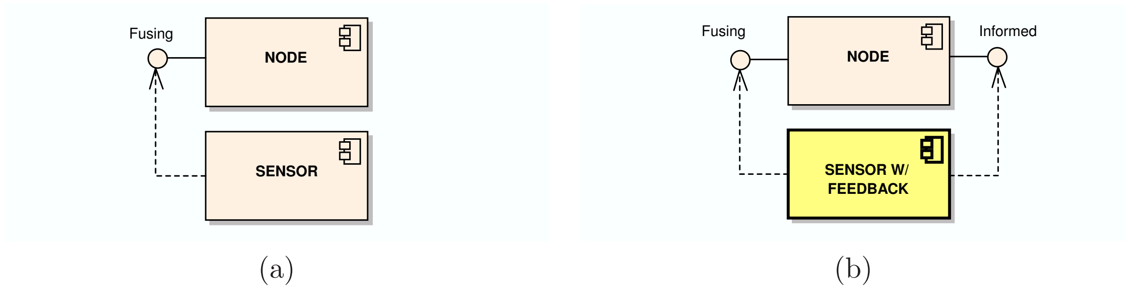
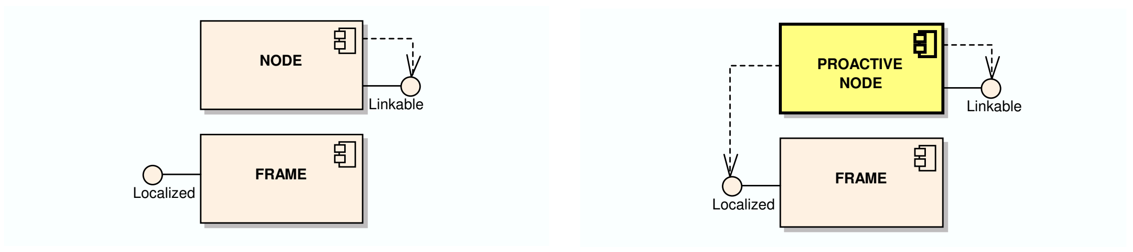


Figure 3.41: A custom SENSOR component with state feedback. The environment information from the NODE component is made available to the custom SENSOR through the *Informed* interface. All interfaces remain unchanged.

Another example is shown in Figure 3.41. A standard arrangement is shown in panel (a): a SENSOR submits its observation likelihoods to a NODE through the *Fusing* interface. Sometimes the sensor would benefit from having access to the current belief about the feature it is trying to observe. The extra information available to the NODE may be a sophisticated prediction model and information coming from the rest of the network. The SENSOR can access it by connecting to the *Informed* interface of the NODE as shown in panel (b).

Another example of a custom component is shown in Figure 3.42. A canonical NODE in panel



(a) (b)

Figure 3.42: A custom NODE component with platform pose input.

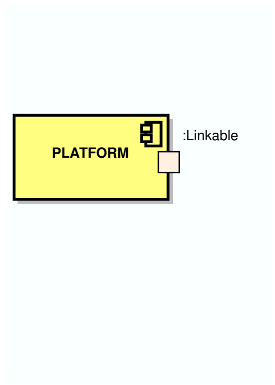
(a) makes decisions about connecting (or disconnecting) to other NODEs without having taken into account the relative positions of the platforms. Panel (b) shows a PROACTIVE NODE which connects to the *Localized* interface of the local FRAME to access its current pose. Each NODE is aware of its current location and can communicate this information to linked NODEs as part of the topology management protocol.

3.7.3 Super-components

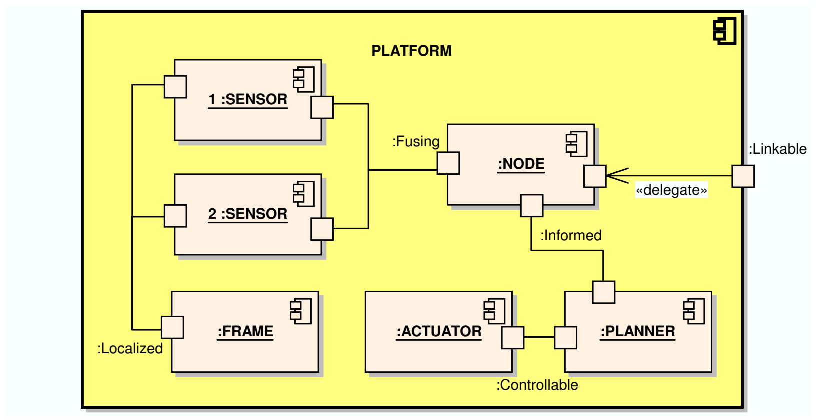
Components may be assembled into larger-scale components in arbitrary hierarchies. In particular, these *super-components* can be built using canonical components as building blocks. A technical reason for such a representation may be that internal and external links use different communication methods.

Figure 3.43 shows a platform-level super-component realized with six canonical components. Out of five interfaces used by the participating canonical components only one, *Linkable*, is exposed to the rest of the system. At this level the entire team could be composed of interacting PLATFORM components. Some functionality has been sacrificed: operator cannot get the beliefs (*Informed* interface is not exposed), external SENSORs cannot submit observations for fusion (*Fusing* interface is not exposed), external actuators cannot be controlled (*Controllable* interface cannot be connected).

Note that the custom VEHICLE component used as an example in Section 3.7.1 can also be built (or simply represented) as a super-component wrapping FRAME and ACTUATOR. In that case, however, it was argued that it may be more efficient to integrate different functions into a single tightly-knit component.



(a)



(b)

Figure 3.43: A custom PLATFORM component composed of six canonical components: (a) specification and (b) a possible realization. Out of five interfaces used internally only one, *Linkable*, is exposed to the outside world.

3.8 Execution Architecture

The Execution architecture traces the mapping of logical components to runtime elements, such as processes and shared libraries. The deployment view shows the mapping of physical components onto the nodes of the physical system. The source code view explains how the software implementing the system is organized. At the architectural level, three items are addressed: execution, deployment, and source code organization.

At the execution level, a component-based implementation is not necessarily decentralized. For any distributed system there is a choice of building functionality of the system into the deployed components or embedding it into the component infrastructure [12]. Szyperski et al. refer to this as direct component interaction versus interaction mediated by the component framework [224]. In the latter case, the decision to use decentralized architectural style means the component framework must be decentralized as well. Following the principle of decentralization, this work puts emphasis on building the required functionality into the components themselves. Components which do not rely on the infrastructure contribute to achieving the objective of designing autonomous self-contained platforms.

For maximum flexibility in using a highly modular system, arbitrary deployment must be supported. This imposes certain requirements on the communication library.

Principle of modularity is very important in source code organization. The approach of this work is to eliminate any cross-dependencies between individual components. Each component is a project whose code dependents only on the framework implementation and some common tools.

3.9 Summary

This chapter described the ASN approach to solving the MSN problem by focusing on three aspects of the solution: architecture, algorithms, and implementation. To fulfill the MSN requirements, all three aspects of the solution are carefully designed for compliance.

The focus of this chapter was on the ASN architecture. Architectural design was performed at multiple levels, starting with the Meta-architecture which established the three guiding design principles: decentralization, modularity, and locality of interactions. The Algorithmic and Execution architectures ensure that the Meta-architecture principles are adhered to on the algorithmic and implementation levels respectively.

The Conceptual and Logical architectures partitioned the system into a set of manageable elements. The Logical architecture, in particular, defined a detailed architecture specification, including canonical component definition and inter-component service interfaces. The six representative component types were described in detail and used in the implementation of experimental systems in this thesis. Several examples of custom components were also given.

This chapter described the high level organization of the system. The remaining chapters will fill in the details of the two remaining aspects of the framework: algorithms and implementation. Algorithmic details of the two main functional groups, information fusion and decision making, are described in Chapters 4 and 5 respectively. Discussion of the ASN implementation will continue in Chapter 6.

Chapter 4

Information Fusion Algorithm

4.1 Introduction

This chapter focuses on the task of information fusion. Two developments from Chapter 3 serve as departure points: the overall algorithmic architecture presented in Section 3.3 and the logical breakdown of the system into canonical components described in Section 3.6. Using these as a foundation, this chapter describes the information flow and detailed information fusion algorithms used in ASN.

Section 4.2 presents a general framework for decentralized Bayesian information fusion. The framework is an evolution of earlier efforts in Decentralized Data Fusion (DDF) and a brief historical reference is provided. Continuing with the top-down description approach, the information flow is described on four, successively more detailed, levels: between components, between NODEs between the internal parts of a single NODE, and inside the NODE parts. The last level of description is sufficiently detailed to identify a small set of information processing operations, termed the DDF *primitives*, which constitute the building blocks of the ASN information fusion algorithm. This will be called the Bayesian Decentralized Data Fusion (BDDF) algorithm.

The DDF primitives are defined for the general Bayesian case but, in practice, a probability density function must be represented in one of several ways. The rest of the chapter shows how the BDDF algorithm can be adapted to a chosen *pdf* representation. All representation-specific algorithms are well known and are described in detail elsewhere. A summary of most algorithms is given in the Appendix and references to the original work is provided throughout.

Section 4.3 specializes the BDDF algorithm and the DDF primitives to the case of Gaussian point features. This representation is most commonly used in the context of DDF due to the storage and algorithmic efficiency which it allows. A group of well known algorithms using Gaussian features are presented as special cases of the general Bayesian form of DDF. An application to tracking point targets is discussed, using simulated and experimental results.

Despite their computational advantages, parameterized probability distributions, such as Gaussians, are often inadequate because of the limiting assumptions they introduce. The most common method of representing general distributions is by discrete, piecewise constant functions. Section 4.4 presents the BDDF algorithm using this type of representation. Several applications are considered in this context: Bayesian non-linear target tracking, feature identification, and area mapping using the Certainty Grid representation.

4.2 Bayesian Decentralized Data Fusion Algorithm

This section describes the main information fusion algorithm of ASN called Bayesian Decentralized Data[1] Fusion (BDDF) algorithm. The algorithm is based on the work started at the robotics group at Oxford University in the late 1980s. The DDF algorithm was originally developed for Gaussian point features and was based on the use of the information form of the Kalman filter. A historical account of DDF can be found in [163].

Initially, the DDF algorithm was applied to a network of cameras tracking multiple targets which included people and robots [180, 182, 183]. The algorithms required a fully connected topology, meaning that each platform communicated with every other platform, which is impractical in large networks. The work of Grime [67, 68] made explicit the information form of the Kalman filter and used this to solve the problem of communicating in tree-connected networks. This work employed a structure called the *Channel Filter* to maintain an estimate of common information between connected fusion nodes. This result was used to demonstrate the operation of decentralized data fusion systems on a model process control plant comprising over 150 distributed sensors. More recently, a practical

[1]In this work, the term *information fusion* is generally preferred to *data fusion*. The latter is retained in the name of the algorithm for historical reasons.

DDF system was developed as part of the ANSER project [163, 221]. A team of up to three custom-built UAV's performed localization of ground features represented as points with Gaussian position uncertainty. Team makeup was static and communication topology was pre-configured.

The general decentralized approach including Channel Filters follows the ideas of Grime [67, 68], Manyika [144, 145], and Nettleton [163, 221]. This work reformulates the approach in general Bayesian terms and uses the existing Gaussian algorithms as a special case.

4.2.1 Inter-Component Information Flow

The information flow between components is considered first. Figure 4.1 shows four canonical components participating in the task of information fusion. The four interfaces appearing in this diagram were described in detail in Chapter 3. Both human and robotic *belief sources* and *sinks* are shown, highlighting the fact that the information flow between components does not depend on whether a human is involved.

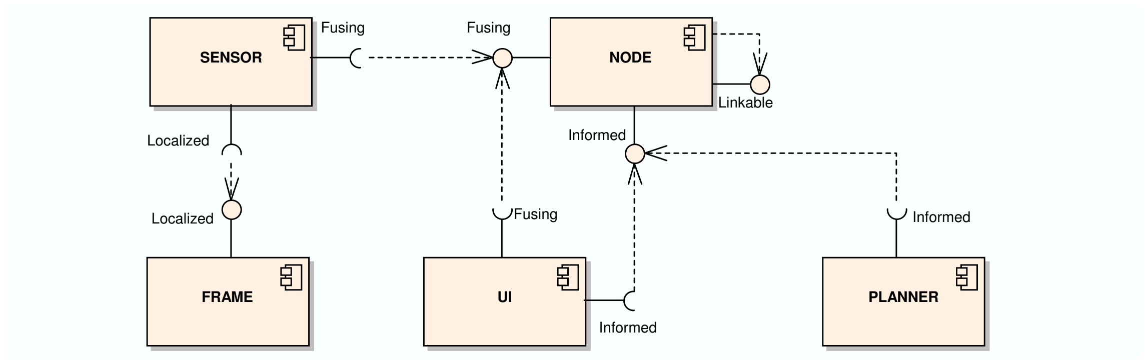


Figure 4.1: Structural diagram of interactions between canonical components used in information fusion. Two types of *belief sources* are shown: robotic SENSOR and human UI, as well as two types of *belief sinks*: robotic PLANNER and human UI.

The following descriptive example illustrates the use of interfaces in implementing the task of information fusion.

Example 4.1

Component types and interfaces participating in information fusion are illustrated using a network of six components shown in Figure 4.2. The diagram shows NODE *1 connected to*

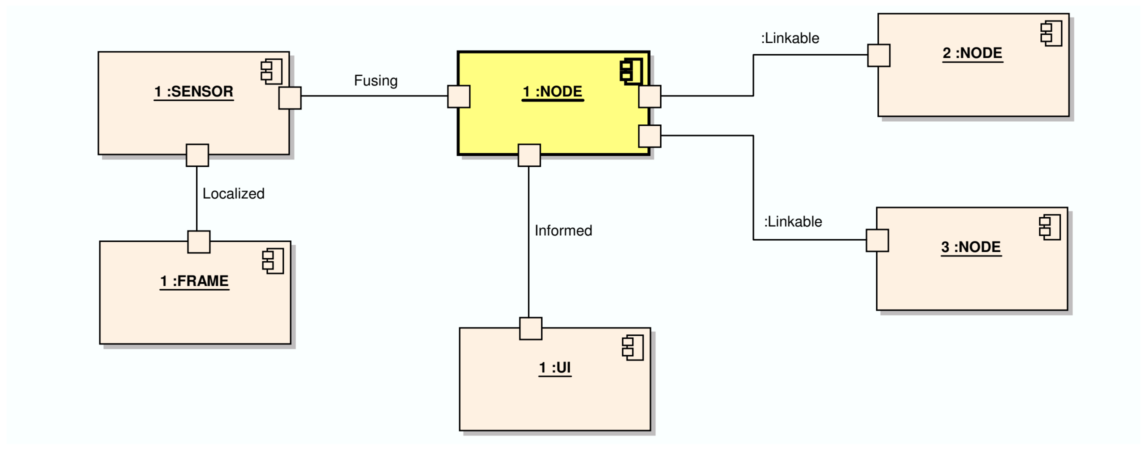


Figure 4.2: A collaboration scenario illustrating the task of information fusion. It involves instances of SENSOR, NODE, FRAME, and UI components. The scenario description focuses on the interactions of NODE 1 (highlighted).

two other NODE*s, 2 and 3.* NODE *1 provides fusing services to a single local* SENSOR *1. The* SENSOR *receives localization information from* FRAME *1. An instance of* UI *receives environment information from* NODE *1.*

The operation of the network is shown in Figure 4.3 as a Sequence diagram.[2] *The diagram shows messages passed between the components participating in the scenario. An event in the environment leads to a measurement* z_1^k *received by* SENSOR *1. A synchronous call to* FRAME *1 through the* Localized *interface returns the current platform pose needed to process the measurement. The* SENSOR *calculates the observation likelihood* b_{l1}^k *and submits it to* NODE *1 through the* Fusing *interface.* NODE *1 fuses the observation with the belief it currently holds.*

At some later time, NODE *3 sends its current belief* b_{x3}^k *through the* Linkable *interface and, independently,* NODE *2 sends its belief* b_{x2}^k*. Both are processed by* NODE *1. Eventually, triggered for example by an internal clock,* NODE *1 sends out its updated belief* b_{x1}^k *to both* NODE*s 2 and 3, which fuse it with their local beliefs. It also updates* UI *1 through the* Informed *interface.*

[2]`UML` A Sequence diagram is a structured representation of behavior as a series of sequential steps over time. It is used to depict work flow, message passing and how elements in general cooperate over time to achieve a result. [223]

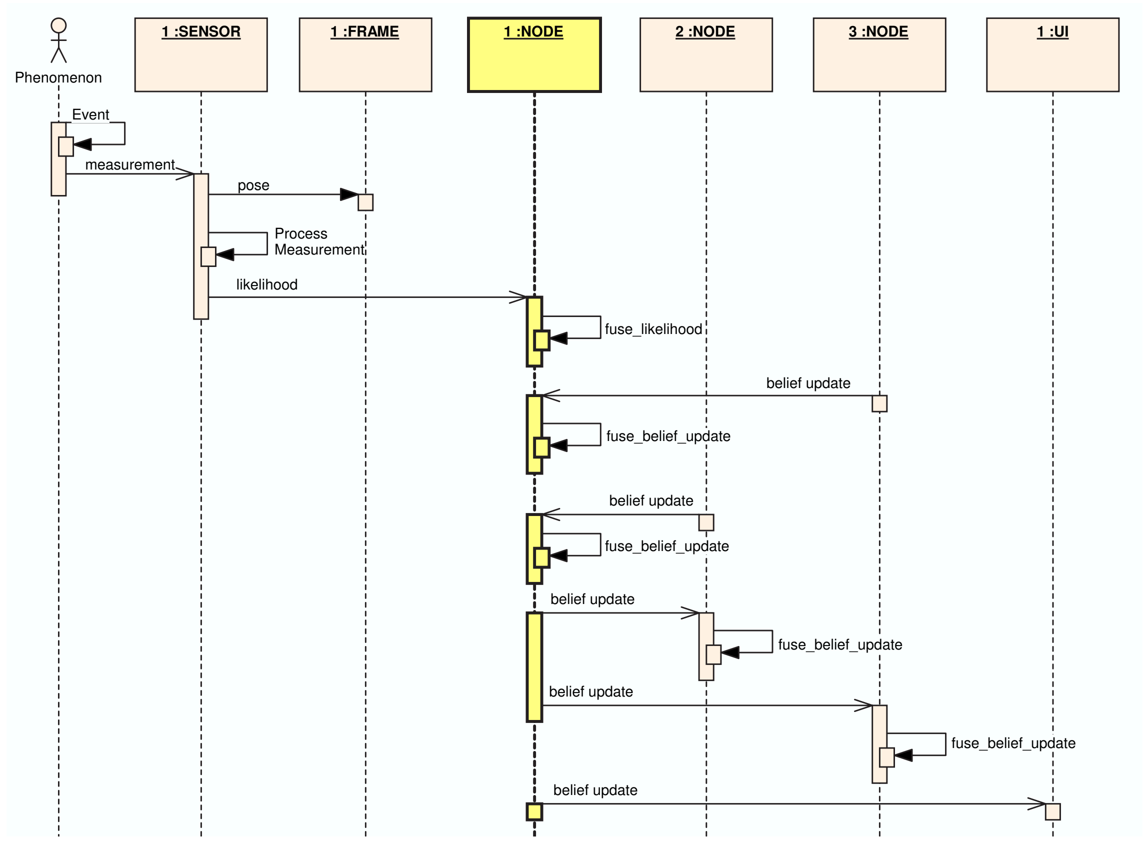


Figure 4.3: Sequence diagram of inter-component communication in the information fusion collaboration scenario. Communication exchanges are shown from the point of view of NODE 1 (highlighted).

From the Example above it is clear that the NODE component is at the center of the information fusion process. For this reason, NODE-to-NODE interactions, the internal NODE structure, and its algorithms are the focus of this chapter. Several internal subsystems of the NODE were identified in Chapter 3. The white-box diagram of internal NODE structure is repeated in Figure 4.4 for convenience.

4.2.2 Inter-Node Topology Management

Belief fusion in ASN is accomplished through pair-wise synchronization of beliefs using peer-to-peer connections between NODEs (see Section 3.3). An important aspect of the fusion process is the global topology formed by local NODE links. The Topology Manager shown in Figure 4.4 is an internal NODE part responsible for forming the network of connec-

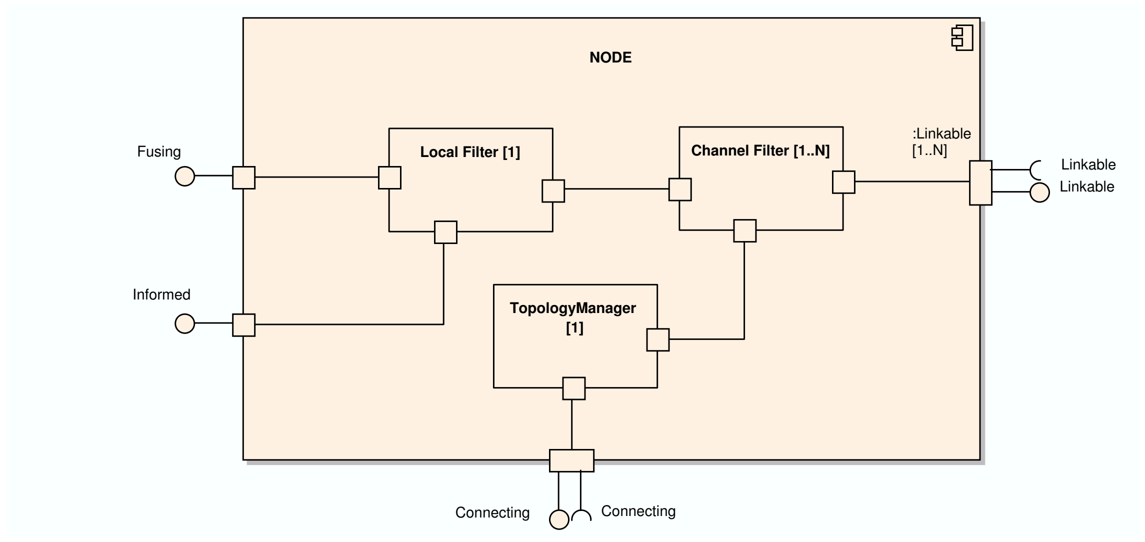


Figure 4.4: A possible realization of the NODE component. Repeated after Figure 3.26.

tions between NODEs. Similar to other network configuration decisions, NODE connections are established in a decentralized fashion. Local connection decisions made by individual Topology Managers shape the global topology of inter-NODE links.

Network topology has a profound effect on the belief fusion algorithms. Following Grime [67, 68], three types of network topologies are considered: fully-connected, tree-connected, and general networks. Information exchange in distributed systems performing Bayesian inference is the subject of *Bayesian communication* which is reviewed in Appendix B.2. The main result states that the key to combining local beliefs from two communicating NODEs is in determining what information they have in common.

Finding common information in fully connected networks is trivial because the belief of all NODEs is the same at the end of each communication cycle. However, fully-connected network topologies do not scale well with the number of network components.

Common information in tree-connected networks is maintained by storing the information that was previously communicated through the link, or *channel*, connecting two NODEs. Channel Filters are internal NODE structures introduced specifically to maintain the information common to two NODEs. Several Channel Filters are shown in Figure 4.4, one for each *Linkable* interface.

In general networks, there is no algorithm for optimal fusion of NODE beliefs within the locality constraints of a decentralized sensing network. The proof of this result can be found in [232]. Three solution avenues have been investigated: data tagging, spanning trees, and dynamic determination of common information. Tagging was determined not to be scalable [165]. Routing using decentralized Bellman-Ford algorithm requires a *Manager* node — a central resource with knowledge of global topology [233, 234]. The algorithm has been demonstrated in networks with relatively low rate of change in topology. In large-scale networks in which the topology changes dynamically on a regular basis, sub-optimal methods of determining the common information are most appropriate [165]. To date, the work on suboptimal (conservative) fusion has been limited to Gaussian point features.

Compared to trees, general network topologies are appealing because of the extra robustness introduced by the redundant links between NODEs and the simplicity of arbitrary connections. However, in the general Bayes case, a solution to the problem of combining beliefs with unknown common information is unknown. For this reason, the general-network approach is not considered fully in this thesis. Conversely, the Channel Filter method guarantees optimal use of information but requires an acyclic network topology. It is best suited for situations when the information has low signal to noise ratio and the network topology is quasi-static. On the other hand, when information is abundant but the makeup and configuration of the network is highly fluid, conservative fusion of information is preferred.

An acyclic network requires special algorithms on the part of the Topology Manager. Addition of new NODEs does not jeopardize the tree structure but exit or failure of NODEs may. When a NODE in a non-leaf position in the tree fails, the network is fragmented into two or more sub-trees and a naive attempt to rejoin the sub-trees may lead to cycles ("head catching the tail"). A simple approach which solves this problem is to disconnect all child NODEs below a failed link and reconnect them individually. Besides simplicity, this approach has the positive side effect of re-balancing the network tree. The penalty is in a high level of communication needed for reconnection and loss of information due to suboptimal fusion at each NODE pair immediately after the connection.[3] Note that all reconnection algorithms require some form of conservative fusion.

Even in a perfectly reliable static network, the topology management algorithm has a sig-

[3]A more sophisticated decentralized algorithm for rebuilding a tree network after a NODE failure is proposed in [163]. It has the advantage of minimal information loss as the Channel Filter state inside the sub-trees is preserved. Its disadvantage is in introducing the concept of the root NODE and increasing (at worst doubling) the tree depth after reconnection.

nificant effect on the performance of the system. One important parameter is the maximum degree for the vertices in a tree network. There is a trade-off between the depth of the tree and the load at highly connected vertices. The problem of organization in decentralized systems from the point of view of performance was investigated by Ho [81]. This work considered how to best configure a decentralized network using fusion nodes with a fixed number of communication links. The performance of all fusion nodes was tracked to determine the effect of different topologies on information flow. For example, in a star-connected topology the belief of the central node typically has smaller error at any given time because it receives information in a more timely manner.

The algorithms above are reactive because they consider topology change in response to communication failure. It has been suggested that proactive algorithms are better suited to continuous data delivery systems and systems with low network dynamics [227]. Expressed in terms of ASN components, this idea was illustrated with a custom component in Figure 3.42 which based connection decisions on the relative positions of the communicating platforms. Proactive algorithms for network topology control is an open research topic.

4.2.3 Node Information Flow

In this section the information flow within the NODE component is considered. The communication diagram in Figure 4.5 shows the flow of information between the external ports of NODE i and its internal elements. The four activities shown in the diagram occur asynchronously, which is illustrated by arranging the messages into four groups. The first two activities are triggered by external events, i.e. the arrival of messages at the *Fusing* and *Informed* interfaces. The other two are triggered internally, by timers or other events. The following listing describes each of the message groups.

1. *Local Update.* When an observation likelihood b^k_{xl} arrives from one of the local SENSORS at the *Fusing* interface, it is passed to the Local Filter where it is fused with local belief. Local belief updates are performed as soon as local observations arrive.

2. *Incoming Channel Update.* When a channel update message arrives from a linked NODE j through the *Linkable* interface, it is first processed internally in the corresponding Channel Filter. Update messages from different channels arrive asynchronously and, if all update messages are fused at the Local Filter on arrival, it may lead

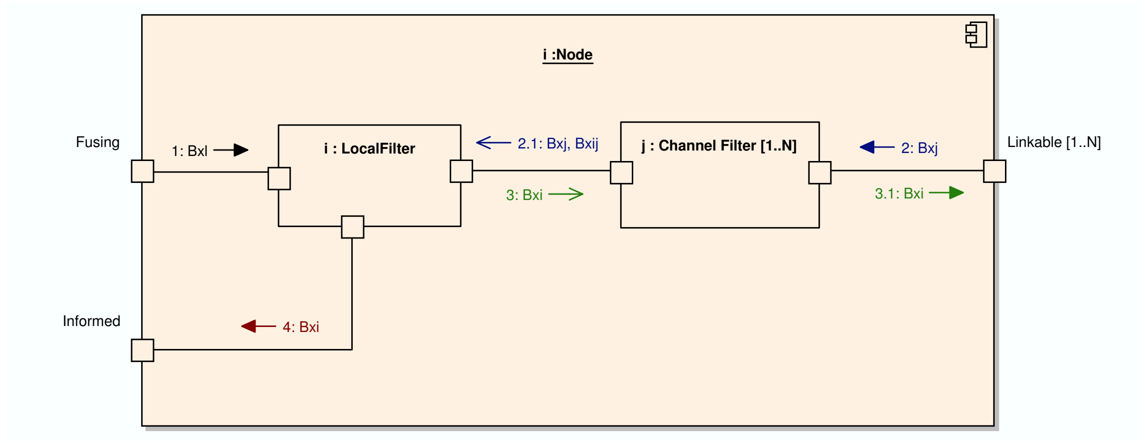


Figure 4.5: Information flow to and from a NODE. Events corresponding to the four groups of messages occur asynchronously. The topology manager module does not participate in belief maintenance and, for this reason, is not shown.

to multiple near-simultaneous processing involving a potentially expensive prediction step. To avoid this, the external beliefs b_{xj}^k contained in channel messages are predicted forward to a specified time in the future (called the *time horizon*). At that time, updates from all channels are incorporated into the local belief simultaneously. For this to happen, the external belief b_{xj}^k and the common belief between the two NODES $b_{xi\cap j}^k$ are passed to the Local Filter for fusion with the local belief b_{xi}^k.

3. *Outgoing Channel Update.* When an internal event triggers synchronization between between NODEs, the current local estimate b_{xi}^k is passed to the Channel Filter which sends an update message to the linked NODE j.

4. *Sink Update.* When an internal event triggers a state update of the connected *belief sinks*, the current local estimate b_{xi}^k is sent out through the *Informed* interface.

4.2.4 Node Algorithms

This section provides details on the information flow shown in Figure 4.5 by describing the algorithmic steps inside the Local and Channel Filters. The information processing steps described in this section constitute the BDDF algorithm in its general Bayesian form.

Local Update. The Local Filter receives observations from one or more local sensors through the *Fusing* interface. The Local Filter contains an array of Bayesian filters repre-

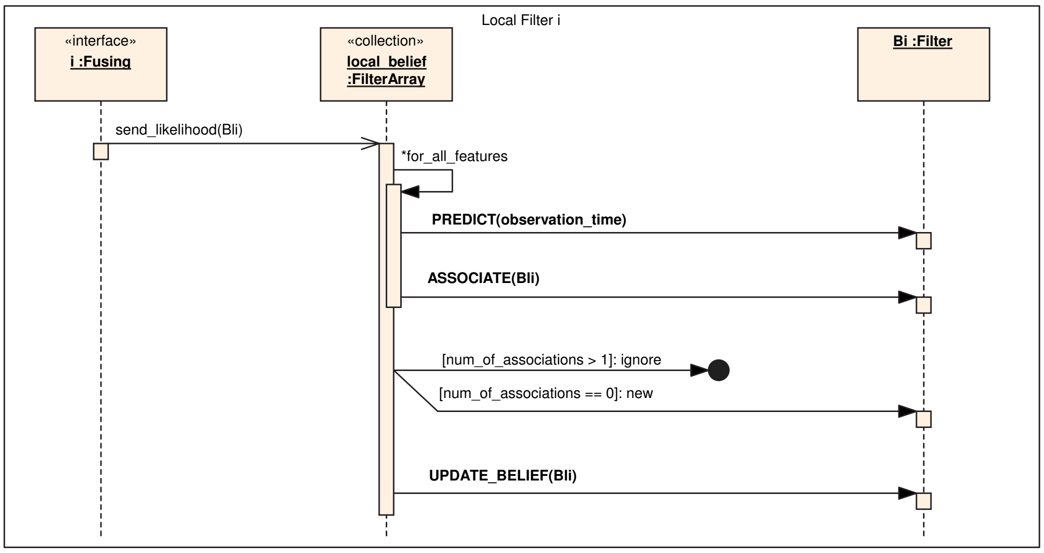


Figure 4.6: Information flow in the Local Filter on arrival of a local observation. (Message 1 in Figure 4.5.)

senting individual features of the environment. Different feature types may be stored side by side and filters may use different *pdf* representations.

A sequence diagram in Figure 4.6 shows the steps inside the Local Filter. Observations arrive at the *Fusing* interface and are passed to the Local Filter. Inside the Local Filter, all features of the matching type are first predicted forward to the observation time. Data association is performed by matching likelihoods with the local beliefs. Multiply-associated observations are ignored. If no matching existing feature is found, a new one is initialized. With a valid association, the Local Filter fuses the observation and the predicted belief.

Incoming Channel Update. Channel Filters maintain the state of the connection to other NODEs. The Channel Filter is structured in the same way as the Local Filter: it contains an array of Bayesian filters representing individual features of the environment.

Processing of the channel update message begins in the Channel Filter as shown in Figure 4.7. When a channel update message is received from NODE j, all estimates of common information are predicted to the next time horizon, and track-to-track data association is performed. If there is no match to an existing feature, a new one is initialized. Multiply associated observations are ignored. If a positive match is found, the remote estimate is predicted to the time horizon and placed into the Local Filter's incoming buffer.

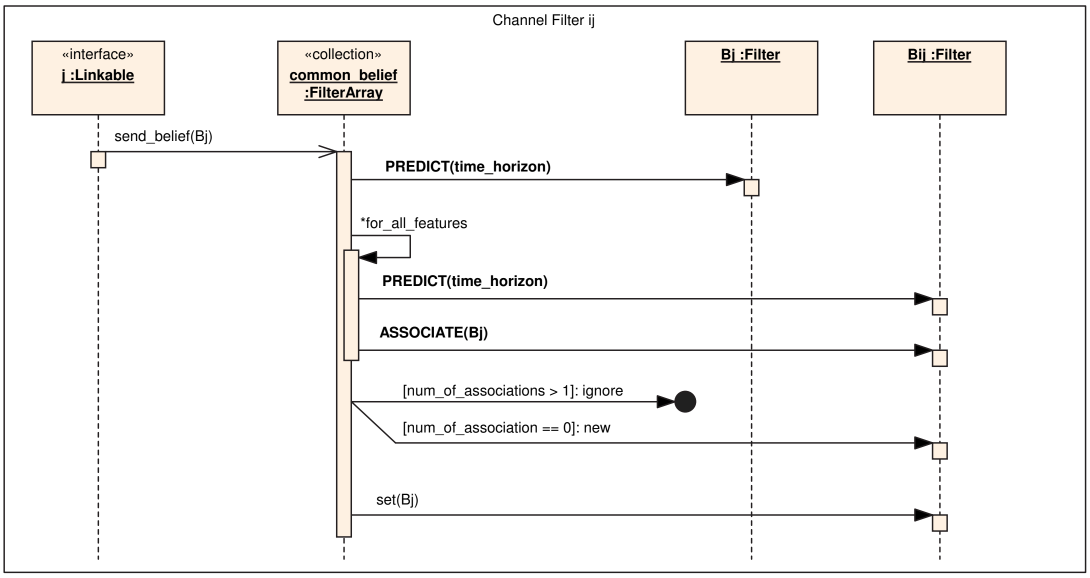


Figure 4.7: Information flow in the Channel Filter on arrival of a channel update message. (Message 2 in Figure 4.5.)

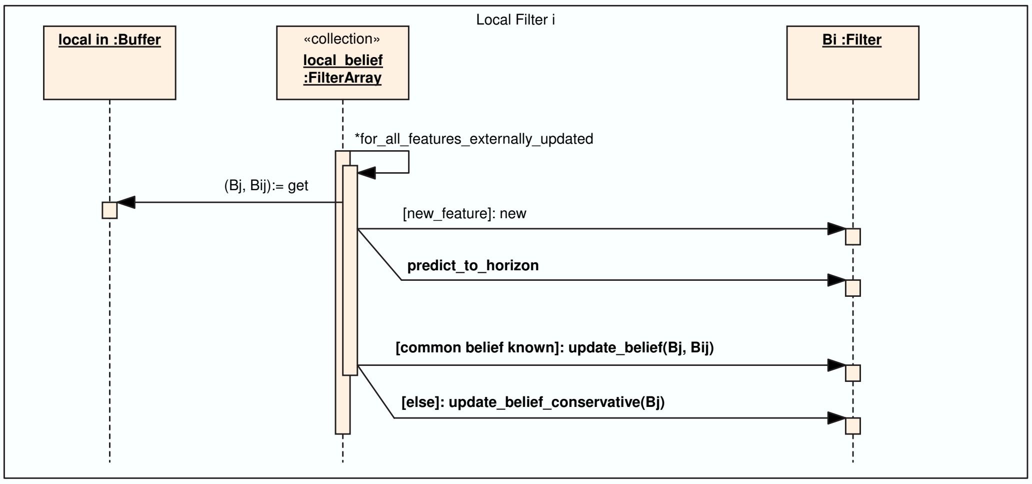


Figure 4.8: Information flow in the Local Filter on arrival of a channel update message. (Message 2.1 in Figure 4.5.)

At the time horizon, the channel update is processed inside the Local Filter as shown in Figure 4.8. The fusion procedure depends on whether the common belief contained between the two communicating NODEs is known. If common belief is not known, then the two beliefs are fused conservatively. This happens when two NODEs initially come in contact and it is not known whether their beliefs are correlated. If the common belief is known, then the local and the external beliefs are fused using the optimal belief update equation.

Outgoing Channel Update. When NODE i has information unknown to NODE j, a decision must be made on whether to send a channel update or not. Simple options were discussed in Section 3.3.5.

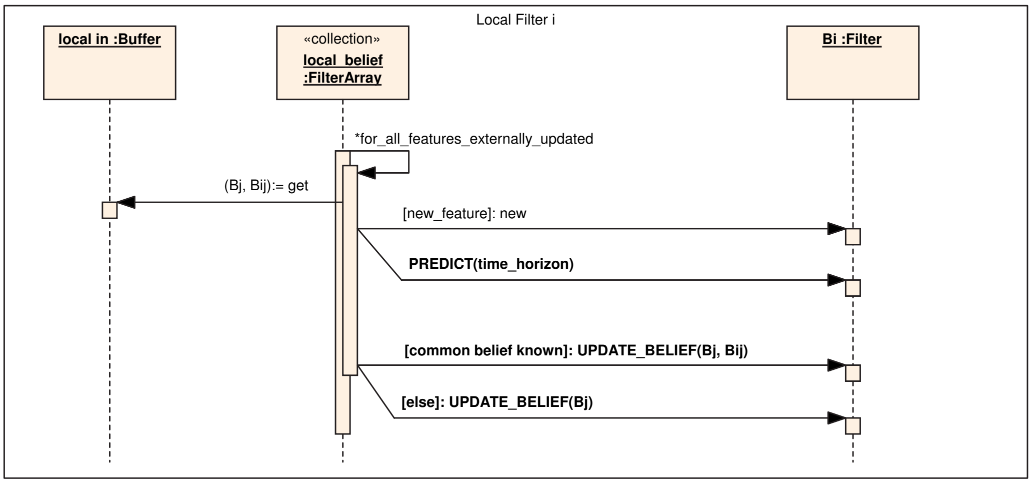


Figure 4.9: Information flow in the Channel Filter on local update. (Message 3 in Figure 4.5.)

Figure 4.9 shows the flow of outgoing information from the Local Filter to the channels. The Channel Filter finds all features which have new information compared to the linked NODE j. The common belief $b^k_{xi\cap j}$ is set to current local belief b^k_{xi} and the update is sent through the *Linkable* interface. In the event that the channel becomes blocked or disconnected, the Channel Filter is updated to the latest belief and then waits for the next available communication opportunity.

Thus, the common belief $b^k_{xi\cap j}$ is updated in two cases: when NODE i receives an update from NODE j (Figure 4.7) and when NODE i sends an update to NODE j (Figure 4.9).

Sink Update. State update through the *Informed* interface is simply a matter of sending the posterior or some part of it to the subscribed *belief sinks*. Figure 4.10 shows the flow of outgoing information from the Local Filter to a *belief sink*. A decision rule similar to the one for channel update can be applied.

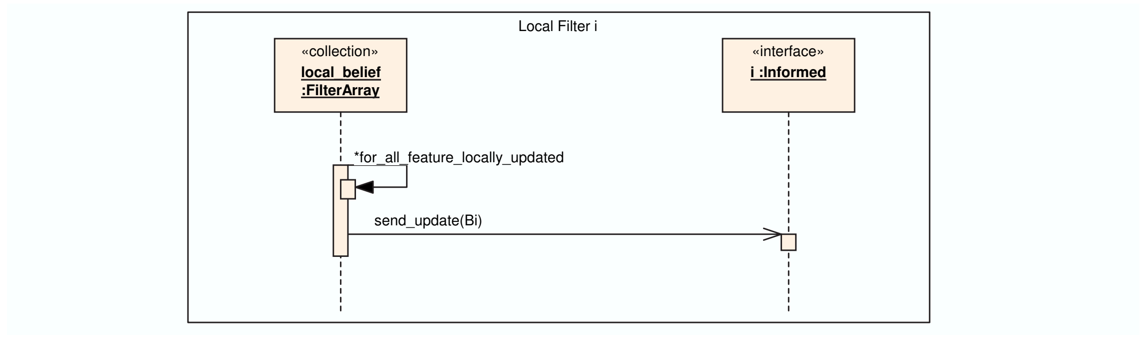


Figure 4.10: Information flow in the Local Filter on sink update. (Message 4 in Figure 4.5.)

4.2.5 Decentralized Data Fusion Primitives

A closer examination of the sequence diagrams in Figures 4.6–4.9 reveals that, regardless of the implementation of the probabilistic filter and the belief representation it uses, the BDDF algorithm can be expressed in terms of three basic operations:

1. Prediction of feature state
2. Belief update
3. Data association

These three basic operations will be referred to as *DDF primitives.* In the diagrams, these three operations are shown in all capitals.

Prediction of Feature State. Also known as *motion update*, this step is used to obtain a belief on the state at the current time based on all past information. The belief prediction equation was defined as part of the reference model in Equation 2.10 and is repeated here for convenience

$$b_{xi}^{k} = G'(b_{xi}^{k-1}). \tag{4.1}$$

The non-linear Bayes filter[4] is reviewed in Appendix B.1. The belief prediction function G' is given in Equation B.5 and is repeated here (omitting the initial prior for clarity)

$$P(x^k \mid Z_i^{k-1}) = \int P(x^k \mid x^{k-1}) P(x^{k-1} \mid Z_i^{k-1})\, dx^{k-1}. \tag{4.2}$$

[4]The term *filtering* refers to the estimation of the (current) state of a dynamic system [10].

It calculates the probability distribution on the feature state x at time t_k based on the information available up to time t_{k-1} and the state transition function $P(x^k \mid x^{k-1})$.

Belief Update. Also known as *information update*, this step combines beliefs from two NODEs. The joint belief of two NODEs is designated $b^k_{xi\cup j}$, it is based on the union of two information sets. The information common to two NODEs is designated $b^k_{xi\cap j}$, it is based on the intersection of two information sets. The belief update function can then be written as

$$b^k_{xi\cup j} = D(b^k_{xi}, b^k_{xj}, b^k_{xi\cap j}), \tag{4.3}$$

which expresses the joint belief $b^k_{xi\cup j}$ as a function of the individual beliefs b^k_{xi} and b^k_{xj}, and the common belief $b^k_{xi\cap j}$. Three cases are possible with respect to availability of common belief $b^k_{xi\cap j}$.

1. The two beliefs b^k_{xi} and b^k_{xj} are known to be independent when conditioned on the true state of the world. The belief update function for this case is given in Equation B.4 and is repeated here

$$P(x^k \mid Z^k_i) = \frac{P(z^k_i = z \mid x^k)P(x^k \mid Z^{k-1}_i)}{P(z^k_i = z \mid Z^{k-1}_i)}. \tag{4.4}$$

   This belief update function is the recursive form of Bayes theorem and it produces the *posterior* distribution based on the prior $P(x^k \mid Z^{k-1}_i)$ and the observation likelihood $P(z^k_i \mid x^k)$. The same algorithm applies to fusing beliefs with observation likelihoods and to fusing two independently obtained beliefs.

2. The two beliefs are known to be conditionally dependent, the network is acyclic, and the common information $b^k_{xi\cap j}$ is known. If the common belief is known, then the local and the external beliefs are fused using the optimal belief update equation given in Equation B.8 and repeated here

$$P(x^k \mid Z^k_i \cup Z^k_j) = \frac{1}{C}\frac{P(x^k \mid Z^k_i)P(x \mid Z^k_j)}{P(x^k \mid Z^k_i \cap Z^k_j)}. \tag{4.5}$$

   This update equation calculates the joint belief $P(x^k \mid Z^k_i \cup Z^k_j)$, based on the local belief $P(x^k \mid Z^k_i)$, external belief $P(x \mid Z^k_j)$, and the belief common between the two NODEs $P(x^k \mid Z^k_i \cap Z^k_j)$.

3. The two beliefs are known to be conditionally dependent and the common information $b^k_{xi\cap j}$ is unknown or the network is cyclic. It is necessary to perform a conservative fusion of two distributions. A simple way to do this is to keep the more informative one of the two beliefs and discard the other one. Entropy can be a measure of informativeness in this case. Nothing more can be done for a general distribution, however, for some representations it may be possible to design a way of combining two correlated distributions in a consistent fashion (see Section 4.3).

Data Association. Sensor observation typically do not contain an explicit reference to the observed feature of the environment[5]. The problem of assigning measurements to features is known as *data association* and, in practice, it is often the most challenging part of the information gathering process. The fundamental approach is to assess the degree of similarity between an observation and the current belief or between two beliefs. Commonly used algorithms are mentioned in the sections on specific probabilistic representations. For general distributions, distance measures [107] may be useful for quantifying dissimilarity between two distributions, but the topic is beyond the scope of the work.

In summary, the BDDF algorithm is defined in five sequence diagrams in Figures 4.6–4.9 and three primitive operations.

[5] *Anonymous* versus *ID* sensors in [199]

DDF Primitives

Prediction of feature state:

$$b_{xi}^{k+1,k} = G'(b_{xi}^{k})$$

Belief update, with no common information:

$$b_{xi\cup j}^{k} = D(b_{xi}^{k}, b_{xj}^{k}, 0)$$

Belief update, with known common information:

$$b_{xi\cup j}^{k} = D(b_{xi}^{k}, b_{xj}^{k}, b_{xi\cap j}^{k})$$

Belief update, with unknown common information:

$$b_{xi\cup j}^{k} = D(b_{xi}^{k}, b_{xj}^{k}, ?)$$

Data Association:

$$DA(b_{li}^{k}, b_{xi}^{k}) = \begin{cases} 1 & \text{if match} \\ 0 & \text{otherwise} \end{cases} \qquad DA(b_{xi}^{k}, b_{xj}^{k}) = \begin{cases} 1 & \text{if match} \\ 0 & \text{otherwise} \end{cases}$$

The BDDF algorithm is defined for an arbitrary *pdf* but implementation details differ depending on the *pdf* representation. In the rest of the chapter the DDF primitives are derived for specific representations of uncertainty: Gaussian point features, discrete general distributions, and Certainty Grids.

4.3 Gaussian Point Representation

Parametric features with Gaussian uncertainty are widely used in the context of data fusion due to the storage and algorithmic efficiency which they allow. This section describes the algorithms used in fusion of information for Gaussian features and presents them as special cases of DDF primitives. Target tracking is a typical application in this category [10]. An application of BDDF to point target tracking is discussed in Section 4.3.2 using simulations and experimental results.

4.3.1 DDF Primitives

State estimation algorithms for Gaussian point features are well known and are summarized for reference in Appendix C. This section maps these algorithms to the DDF primitives.

Consider the combined vector of platform and the relevant feature states $\mathbf{x} = [\mathbf{x}_{pi}\,\mathbf{x}_f]^T$. When the state uncertainty is represented by a Gaussian distribution, it is fully described by the mean value $\hat{\mathbf{x}}(i \mid j)$ and variance $\mathbf{P}(i \mid j)$ at time t_i, given observations up to time t_j

$$\hat{\mathbf{x}}(i \mid j) \triangleq E\left\{\mathbf{x}(t_i) \mid \mathbf{Z}^j\right\}, \qquad \mathbf{P}(i \mid j) \triangleq E\left\{(\mathbf{x}(t_i) - \hat{\mathbf{x}}(i \mid j))(\mathbf{x}(t_i) - \hat{\mathbf{x}}(i \mid j))^T \mid \mathbf{Z}^j\right\}. \quad (4.6)$$

The *Kalman filter* is a recursive linear estimator which calculates an estimate for a continuous valued state, that evolves over time, on the basis of periodic observations. When a number of strict constraints are met, the Kalman filter is the optimal estimator [9, 150]. The *Information filter* is the "information" or "inverse covariance" form of the Kalman filter. The Information filter is defined in terms of two *information-space* variables, the information vector $\hat{\mathbf{y}}(\cdot \mid \cdot)$ and the information matrix $\mathbf{Y}(\cdot \mid \cdot)$

$$\hat{\mathbf{y}}(i \mid j) \triangleq \mathbf{P}^{-1}(i \mid j)\hat{\mathbf{x}}(i \mid j), \qquad \mathbf{Y}(i \mid j) \triangleq \mathbf{P}^{-1}(i \mid j). \quad (4.7)$$

In addition, the information associated with an observation taken at time t_k is defined as follows

$$\mathbf{i}(k) \triangleq \mathbf{H}^T(k)\mathbf{R}_k^{-1}\mathbf{z}(k), \qquad \mathbf{I}(k) \triangleq \mathbf{H}^T(k)\mathbf{R}_k^{-1}\mathbf{H}_k. \quad (4.8)$$

Detailed derivation of the Information filter can be found in [144]. A summary of the algorithm is presented in Appendix C.2.

The update stage in information form is computationally simpler than the update stage for the Kalman filter, at the cost of increased complexity in the prediction stage. The value of this in decentralized sensing is that estimation occurs locally at each NODE, requiring partition of the estimation equations which are simpler in their information form. The information form of the Kalman filter, while widely known, is not commonly used because the update terms are of dimension the state, whereas in the Kalman filter updates are of dimension the observation. For single sensor estimation problems, this argues for the use of the Kalman filter over the Information filter. However, in multiple sensor problems, the opposite is true.

The prediction and belief update DDF primitives follow naturally from the equations of the Information filter. The belief update employing the Channel Filter can also be written in terms of information variables. Channel algorithms for Gaussian features are described in [67, 68] and more recently in [163, 164].

If the Channel Filter is not initialized or is invalid, then an alternative method of extracting common information must be employed. The Covariance Intersect (CI) algorithm provides a solution to the problem of combining two Gaussian random variables in the case when the correlation between the variables is unknown [105]. The algorithm is based on the geometric interpretation of the covariance or information matrices. The CI algorithm is reviewed in Appendix C.3.

To perform data association for Gaussian features, a simple gating mechanism can be employed which dismisses observations which are considered unlikely based on the current belief. The values of the gate are drawn from Chi-Squared distribution [9]. Data association in the case of Information filter is described in Appendix C.2.

Decentralized Data Fusion algorithm for Gaussian point features can be summarized as follows:

DDF Primitives for Gaussian point features

Prediction:

$$\begin{aligned}\hat{\mathbf{y}}(k \mid k-1) &= [1 - \mathbf{M}_k \mathbf{B}_{wk} \boldsymbol{\Sigma}_k^{-1} \mathbf{B}_{wk}^T] \mathbf{F}_k^{-T} \hat{\mathbf{y}}(k-1 \mid k-1) + \mathbf{Y}(k \mid k-1) \mathbf{B}_{uk} \mathbf{u}^k, \\ \mathbf{Y}(k \mid k-1) &= \mathbf{M}_k - \mathbf{M}_k \mathbf{B}_{wk} \boldsymbol{\Sigma}_k^{-1} \mathbf{B}_{wk}^T \mathbf{M}_k\end{aligned}$$

where $\boldsymbol{\Sigma}_k = \mathbf{B}_{wk}^T \mathbf{M}_k \mathbf{B}_{wk} + \mathbf{Q}_k^{-1}$ and $\mathbf{M}_k = \mathbf{F}_k^{-T} \mathbf{Y}(k-1 \mid k-1) \mathbf{F}_k^{-1}$.

Belief Update, with no common information:

$$\begin{aligned}\hat{\mathbf{y}}_{i \cup j}(k \mid k) &= \hat{\mathbf{y}}_i(k \mid k) + \hat{\mathbf{y}}_j(k \mid k) \\ \mathbf{Y}_{i \cup j}(k \mid k) &= \mathbf{Y}_i(k \mid k) + \mathbf{Y}_j(k \mid k),\end{aligned}$$

where in the case of a single observation $\hat{\mathbf{y}}_j(k \mid k) = \mathbf{i}(k), \;\; \mathbf{Y}_j(k \mid k) = \mathbf{I}(k)$.

Belief Update, with known common information:

$$\begin{aligned}\hat{\mathbf{y}}_{i \cup j}(k \mid k) &= \hat{\mathbf{y}}_i(k \mid k) + \hat{\mathbf{y}}_j(k \mid k) - \hat{\mathbf{y}}_{i \cap j}(k \mid k-1) \\ \mathbf{Y}_{i \cup j}(k \mid k) &= \mathbf{Y}_i(k \mid k) + \mathbf{Y}_j(k \mid k) - \mathbf{Y}_{i \cap j}(k \mid k-1)\end{aligned}$$

Belief Update, with unknown common information:

$$\begin{aligned}\hat{\mathbf{y}}_{i \cup j}(k \mid k) &= \omega \hat{\mathbf{y}}_i(k \mid k) + (1-\omega) \hat{\mathbf{y}}_j(k \mid k) \\ \mathbf{Y}_{i \cup j}(k \mid k) &= \omega \mathbf{Y}_i(k \mid k) + (1-\omega) \mathbf{Y}_j(k \mid k),\end{aligned}$$

where $\omega = \arg\max |\mathbf{Y}_{i \cup j}(k \mid k)|$.

Data Association:

$$\mathbf{v}_2^T(k) \mathbf{B}_2(k) \mathbf{v}_2(k) < \gamma,$$

where $\mathbf{v}_2(k)$ and $\mathbf{B}_2(k)$ are defined in Appendix C.2.

4.3.2 Application to Target Tracking

The term *tracking* refers to the estimation of the state of a moving object based on remote measurements, using one or more sensors at fixed locations or on moving platforms [10]. Tracking is one of the main applications of Kalman filtering techniques [9, 150]. ANSER project is a notable example of decentralized tracking using the Information filter algorithm to track ground targets from multiple aircraft [187, 221]. The feature state in this case represents the location of the target. The following two numeric examples illustrate the approach with a simple two-dimensional problem.

Example 4.2

As an example consider tracking a target in two dimensions. The sensing network used in this and following examples is shown in Figure 4.11(a). The sensing platforms are stationary. The hardware makeup and the software deployment strategy is not specified and is not important in this example. The system consists of two NODE*s, each with a local* SENSOR *and connected to each other through the* Linkable *interface. The information exchange between the components is illustrated in Figure 4.11(b). At time step t_0, the two* NODE*s have the same belief and properly initialized Channel Filters, i.e. both know that the beliefs they hold are common between them.*

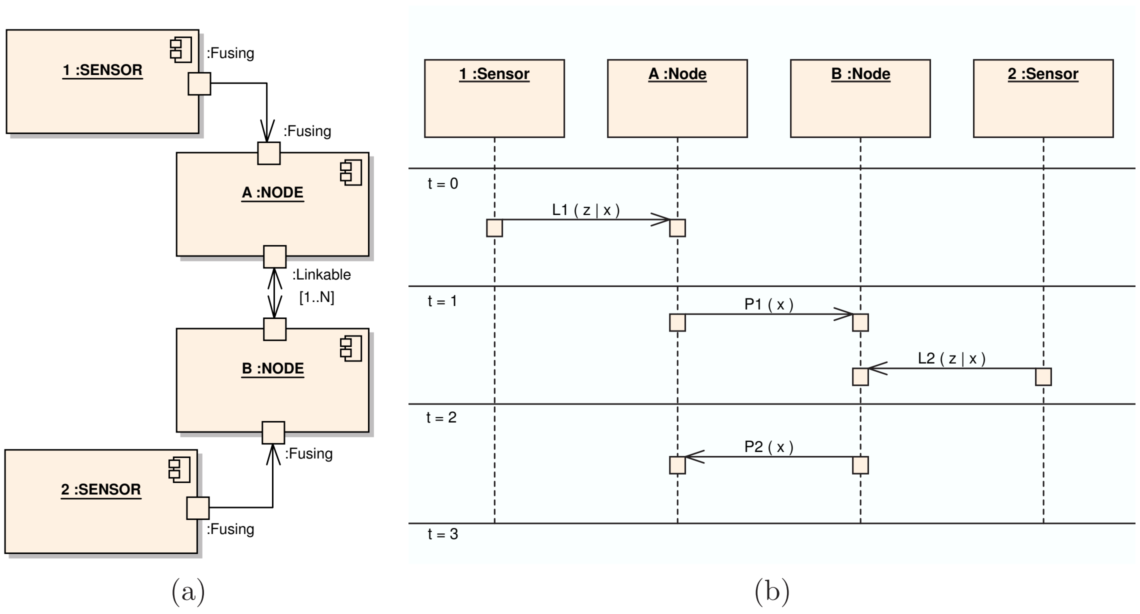


Figure 4.11: Two-NODE information fusion example: (a) component diagram and (b) information flow between the component. Each NODE has a single local SENSOR. The two NODEs are linked into a network through the *Linkable* interface.

The time steps shown in the sequence diagram correspond to the time horizon used by the NODE*s. Each of the two* NODE*s receives a local observation during time horizons t_1 and t_2 respectively. A channel update message is sent from* NODE *A to* NODE *B during step t_1,*

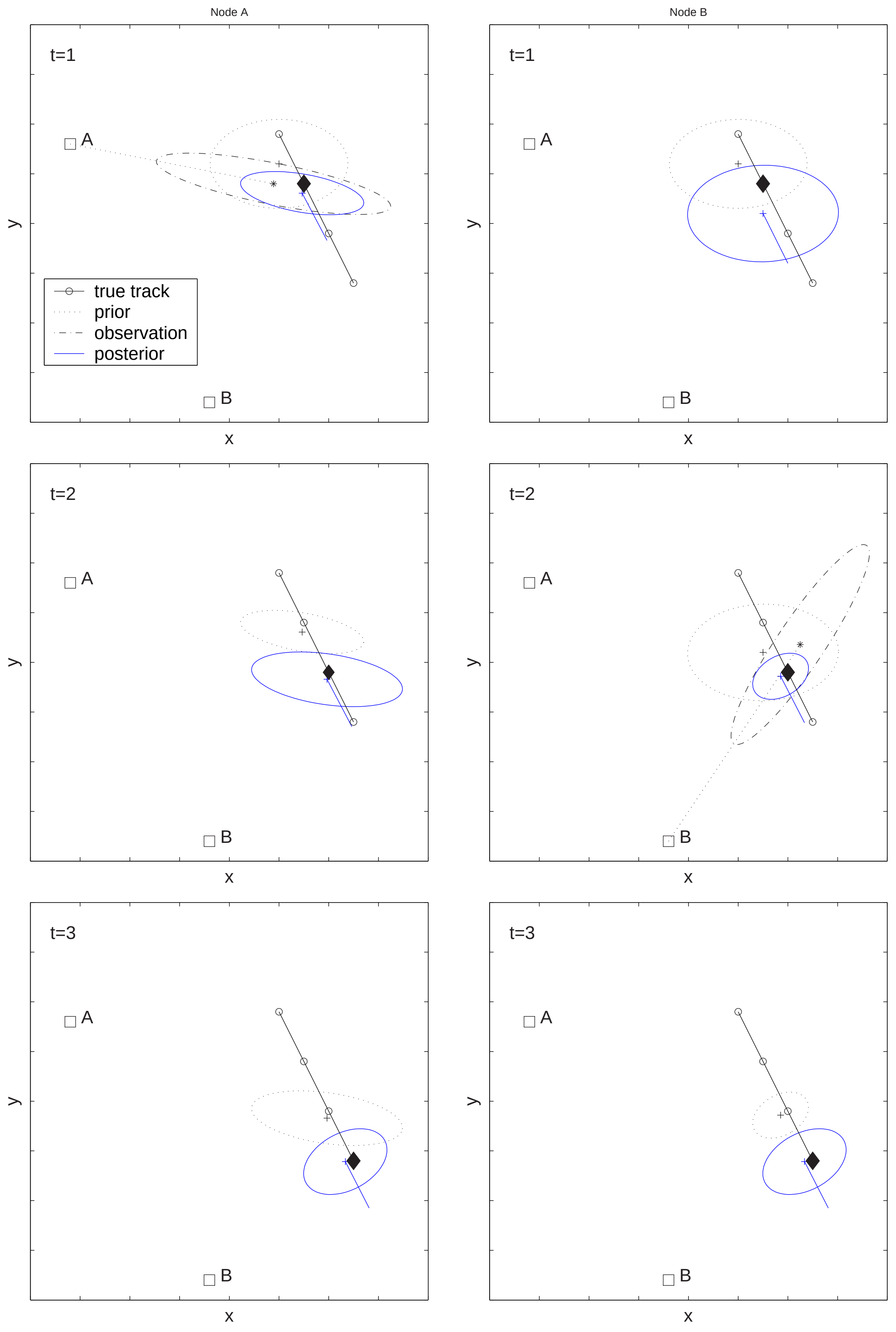


Figure 4.12: Belief evolution in the Information filter example. The panels show platform positions (□), current true target position (♦), and observations ($*$). The target velocity is shown as a line drawn from the mean of the position.

and from NODE *B to A during step t_2.*

The target moves with constant velocity. The observations are of range-bearing type with uncertainty in range much higher than in bearing. The evolution of local beliefs for each NODE *is shown in Figure 4.12. The benefit of information fusion in this case is driven by making observations from two different vantage points. At time step t_3, the belief on the target location is the same for both* NODE*s. This result means that the two* NODE*s have exchanged all the information they have.*

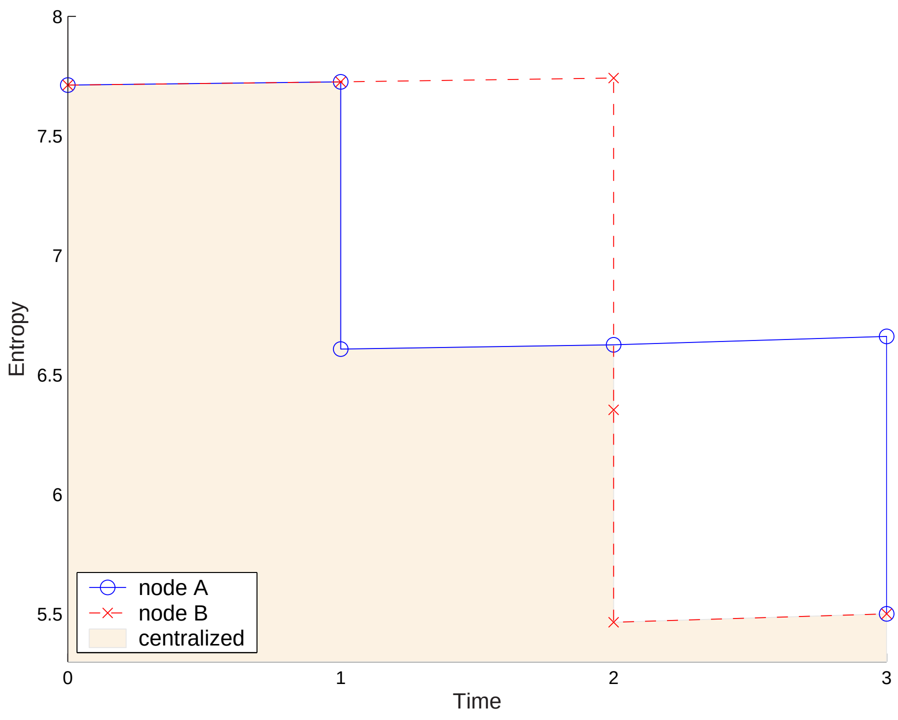


Figure 4.13: Entropy evolution in the Information filter example. Lower entropy corresponds to higher information quality, i.e. information content of the belief increases with each observation. The entropy level of the beliefs at the two NODEs is the same after all information is exchanged.

The same idea is conveyed by Figure 4.13, which plots entropy of the beliefs at the two NODE*s.* Entropy *is a measure of belief quality described Appendix B.3. The final entropy levels for the two beliefs are the same. At all steps in the tracking process the centralized fusion produces optimal results. The result of decentralized algorithm differs from the centralized version only by time delays which demonstrates that, ultimately, no information is lost.*

This is the first in a series of examples demonstrating the BDDF algorithm for different probability representations and different systems. In all examples to follow the basic interaction pattern between components remains unchanged. The message exchange between any two NODE*s is always similar to the one shown in Figure 4.11(b). The beliefs of the linked* NODE*s evolve in near-unison as in Figure 4.12, only one message away from being synchronized. The entropies of the* NODE *beliefs rise during the prediction stage and fall following an observation or a channel update, always one message away from being the same.*

In the previous example, the initial estimates of the NODEs are the same and the NODEs are aware of it. The next example describes information fusion with an uninitialized Channel Filter.

Example 4.3 ———

The network topology and geometry of the problem is the same as in the previous example. The difference is in that when the two NODE*s come in contact, they both have beliefs on the target location. Both have communicated with the network before and it is unknown if the two beliefs are based on independent information sets.*

Because the common information in this case is unknown, a conservative fusion algorithm must be used. Covariance Intersect algorithm described in Appendix C.3 can be used in this situation. Due to the conservative nature of the algorithm, some information is lost during the fusion process. A comparison of the entropy levels in NODE *beliefs is shown in Figure 4.14. The final entropy level for the case of conservative fusion is correspondingly higher than in the case of optimal (centralized or decentralized) fusion.*

For comparison, the entropy of the belief obtained using the "throw away" method is shown. This "fusion" method keeps the more informative belief and discards the other. The entropy level using the CI algorithm is lower, due to the fact that it actually combines the information contained in two beliefs.

This example demonstrated how a conservative fusion algorithm is used in situations when common information between two NODE*s cannot be determined. Note that conservative fusion has to be applied only once, right after the two* NODE*s connect. Once the Channel*

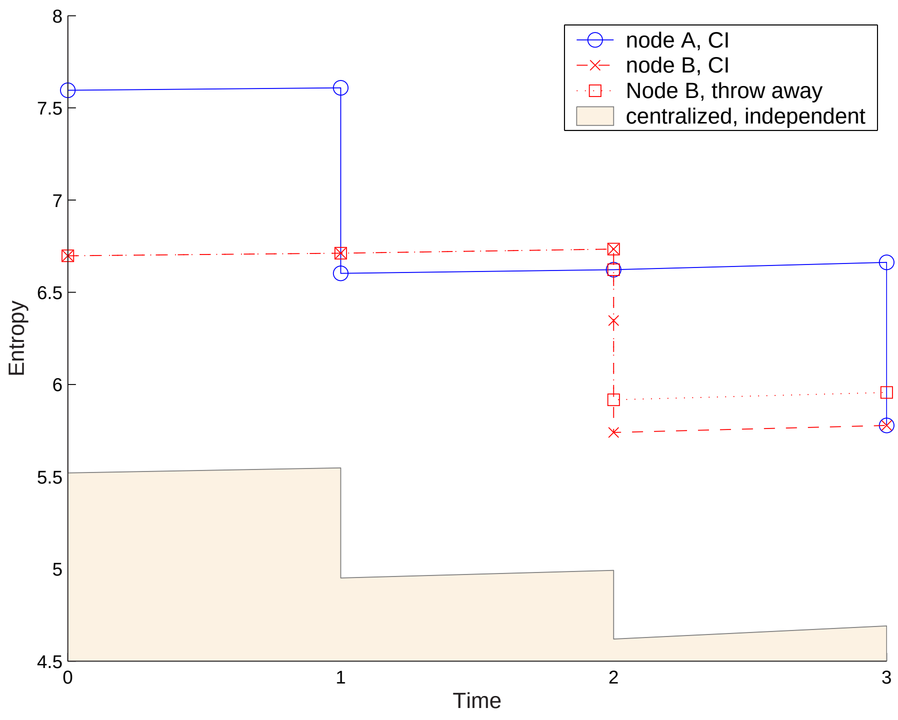


Figure 4.14: Entropy evolution in the Information filter example with Covariance Intersect. The belief obtained as a result of decentralized fusion is more uncertain due to the sub-optimal nature of CI algorithm. The beliefs at the two NODEs are the same at the last step.

Filters on both NODE*s are initialized (and assuming that the topology is acyclic), the fusion process becomes optimal, which is equivalent to getting to the start of Example 4.2.*

4.3.3 Experiments with Point Target Localization

This section describes an experiment demonstrating information fusion on an indoor sensor network.[6] The task in this set of experiments is to estimate locations of stationary point targets. Figure 4.15 shows the overall view of the system. The team consists of two Pioneer robots, a stationary sensor module, and two operator workstations (one stationary and one mobile). The two Pioneers and the stationary sensor are equipped with laser range finders. The experiments ran for up to an hour at a time. The precise component makeup of the

[6]Parts of this section have been previously published in [137, 138].

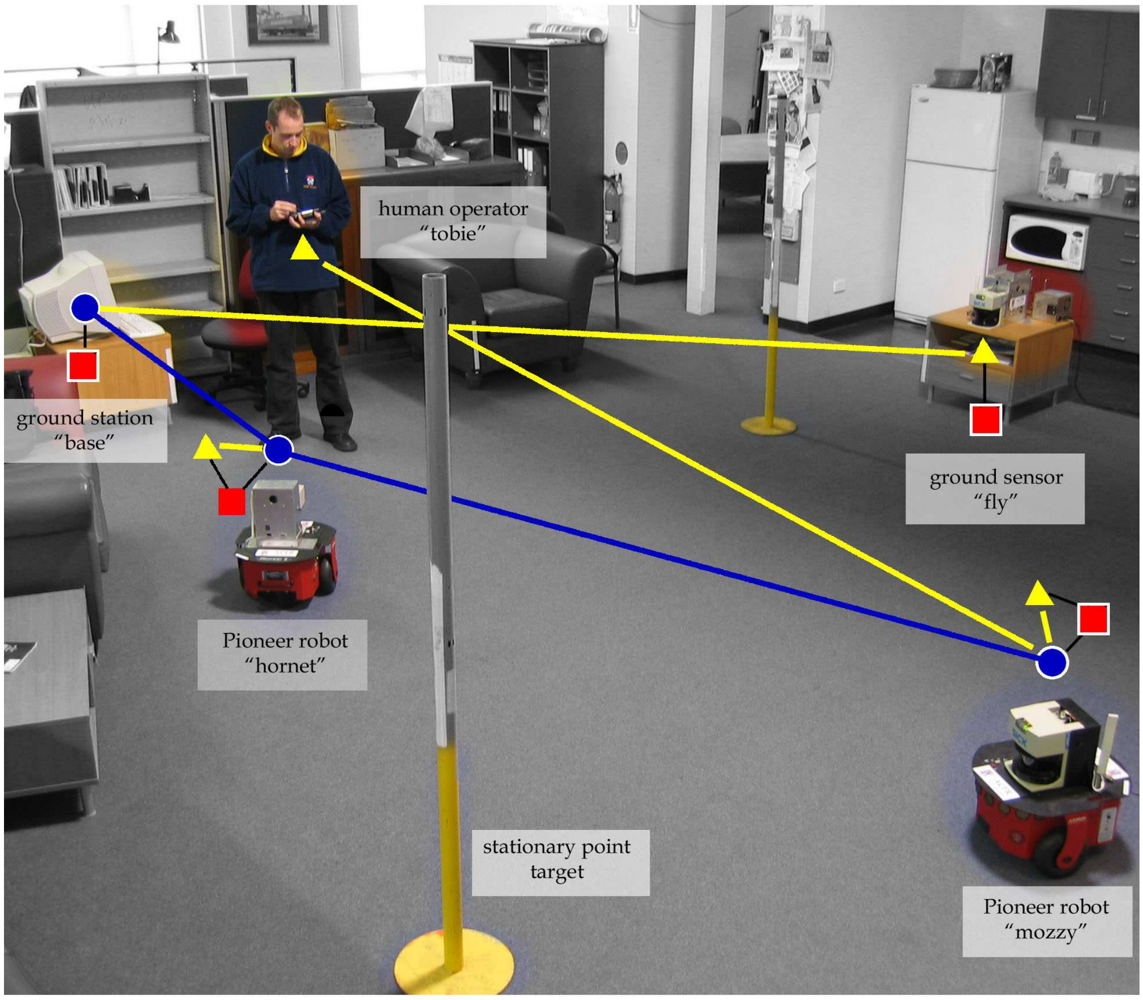


Figure 4.15: Indoor network for localizing stationary point targets. Four platforms are involved: two Pioneer robots *mozzy* and *hornet*, a stationary sensor module *fly*, and a ground station *base*. In addition, there is a mobile human operator *tobie*. The network topology is shown with links connecting four FRAMEs (□), three NODEs (○), and four SENSORs (△).

system and the deployment strategy is not important in this example (it is revisited in Section 6.3.5).

The main objective of this experiment is to demonstrate information exchange between NODEs executing the BDDF algorithm with Gaussian point features. The result, shown in Figure 4.16, is that the beliefs of the two NODEs (and platforms) are synchronized. The screen shots are taken during the system startup and only three platforms are operational at this point.

The screen shot in Figure 4.16(a) shows the view of the world from the point of view of the NODE on platform *hornet*. Feature 1 is located within *hornet's* sensor range. The larger ellipse represents the latest observation, the smaller one represents the current belief about the features location. Feature 2 is outside *hornet's* sensor range and the information about it was sent from the NODE on platform *base*.

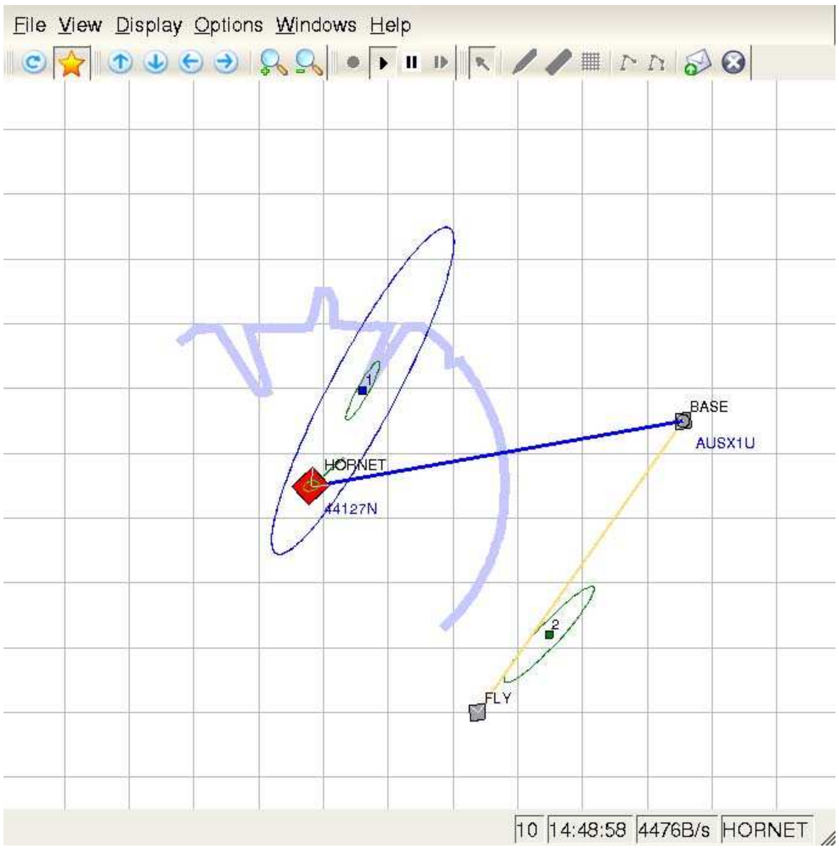
The screen shot in Figure 4.16(b) shows the view of the world from the point of view of the NODE on platform *base*. No current laser scan is shown because *base* does not have any local sensors. Observations arrive from a remote SENSOR executed on platform *fly*. Note that the feature ID's used by the two NODEs are different, $\{1, 2\}$ on *hornet* and $\{0, 1\}$ on *base*. This highlights the need for data association, which in this case is performed using the Information Gate algorithm described in Appendix C.2.

The second objective of this experiment is to demonstrate simultaneous information input from robotic and human sources. Figure 4.17 shows manual input of feature information though the UI. Four platforms are operational at this point and the world view is shown from the point of view of the NODE on platform *mozzy*. Two features, with ID's 0 and 1, were observed by the robotic platforms.

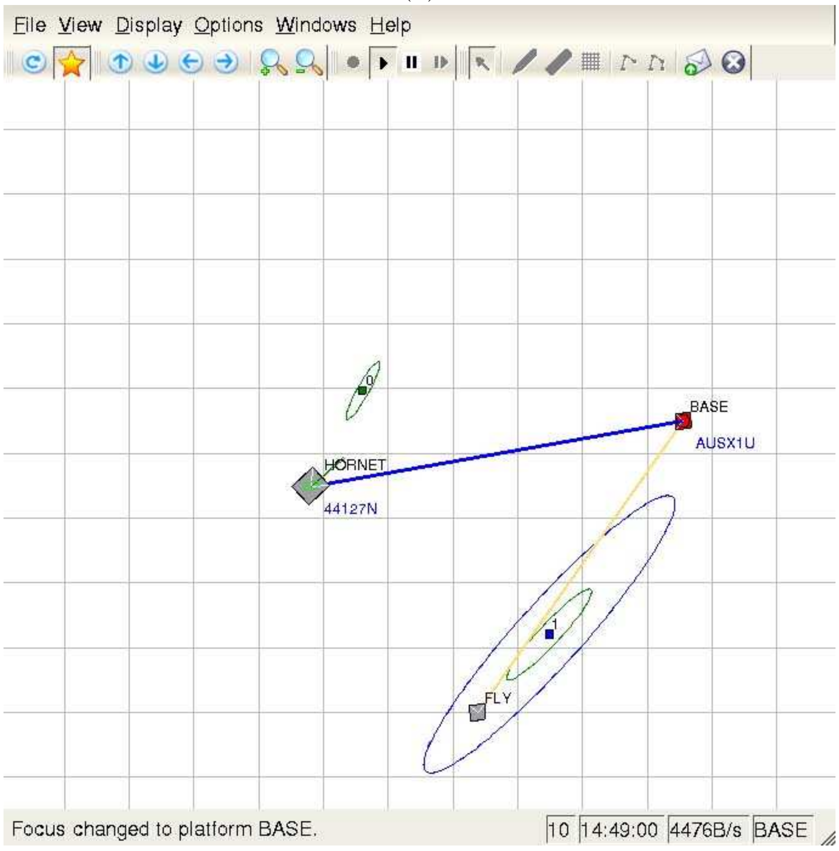
The human operator identifies the likely location of a feature currently unknown to the network which lies outside the sensor range of all platforms. The uncertainty circle[7] drawn by the operator is centered at the most probable location of the new feature, and the radius of the circle is interpreted to be equal to three standard deviations of its position uncertainty. This information is physically entered through a hand-held device as shown in Figure 4.15. The circle properties are sent to the GUI NODE as an observation message and is indistinguishable from messages submitted by regular (robotic) sensors. This observation can be sent to any one (and only one) NODE. After the new feature is initialized as feature 2, all other NODEs are informed about its existence.

Note that in this example the operator actually sent two consecutive observation messages. The second one is more certain (the circle is tighter) and its mean lies to the right from the original observation. The NODE associates the second observation with an existing feature

[7]Uncertainty circle is used instead of a more common ellipse to simplify implementation.


File View Display Options Windows Help
1
2
BASE
AUSX1U
HORNET
44127N
FLY
10 14:48:58 4476B/s HORNET

(a)



Focus changed to platform BASE.

(b)

Figure 4.16: View of the environment from two Gaussian point NODEs. Beliefs of the two NODEs are essentially the same despite the fact that the information about the two features came from two different SENSORs.

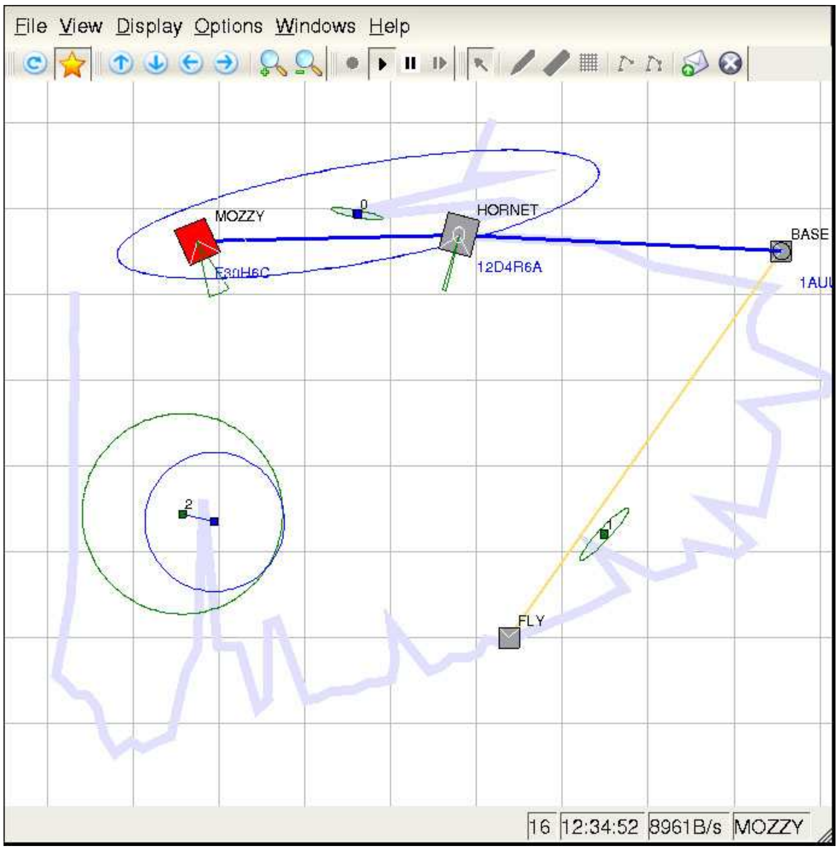


Figure 4.17: Human input of point feature location. Feature 2 lies outside the sensor range of all robotic platforms. Its position is entered by a human operator using two consecutive observations.

and fuses them together. This behavior is identical to what happens when observations arrive from multiple robotic SENSORs and the fact that these particular observations are entered by an operator (or several operators) makes no difference.

The series of experiments described in this section demonstrated application of the Gaussian version of the BDDF algorithm to a small team of mobile indoor robots. Although the scalability metrics were not explicitly measured in these experiments, the results on computational and communication scalability reported in [164, 221] apply because the information fusion algorithms are the same.

4.4 Discrete Grid Representation

Despite their computational advantages, parameterized probability distributions, like Gaussian, are often insufficient for a task because of their limited representation ability. The most common method of representing general distributions in centralized information fusion employs discrete, piecewise constant, representations of the belief. Although, in theory, the resulting tessellation can be of arbitrary shape, the use of a uniform *grid* is the simplest and most common representation.

A key advantage of the discrete grid is that it can represent arbitrary probability distributions. However, the computational, storage, and communication burden required in practice presents a problem. As the complexity grows exponentially with the number of dimensions, grid-based approaches can only be applied to low-dimensional estimation problems. Decentralization of the grid algorithms is straightforward and the DDF primitives for the case of grid representation are listed in Section 4.4.1.

Application of grid-based methods to decentralized target tracking is covered in Section 4.4.2. The state space is divided into a large number of cells, each representing the probability of the target being in that cell. If the number of discrete states is small, the states can be thought off as categories and the approach is sometimes called decentralized identification [16]. This application is addressed in section 4.4.3. A special case of probabilistic identification is the Certainty Grid map which also divides the environment into cells. Each cell characterizes a certain aspect of the environment but the difference is that each cell is independent from all other cells. The algorithm for Decentralized Certainty Grid maps is described in Section 4.4.4.

4.4.1 DDF Primitives

The prediction and update equations for the discrete representation are easily derived from the general Bayes case summarized in Appendix B.1. Prediction is the same as in the general case but integrals are replaced by sums. Belief update is also the same as in the general case with *pdf* multiplication performed element by element.

The combined estimate based on the observations of two NODEs is found through element-by-element multiplication and division of the two individual estimates, and the information they have in common. At present there is no known algorithm to conservatively combine two distributions with unknown amount of common information. It is always safe, however, to simply pick the more informative estimate and discard the other.

DDF Primitives for Discrete Grids

Prediction:

$$P(x^k \mid x^{k-1}, Z^{k-1}, x^0) = \sum_{x^{k-1}} P(x^k \mid x^{k-1}) P(x^{k-1} \mid Z^{k-1}, x^0)$$

Belief Update, with no common information:

$$P(x^k \mid Z^k_{i \cup j}) = P(x^k \mid Z^k_i) P(x^k \mid Z^k_j),$$

where in the case of a single observation $P(x^k \mid Z^k_j) = \frac{1}{C} L(z^k_j \mid x^k)$.

Belief Update, with known common information:

$$P(x^k \mid Z^k_{i \cup j}) = \frac{1}{C} \frac{P(x^k \mid Z^k_i) P(x^k \mid Z^k_j)}{P(x^k \mid Z^k_{i \cap j})}$$

Belief Update, with unknown common information:

$$P(x^k \mid Z^k_{i \cup j}) = \begin{cases} P(x^k \mid Z^k_i) & \text{if } \mathcal{E}n(x^k \mid Z^k_i) < \mathcal{E}n(x^k \mid Z^k_j) \\ P(x^k \mid Z^k_j) & \text{otherwise} \end{cases}$$

Data Association:

same as in the general case.

4.4.2 Application to Target Tracking

In the case of target tracking, each cell of the spatial grid contains the probability of target occupancy. Observation are defined on the same grid, and each cell contains the likelihood of that particular observation being made if the target was at that location.

Continuous states must be discretized to be used with grid representation. The total size of the grid is then the product of the number of discrete cells for each state. With this in mind, storage requirement for feature models and beliefs in grid format can be significant if the entire grid is pre-allocated. As a practical solution, grids with dynamic resolution were implemented in the Nodestar submarine tracking system [218, 219]. This system applied to the case of a single platform and a single target but the same approach can be used in the DIG context.

Example 4.4

Consider again the scenario described in Example 4.2 but with the process and observation models now nonlinear and the noise non-Gaussian. The state transition model G is illustrated in Figure 4.18(a). The velocity of the target is not part of the state to save storage and computation. It is assumed to be a constant vector with mean $[2.5, -5.0]$ *and a covariance* $[3.0, 1.5]$ *in the rotated coordinate system.*

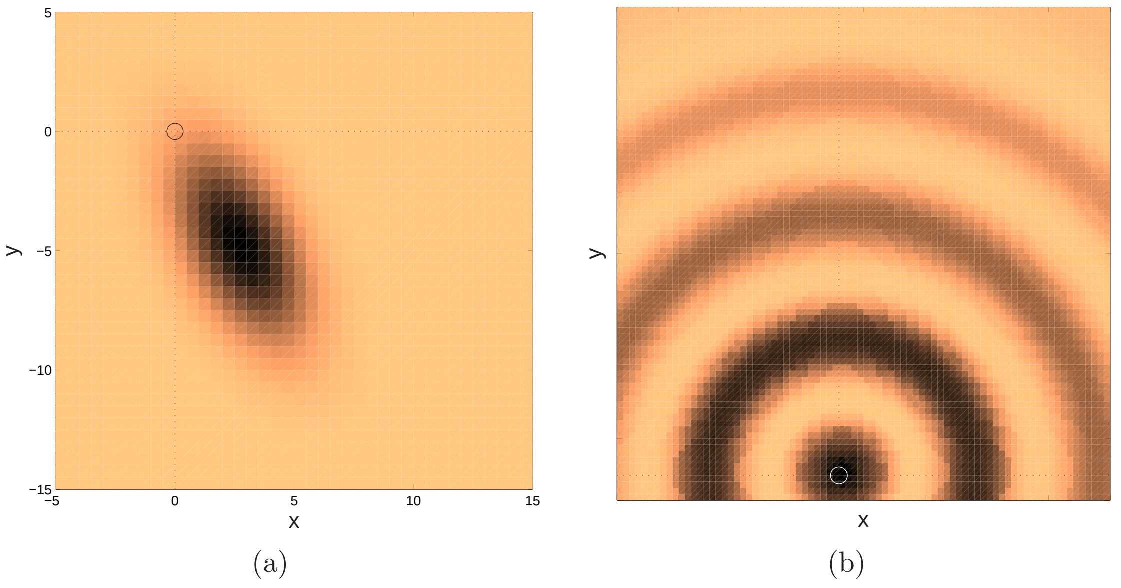


Figure 4.18: Motion and observation models for non-linear filtering: (a) Gaussian motion model and (b) probability of detection for an acoustic sensor [219]. Darker colors correspond to higher probability.

Both sensor platforms are stationary. Platform A is equipped with a range-bearing sensor. Its observation uncertainty is exponential in range and Gaussian in bearing. Platform B is equipped with an acoustic sensor described in [219] (p.47). The probability of target detection is shown in Figure 4.18(b). The range-dependent probability of detection is combined with Gaussian bearing likelihood function to obtain the full observation likelihood function.

Figure 4.19 shows the estimate of target position available at NODEs *A and B over time.*

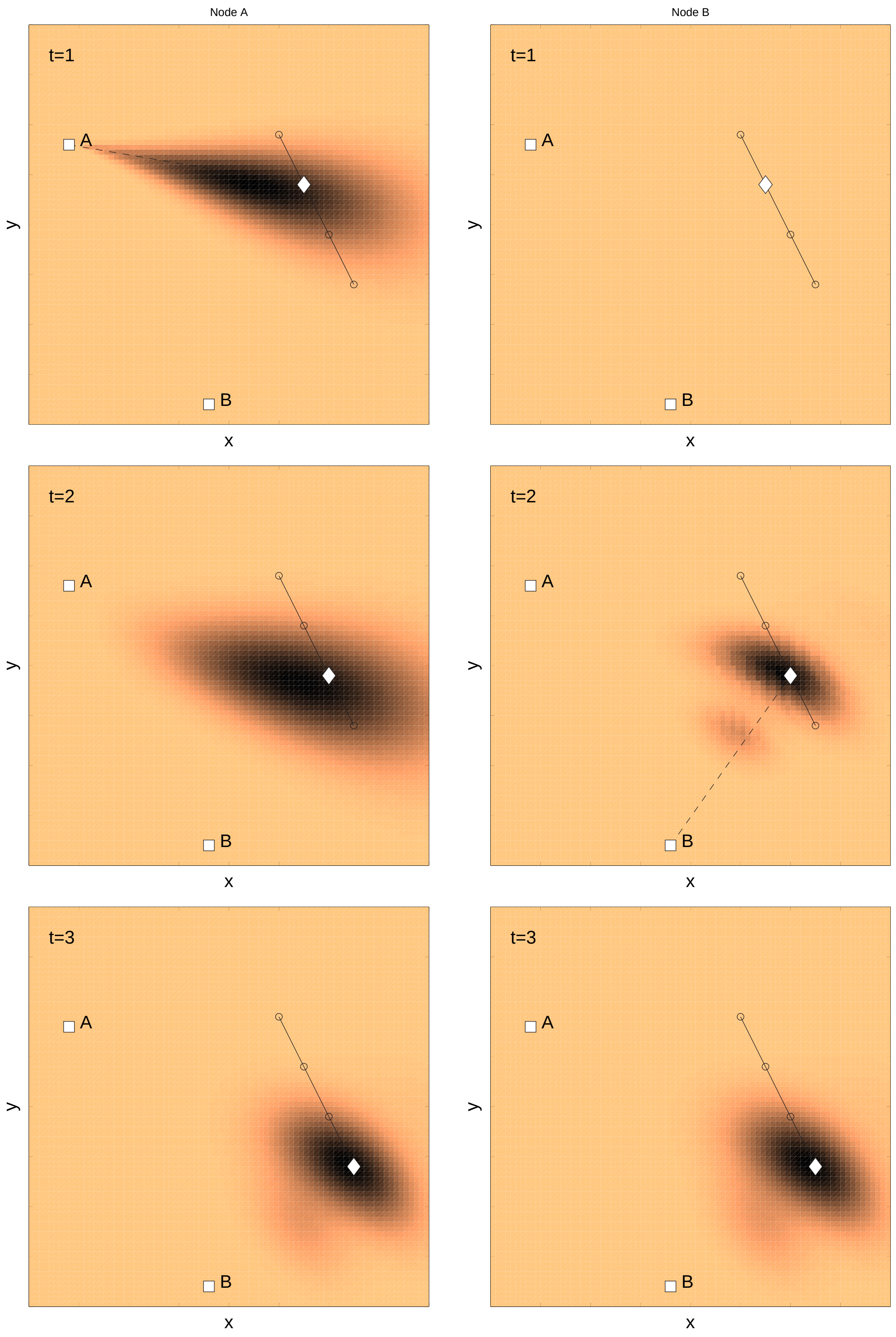


Figure 4.19: Example of decentralized target tracking using nonlinear filtering techniques. The panels show platform positions (□) and current true target position (◊). Darker colors correspond to higher probability.

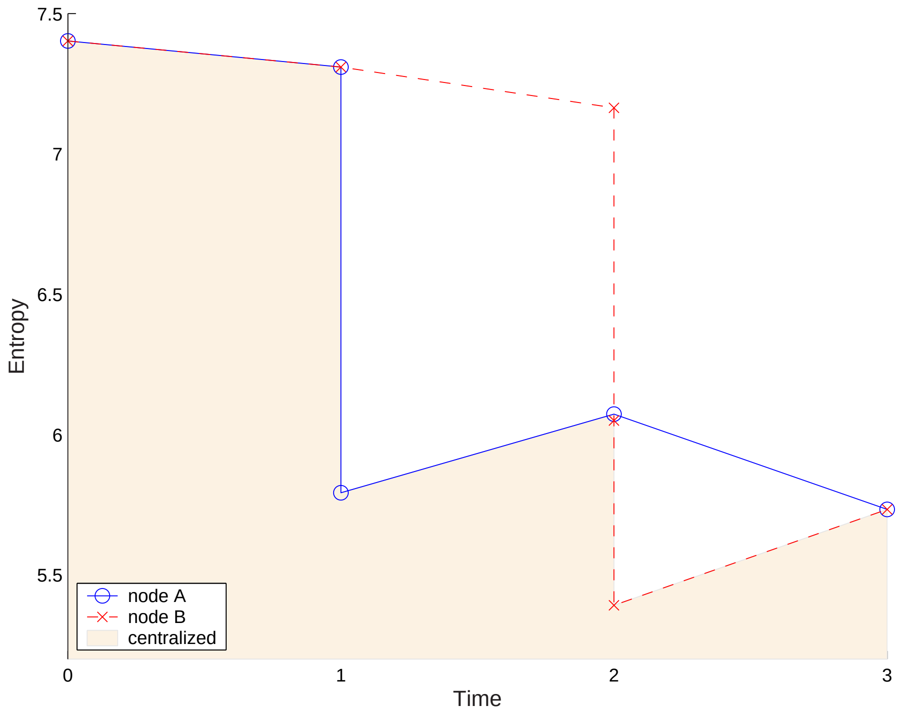


Figure 4.20: Entropy evolution in the grid tracking example.

Both platforms start with an uninformative prior distribution. Platform A makes an observation at t_1. The posterior $P_A(\mathbf{x} \mid Z_A^1)$ is exponential in range and Gaussian in bearing. NODE *A sends a channel update to* NODE *B.* NODE *B receives the message and first calculates the new information contained in it by subtracting common information stored in the Channel Filter. Platform B also makes a local observation. The result is sent to* NODE *B.*

At t_2 platform A does not make local observation and only predicts its belief forward which results in less informative "more spread out" distribution. At t_3 it receives a channel update message from NODE *B and fuses it with its local belief. Both* NODE*s predict their beliefs forward.*

The beliefs at both NODE*s at t_3 are the same. This fact is further highlighted in Figure 4.20 which shows the evolution of the entropy of the state estimate at the two* NODE*s. At the last step t_3 the entropies of the two beleifs are the same.*

4.4.3 Application to Feature Identification

This application is similar to discrete filtering but the state space is usually small and state transition models are trivial. The feature state defines probability of the feature belonging to a certain class, for example $\mathcal{X} = \{rock, bush, tree, other\}$. Observations express probability of a particular identification being made if the feature in fact belonged to a particular class. The following example illustrates decentralized feature identification with two NODEs.

Example 4.5

Consider again the scenario described in Example 4.2. Instead of tracking the target's location the platforms try to identify its type. The target can be one of three types: human, robot, or other: $\mathcal{X} = \{H, R, O\}$. The categories are assumed to be mutually exclusive.

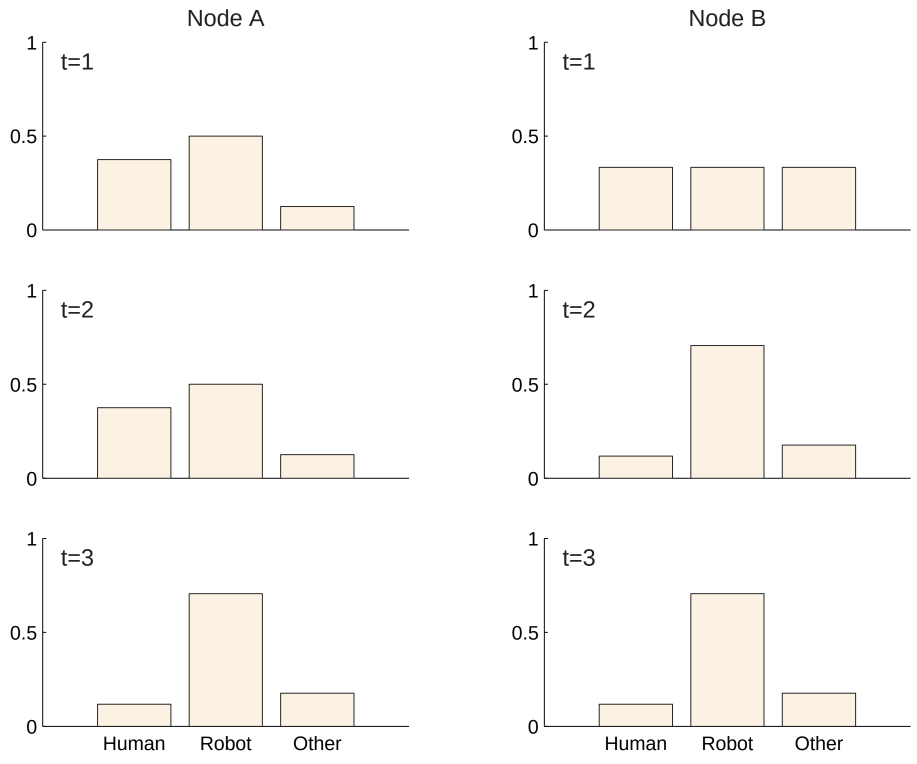


Figure 4.21: Belief evolution in the identification example. At t_1, platform A observes a feature moving with *medium* speed, indicative of a human or a robot. At t_2, platform B observes a *cold* feature, making the presence of a robot more likely. At t_3, the belief on feature identity is the same for both NODEs.

The first sensor detects motion and returns one of four possible values: fast, medium, slow, *and zero speed:* $\mathcal{Z}_1 = \{F, M, S, Z\}$. *Note that the sensor space is different from the state space. The sensor model is described by the likelihood matrix* $L_1(z \mid x)$

$$\begin{array}{ccccc} & z_1 & z_2 & z_3 & z_4 \\ x_1 & 0.05 & 0.50 & 0.30 & 0.15 \\ x_2 & 0.05 & 0.10 & 0.40 & 0.45 \\ x_3 & 0.20 & 0.10 & 0.10 & 0.60 \end{array}.$$

The second platform has an infrared sensor which returns one of two values: warm and cold: $\mathcal{Z}_2 = \{W, C\}$. *Note that the sensor space of this sensor is different from state space and the sensor space of the first sensor. The sensor is described with a likelihood matrix* $L_2(x \mid x)$

$$\begin{array}{ccc} & z_1 & z_2 \\ x_1 & 0.80 & 0.20 \\ x_2 & 0.10 & 0.90 \\ x_3 & 0.10 & 0.90 \end{array}.$$

The motion sensor can discriminate between all feature types but is fairly noisy. The thermal sensor can identify human well but cannot tell robots apart from the environment.

Observations and message exchanges follow the communication sequence in Figure 4.11. The evolution of the local beliefs is shown in Figure 4.21. Both NODE*s start with an uninformative prior distribution. At* t_1 *platform* A *observes a feature moving with medium speed* $z_A^1 = M$. *After this observation is fused, the belief has high probability for both human and robotic objects:* $P_A(x \mid Z_A^1) = (0.38, 0.50, 0.13)$. NODE A *sends a channel update to* NODE B. NODE B *receives the message and first calculates the new information contained in it by subtracting common information stored in the Channel Filter*

$$\begin{aligned} P_B(\mathbf{x} \mid Z_{A\setminus B}^1) &= \alpha \frac{P_A(x \mid Z_A^1)}{P_B(x \mid Z_{A\cap B}^1)} \\ &= \alpha(0.38, 0.50, 0.13) \oslash (0.33, 0.33, 0.33) \\ &= (0.38, 0.50, 0.13), \end{aligned}$$

where $\oslash$ *means element-by-element division.*

The thermal sensor on platform B *observes a* cold *object,* $z_B^2 = C$. *Local observation is fused with current (uninformative) local belief resulting in a posterior which assigns equal*

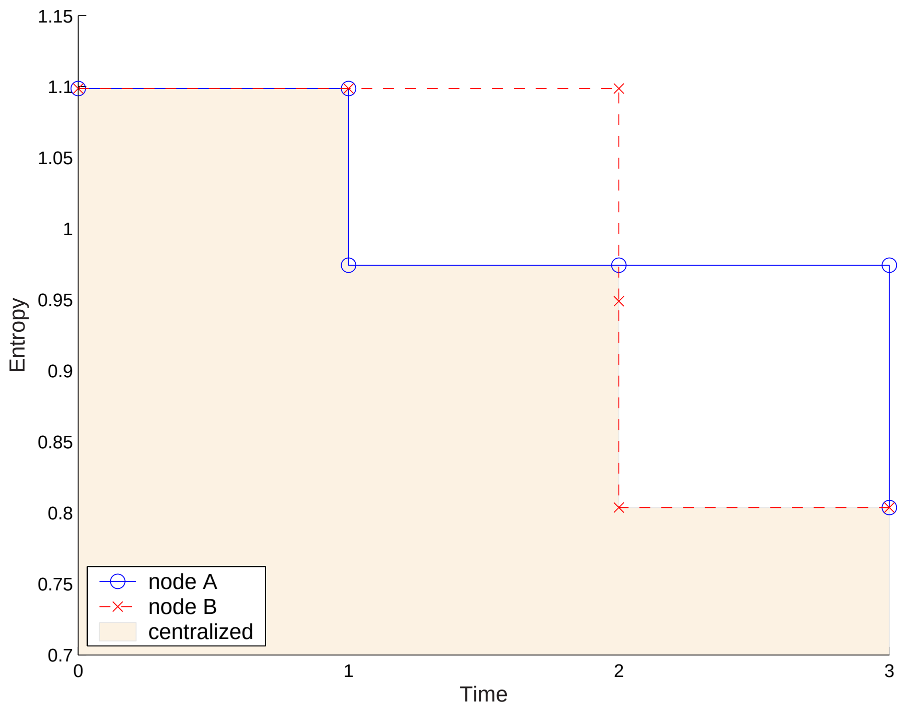


Figure 4.22: Entropy evolution in the identification example.

probability to robots and background, $P_B(x \mid Z_B^2) = (0.10, 0.45, 0.45)$*. The new information contained in the message from* NODE A *is also combined with the local posterior*

$$\begin{aligned} P_B(\mathbf{x} \mid Z_{A\cup B}^2) &= \alpha P_B(x \mid Z_{A\setminus B}^1) P_B(x \mid Z_B^2) \\ &= \alpha(0.38, 0.50, 0.13) \otimes (0.10, 0.45, 0.45) \\ &= (0.12, 0.71, 0.18), \end{aligned}$$

where $\otimes$ *means element-by-element multiplication.*

The resulting posterior is quite confident in the presence of a robot. Now the posterior is sent back to NODE A*, where the new information is extracted*

$$\begin{aligned} P_A(\mathbf{x} \mid Z_{B\setminus A}^2) &= \alpha \frac{P_B(x \mid Z_B^2)}{P_A(x \mid Z_{A\cap B}^2)} \\ &= \alpha(0.12, 0.71, 0.18) \oslash (0.38, 0.50, 0.13) \\ &= (0.10, 0.45, 0.45) \end{aligned}$$

and then fused with the local belief

$$\begin{aligned} P_A(\mathbf{x} \mid Z^3_{A\cup B}) &= \alpha P_A(x \mid Z^2_{B\setminus A}) P_A(x \mid Z^3_A) \\ &= \alpha(0.10, 0.45, 0.45) \otimes (0.38, 0.50, 0.13) \\ &= (0.12, 0.71, 0.18). \end{aligned}$$

At this point the beliefs at the two NODE*s are synchronized, $P_A(x \mid Z^3_{A\cup B}) = P_B(x \mid Z^3_{A\cup B})$, and are the same as would be obtained in the case of centralized fusion. The entropies of the two distributions are shown in Figure 4.22 and, as expected, after the last step they are the same.*

4.4.4 Application to Certainty Grid Maps

This section applies the BDDF algorithm to the problem of constructing Certainty Grid (CG) maps. The CG representation was pioneered by Elfes [47]. It allows a simple and intuitive means of capturing distributed spatial information such as occupancy for indoor spaces or traversability for the outdoors. The CG representation is popular because it supports path planning and navigation [48, 148].

Formally, the CG map is a discrete-state binary random field. Each element encodes the probability of the corresponding grid cell being in one of several possible states. The number of states is not limited to two, although the vast majority of applications use the original binary state space, $\mathcal{X} = \{occupied, empty\}$. [8]

As the CG representation is probabilistic and discrete, the problem of building a map with multiple platform may be viewed as an instance of decentralized identification. In the case of occupancy maps, the states to be identified are *occupied* and *empty* for each grid cell.

The simplicity of the representation makes some of the DDF primitives rather trivial. Data association is not required because the "features" (the CG cells) are referred to explicitly by their locations. A stationary environment is a standard assumption for CG, so location of the cells in the CG map are assumed to be constant and known. This makes the

[8] For an example of a richer representation see [217].

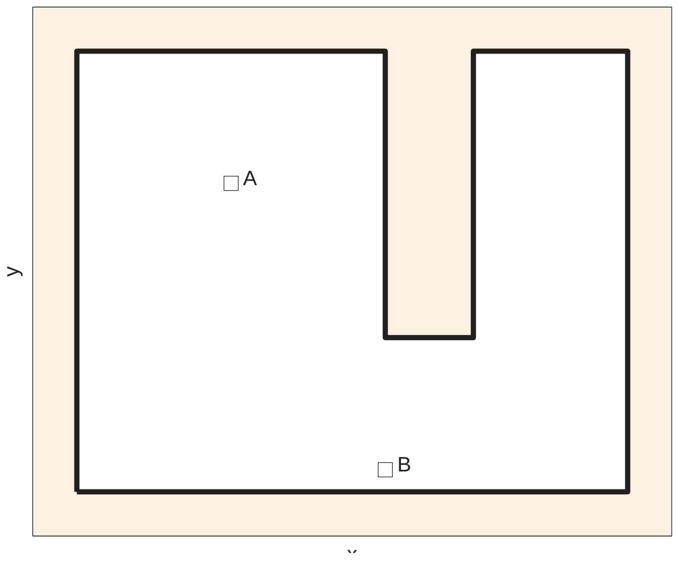


Figure 4.23: Simulated world for the CG example. White color designates open space, black obstacles.

prediction step unnecessary. It is possible to relax this assumption and allow quasi-static environments by introducing a certain amount of information "forgetting" or blurring [48]. The time scale associated with information loss may reflect the expected dynamic aspect of the environment. Any information entered into the map is not permanent and has to be verified periodically.

Standard Channel Filter techniques are applied to keep track of common information between platforms.[9]

Example 4.6

As an example consider a small Certainty Grid map with 22 by 22 cells representing traversability of a flat two-dimensional space. The world layout is shown in Figure 4.23. The locations of two platforms are shown as squares. The sensor model is obtained as first proposed in [47] and is explained in detail in the appendix of [119]. Each platform executes an instance of FRAME*,* SENSOR*, and* NODE *components. The information exchange between the two* NODE*s follows the pattern of Figure 4.11.*

The evolution of the two local belief is shown in Figure 4.24. As in the previous examples,

[9]The problem of fusing information from multiple mappers was considered in the original CG work. An argument is presented in [47] for using the Independent Opinion Pool approach [16] for combining two maps under the assumption that the two maps were built using independent observations.

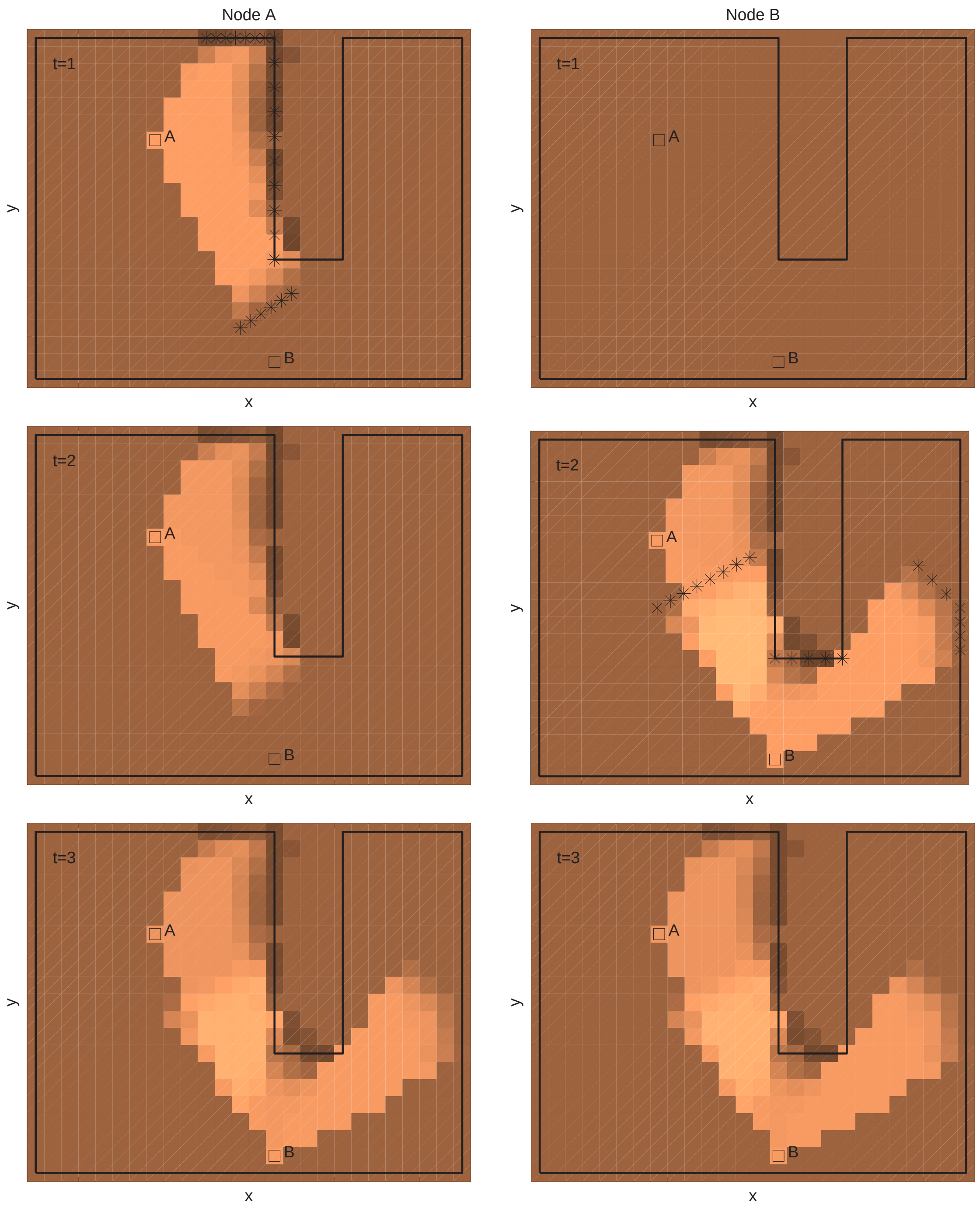


Figure 4.24: Belief evolution in CG map example. Lighter colors correspond to higher probability of traversable (empty) space, darker — to less traversable (occupied). The panels show platform positions (□) and sensor returns (∗).

the maps stored at the two NODE*s are the same after all information has been exchanged. The change in entropy of the entire map is plotted in Figure 4.25. Both platforms start with a blank uninformative map with equal (maximum) entropy levels. The increase in entropy between observations is due to the blurring process.*

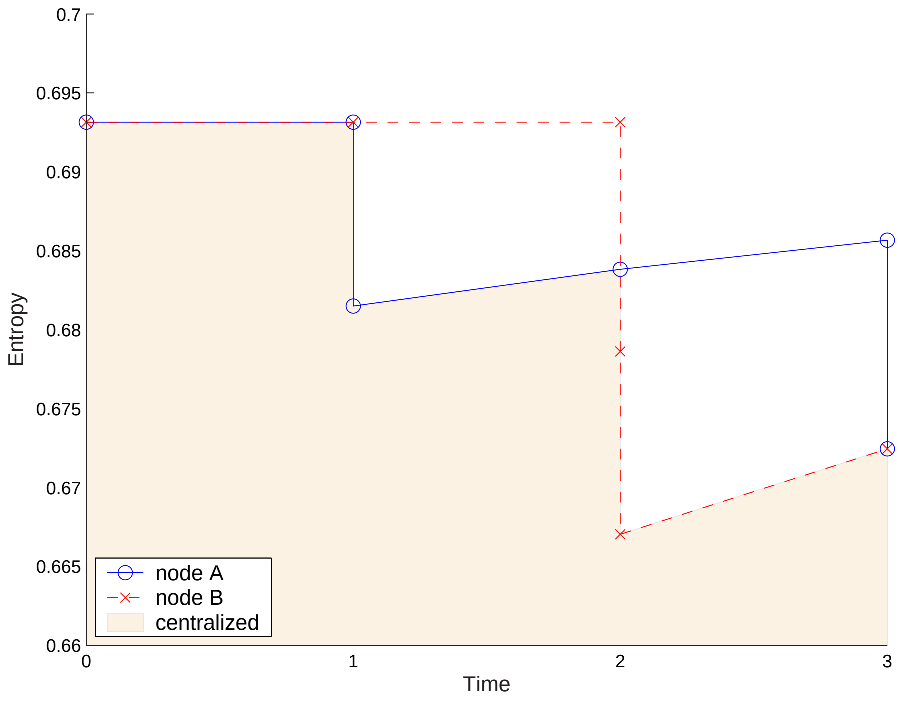


Figure 4.25: Entropy evolution in Certainty Grid map example.

The changes in the entropy are shown in Figure 4.25. Once again, at the last step the entropy is the same. A simple multiplicative form of information "forgetting" is implemented. This explains the increase in entropy (information loss) in the absence of observations. Note that until some information is added to the map, the entropy cannot increase. This explains the fact that the entropy of the belief does not increase until t_1 for NODE *A, and until t_2 for* NODE *B.*

4.4.5 Experiments with Motion Detection Using Certainty Grid Maps

In this section, the decentralized Certainty Grid algorithm is applied to mapping motion in an indoor environment. Unlike the previous simulation example, the following results were obtained experimentally.

The aim of the experiment is to use a dynamically configured sensor network to monitor motion in an office environment over a extended period of time. The office space of approximately 25 by 25 m is represented by a CG map with 30 cm square cells. The state space of each grid cell is binary, $\mathcal{X}=\{static, dynamic\}$.

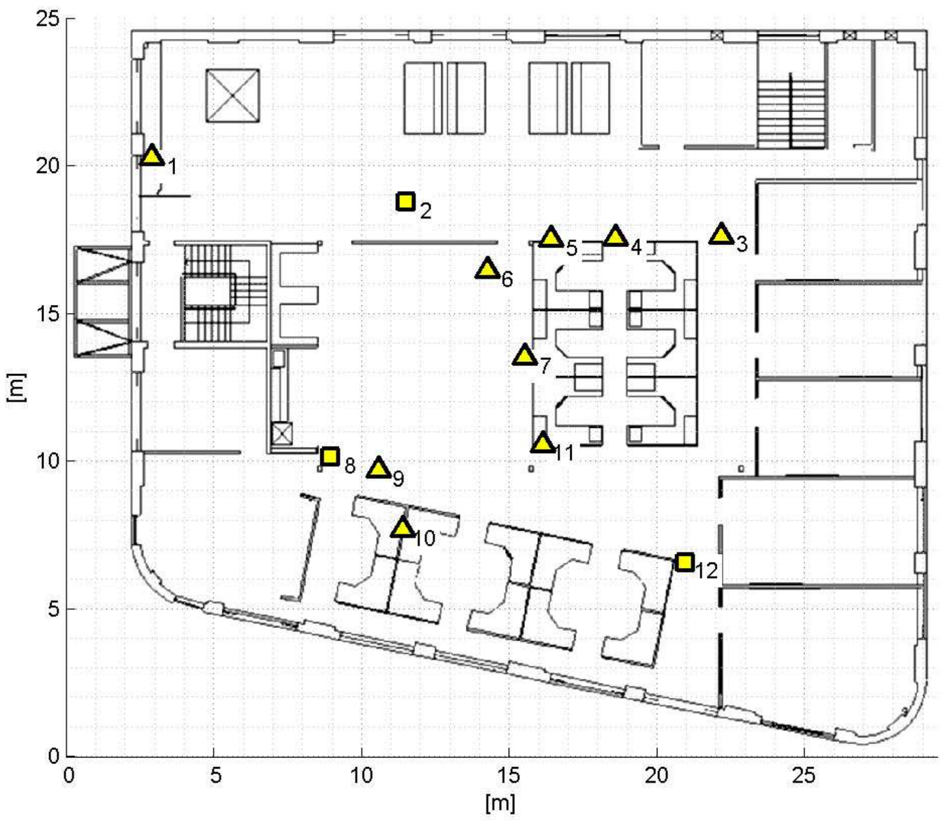


Figure 4.26: Deployment of sensors in the motion mapping experiment. The experiment employed a total of 12 sensors of two types: lasers (□) and video cameras (△).

The task is performed by a team of stationary sensing platforms. Two types of physical sensors were used to detect motion: video cameras and laser scanners. The elements of the sensor network were deployed manually so as to achieve reasonable coverage, as shown in Figure 4.26. Each sensing platform has a processor and executes three software components: a FRAME, a SENSOR, and a NODE. The platform positions were surveyed manually and specified on initialization. Simple frame-differencing algorithms were implemented for both vision and laser sensors. The raw measurements were converted to the likelihoods of motion in particular grid cells. Geometric projections were used to convert from observations in local coordinate systems into the global CG frame.

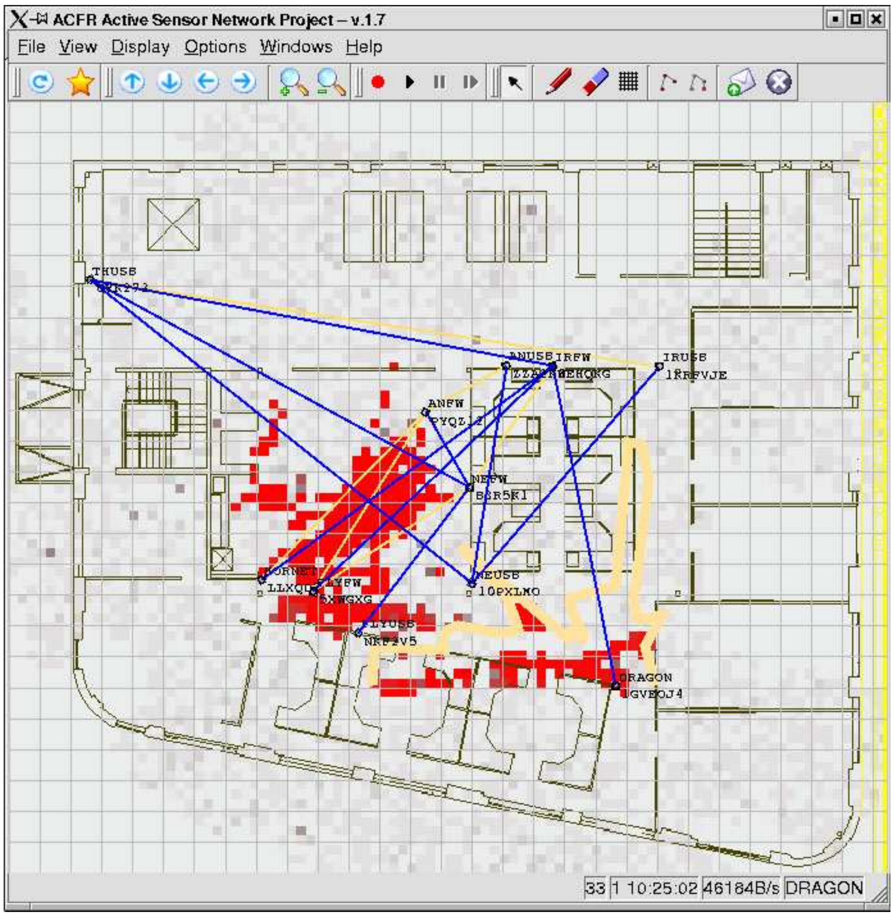

ACFR Active Sensor Network Project – v.1.7
File View Display Options Windows Help
THUSB
ANUSB
IRFW
IRUSB
1RRFVJE
ANFW
NEFW
NEUSB
10PXLMO
LLXQU
LYFW
LYUSB
NKF2V5
DRAGON
33 1 10:25:02 46184B/s DRAGON

Figure 4.27: Certainty Grid map of motion built by the indoor sensor network. Darker red color corresponds to higher probability of motion, lighter blue colors — to higher probability of stationary environment.

The world view of the operator station in Figure 4.27 displays the state of the world as seen from the platform to which the GUI is connected. The motion grid, from the point of view of the NODE in the lower-right corner, is shown in shades of gray and red. Dark-red colors indicate higher certainty of motion in that cell. A large dark-red blob near the center of the map corresponds to a constantly moving group of a dozen or so people participating in a meeting. The office floor plan is overlaid manually for clarity. The hand-drawn plan is inaccurate, which explains the apparent overlap of walls and human motion in some areas.

Blue lines indicate connections between NODEs, along which beliefs are shared.

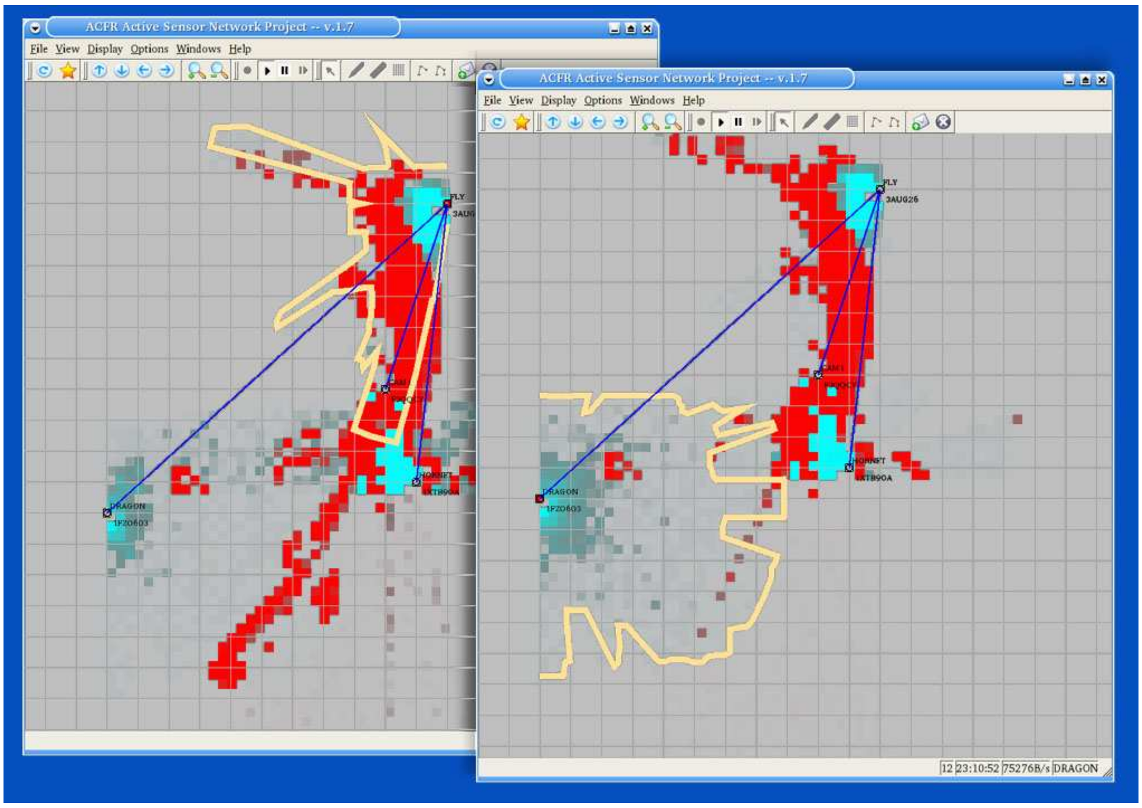

ACFR Active Sensor Network Project -- v.1.7
File View Display Options Windows Help
FLY
3AUG26
DRAGON
1P20603
LKTB90A
12 23:10:52 75276B/s DRAGON

Figure 4.28: View of the environment from two Certainty Grid NODEs.

Figure 4.28 shows near-simultaneous views of the same environment from two NODEs. Instantaneous range-bearing scans from the local lasers are shown in yellow. The two screen shots illustrate the fact that the belief of two NODEs is the same allowing for a certain amount of communication a delay. A person's track coming from the bottom is actually old information which has already disappeared in the belief of the NODE who observes this part of the environment directly (shown in the right panel). The update has not yet reached the NODE whose belief is shown on the left.

This series of experiments described in this section demonstrated application of the Certainty Grid version of the BDDF algorithm to a medium-size team of stationary indoor sensors. Feasibility was demonstrated although hard data on scalability was not collected.

4.5 Summary

This chapter described how the task of information fusion is performed in ASN. The approach was presented in two parts: a general framework for decentralized Bayesian information fusion, called the BDDF algorithm, and its specialization to several *pdf* representations. To make the process of specialization more straightforward, the DDF primitives were identified. These are a small set of information processing operations, which constitute the ASN information fusion algorithm.

A series of simulations for different representations and applications demonstrated the use of the BDDF algorithm. Two specific representations were considered: Gaussian and the discrete grid. Well known algorithms for these representations have been shown to implement various aspects of the BDDF algorithm. A special case of the discrete *pdf* representation is the CG map. The Decentralized Certainty Grid algorithm presented in this thesis is a novel application of the DDF approach to a well known Certainty Grid environment representation.

When choosing an appropriate *pdf* representation for a feature, it is often possible to break up the state vector into independent subsets and represent each subset with the most appropriate representation. The BDDF algorithm and the primitives are applied individually to each group of states. The combined *hybrid* state can easily be decentralized, provided that the DDF primitives for the individual representations exist. One applications of the hybrid state representation is in the integrated track existence and target tracking. An example of this approach is Integrated Probabilistic Data Association (IPDA) [159] which incorporates probability of track existence and target tracking information. Another application combines target identification and tracking. Another use for a hybrid state is in joint target tracking and classification algorithms [29]. Kalman filter tracking and particle filter identification were combined in [59, 199].

The ideas of probabilistic information fusion naturally lead to sensor management. Decentralized decision making is the subject of the next chapter.

Chapter 5

Decision Making Algorithm

5.1 Introduction

This chapter focuses on the task of decision making. As in the case of the information fusion algorithm, two sections from Chapter 3 are used as departure points. The overall algorithmic structure from Section 3.3 and the logical breakdown of the system into canonical components from Section 3.6. Using these as a foundation, this chapter describes the information flow and detailed decision making algorithms used in ASN.

Section 5.2 presents a general framework for decentralized Bayesian decision making. The framework is an evolution of earlier work in decentralized sensor management and a brief historical reference is provided. Continuing with the top-down description approach, the algorithm is described on three, successively more detailed, levels: the information flow between components, between the internal parts of the PLANNER, and inside the internal PLANNER parts. The last level of description is sufficiently detailed to identify a small set of information processing operations, called the Decentralized Decision Making (DDM) *primitives*, which constitute the building blocks of the algorithm. The information processing steps, expressed in terms of the three primitive operations, define the decision making algorithm of ASN called Bayesian Decentralized Decision Making (BDDM) algorithm.

Section 5.3 specializes the algorithm and, specifically, the DDM primitives to the case of Gaussian point features. A group of well known algorithms using Gaussian features are presented as special cases of the general Bayesian form of DDM. An application to tracking point targets is discussed, using simulated and experimental results.

Section 5.4 repeats the process for the discrete grid representation. Two applications are considered in this context: Bayesian feature identification and indoor exploration using Certainty Grid representation.

5.2 Bayesian Decentralized Decision Making Algorithm

This section describes the main decision making algorithm of ASN called Bayesian Decentralized Decision Making Algorithm (BDDM). The algorithm is based on the work started at the robotics group at Oxford University in the early 1990s. Initially, the algorithm was applied to sensor management [144, 145], which is concerned with making discrete assignments of environment features to localization trackers. The approach used information measures in evaluating expected utility and introduced an iterative bargaining algorithm for decentralized sensor assignment. The algorithm was implemented on OxNav, an experimental indoor vehicle. The work of Grocholsky [70, 71] tied the tasks of information fusion and decision making closer together by utilizing the Channel Filter approach for cooperative planning. It also extended the work to the continuous case of controlling a mobile sensing platform. The solution procedure casts platform's decision making as an optimal control problem. Feature representation was limited to Gaussian points and communication topology was pre-configured.

Continuing with the top-down description approach, the information flow between components is considered first in Section 5.2.1. Section 5.2.2 looks inside the PLANNER component and describes the information flow between its internal parts. Then each part is considered in detail and the algorithm is represented in terms of a set of primitive operations.

5.2.1 Inter-Component Information Flow

The information flow between components is considered first. Figure 5.1 shows five canonical components participating in the task of decision making. The four interfaces employed in this diagram are described in detail in Chapter 3. Both human and robotic *action sources* are shown highlighting the fact that the two sources are identical in terms of information flow.

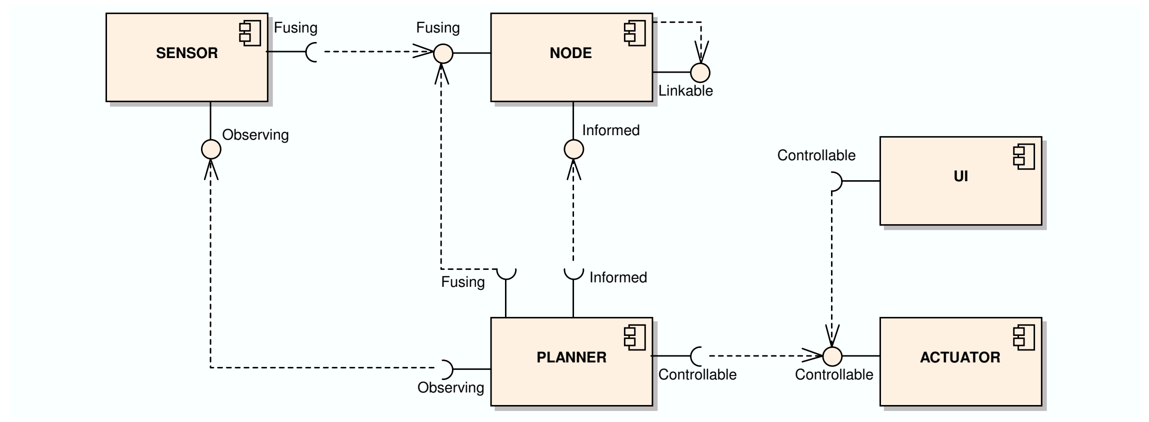


Figure 5.1: Structural diagram of interactions between canonical components used in decision making. Two types of *action sources* are shown: human UI and robotic PLANNER. Here and below, services used in implementing the task of information fusion are omitted for clarity.

System operation depends on the choice of one of three synchronization modes listed in Table 3.1. The case of an isolated decision maker is trivial because the decisions are made locally based only on local information. The other two types, coordinated and cooperative, are illustrated with examples below. A coordinated system is considered first. In this synchronization mode, platforms' beliefs are synchronized but their plans are not.

Example 5.1

Collaboration between components in coordinated mode is illustrated using the scenario shown in Figure 5.2. The diagram shows two Platforms, 1 and 2, each executing four canonical components: a SENSOR*, an* ACTUATOR*, a* NODE*, and a* PLANNER*. The* NODE*s on the two platforms are linked together for belief synchronization. The* PLANNER*s communicate only with components co-located on the same platform.*

The sequence diagram in Figure 5.3 shows the messages passed between the components on platform 1 and cross-platform communication between the NODE*s. The scenario begins with* PLANNER *1 collecting information about the available sensing and actuation hardware through the* Observing *and* Controlling *interfaces of* SENSOR *1 and* ACTUATOR *1 respectively. The two "belief"* NODE*s, B1 and B2, periodically synchronize their beliefs through the* Linkable *interface.*

At decision time, PLANNER *1 gets current belief from local* NODE *B1 and the current actuator*

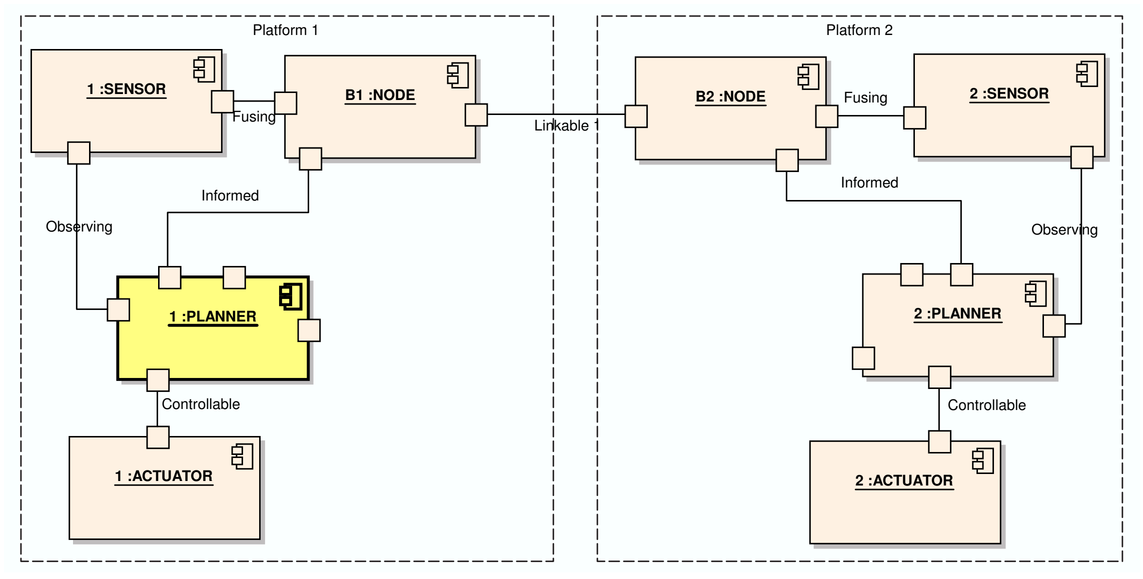


Figure 5.2: A collaboration scenario illustrating coordinated decision making. Two platforms coordinate their actions by synchronizing the beliefs of the corresponding "belief" NODEs, B1 and B2. The scenario description focuses on interactions of PLANNER 1 (highlighted).

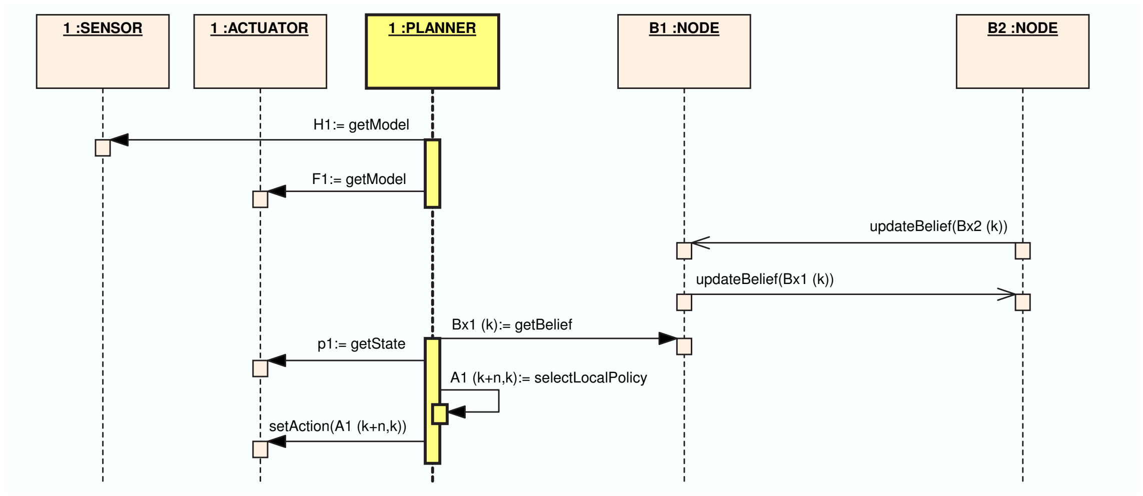


Figure 5.3: Inter-component communication in coordinated decision making.

state from ACTUATOR *1. On the basis of all collected information,* PLANNER *1 generates an action plan. The commands are sent to the* ACTUATOR *1 for execution through the* Controllable *interface. It is then the responsibility of the* ACTUATOR *to implement the* *action plan.*

The next example describes a cooperative system, in which both the beliefs and the observation plans are synchronized between platforms.

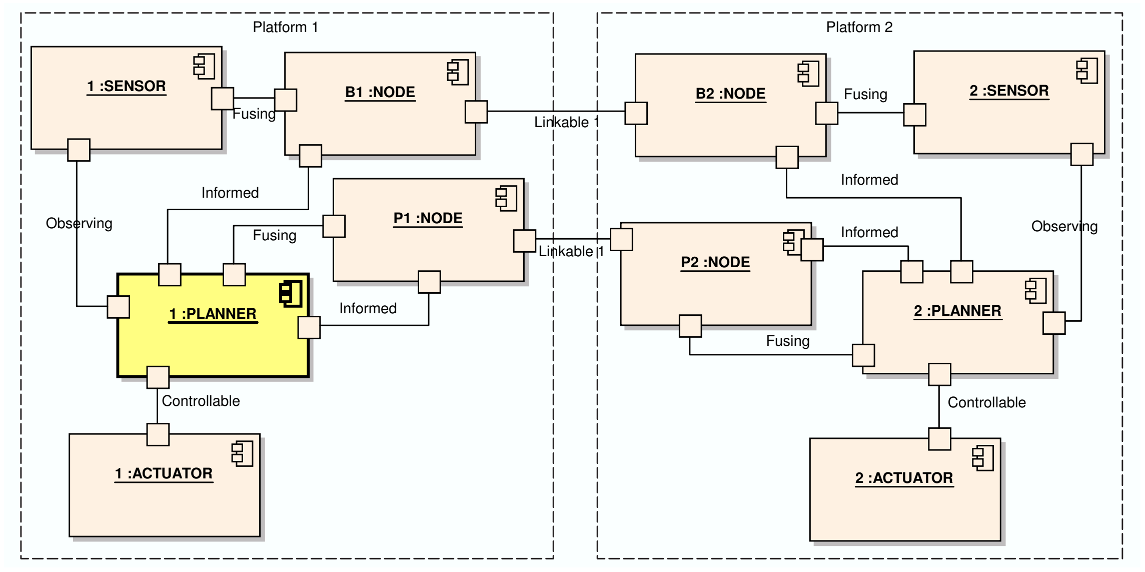


Figure 5.4: A collaboration scenario illustrating cooperative control. Compared to the coordinated scenario in Figure 5.2, the system has two additional "plan" NODEs, P1 and P2, which maintain the belief on the future state of the environment. The scenario description focuses on interactions of PLANNER 1 (highlighted).

Example 5.2 ---

Collaboration between component in cooperative mode is illustrated using a scenario shown in Figure 5.4. The diagram shows two platforms, 1 and 2, each executing five components: a SENSOR, *an* ACTUATOR, *two* NODE*s and a* PLANNER. *The two* NODE*s, B1 and P1, maintain the current and future beliefs respectively. The corresponding* NODE*s on the two platforms are linked together for belief and plan synchronization.*

A sequence diagram in Figure 5.5 shows messages passed between the components. As before, PLANNER *1 collects information about the available sensing and actuating hardware. The two* NODE *pairs, B1/B2 and P1/P2, periodically synchronize their beliefs through the* Linkable *interface.*

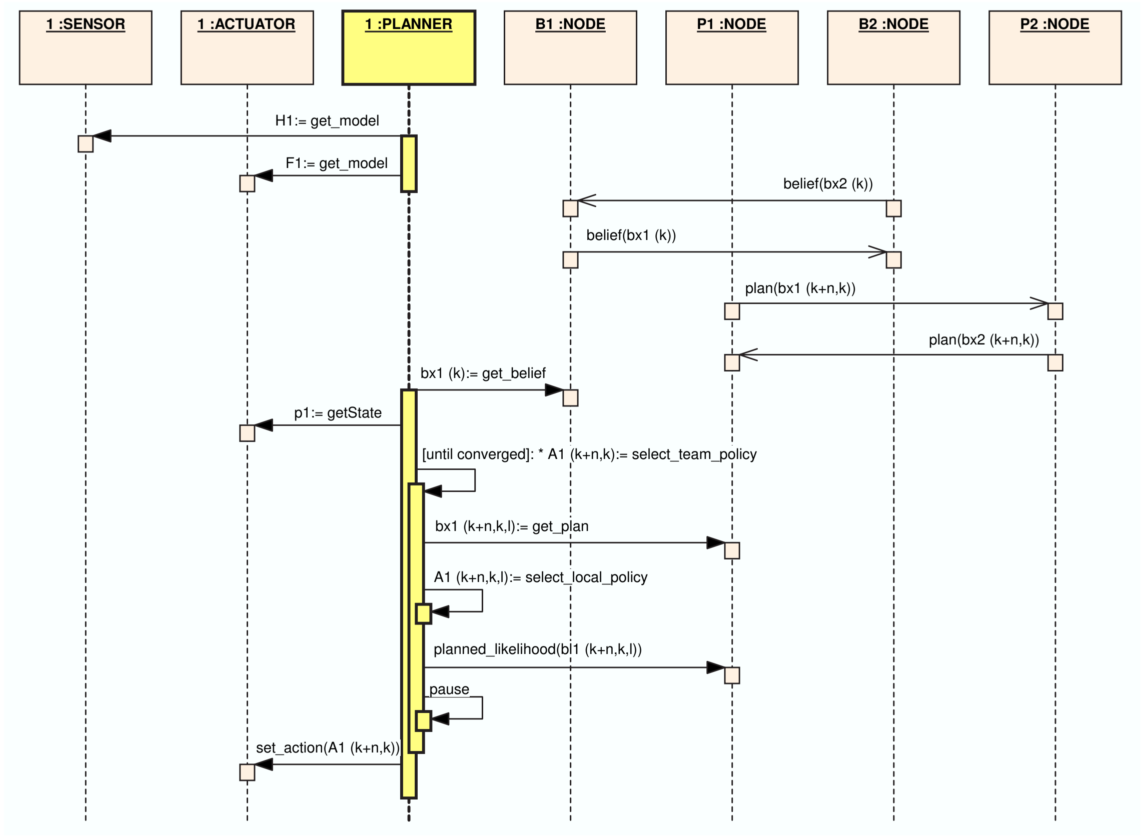


Figure 5.5: Inter-component communication in cooperative decision making.

At decision time, PLANNER *1 gets current belief from local "belief"* NODE *B1, future belief from "plan"* NODE *P1, and the current actuator state from* ACTUATOR *1. On the basis of all collected information,* PLANNER *1 generates a local action plan. Its effect on the world belief is summarized in a single planned observation likelihood which is forwarded to* NODE *P1 for fusion. At this point iteration may be necessary if the actions of the platforms are coupled. After a pause to allow* NODE *P1 to synchronize its plans with the rest of the team,* PLANNER *1 gets an updated team plan, generates a new observation plan, and submits the new planned observation likelihood to* NODE *P1. Once the team plan converged sufficiently, the commands are sent to* ACTUATOR *1 for execution through the* Controllable *interface.*

From the example above it is clear that the PLANNER component is at the center of the decision making process. For this reason, the internal PLANNER structure and algorithms are the focus of this chapter. The main internal subsystems of the PLANNER were identified in Chapter 3. The white-box diagram of internal PLANNER structure is repeated for convenience in Figure 5.6. Belief synchronization is a critical aspect of decision making and is implemented by NODE components. The algorithms for synchronizing observation plans are identical to those described in Chapter 4 in the context of current belief synchronization.

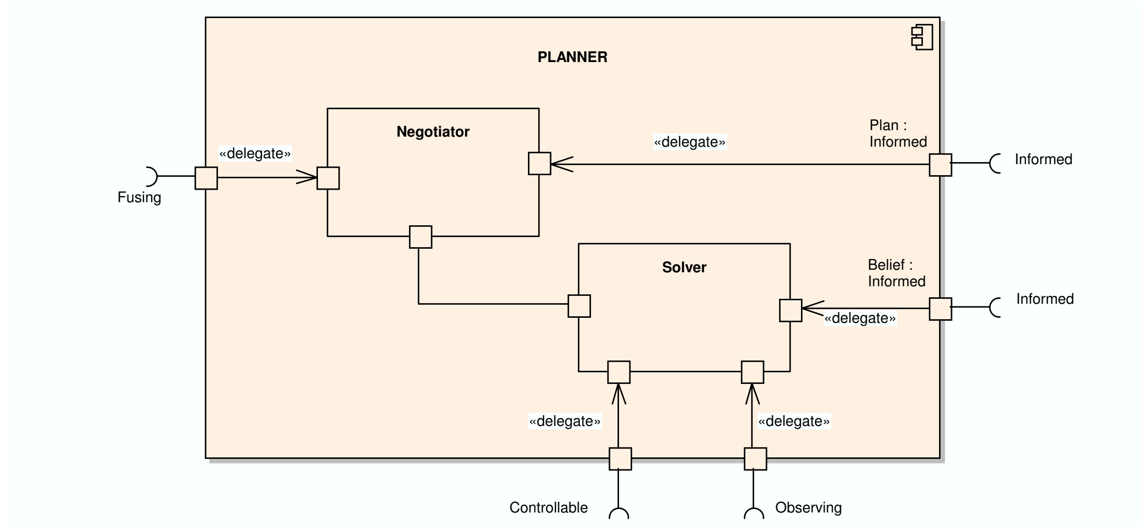


Figure 5.6: A possible realization of the PLANNER component. Repeated after Figure 3.35.

5.2.2 Planner Information Flow

Information flow within the PLANNER component is considered next. Communication diagrams in Figures 5.7 and 5.8 show the flow of information between the PLANNER's external ports and its two internal parts.

Decision making in coordinated mode is confined within a single platform. The process of local policy selection is illustrated in Figure 5.7 with all of PLANNER's functionality implemented by the Solver. It first collects the sensing and actuation models, the current state of the actuator and the environment (messages 1–1.3). Based on this information, the Solver evaluates utility of feasible actions and selects the best one. The chosen action plan is sent through the *Controllable* interface (1.4). Note that, when in coordinated mode, the *Fusing* and *Plan:Informed* interfaces of the PLANNER component are not used.

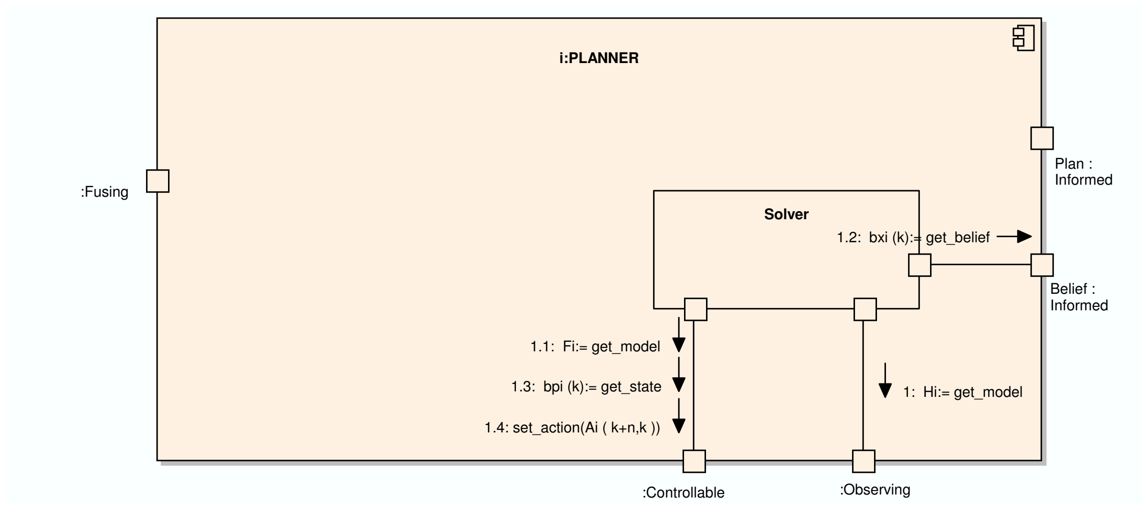


Figure 5.7: Information flow inside PLANNER in coordinated mode.

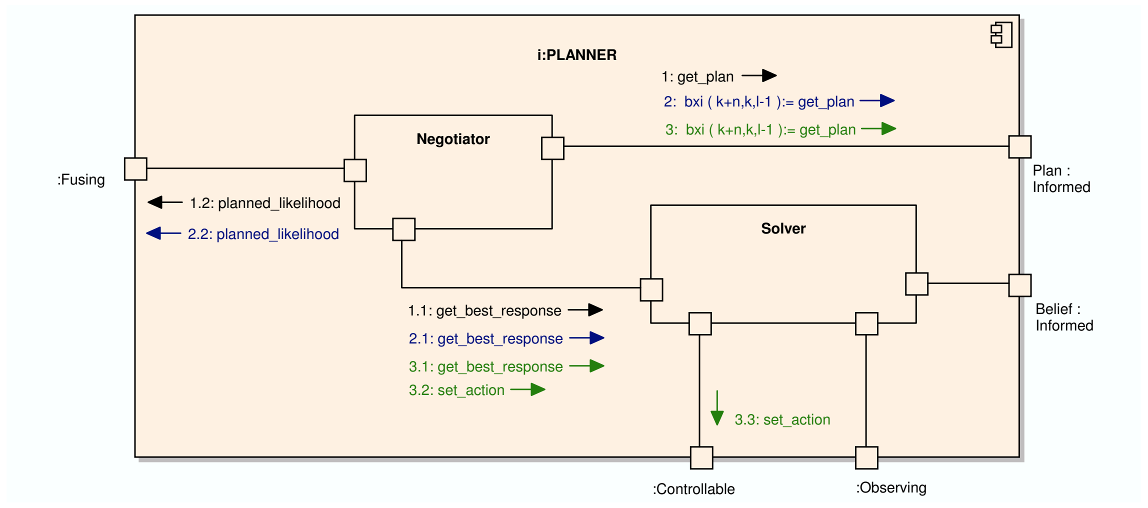


Figure 5.8: Information flow inside PLANNER in cooperative mode.

In cooperative mode, the platform's action policy is negotiated with the rest of the team. The process of team policy negotiation is illustrated in Figure 5.8 and involves two of PLANNER's internal parts: the Solver which evaluates local actions (as in coordinated mode) and the Negotiator which negotiates with the rest of the team. The Negotiator gets the team plan through the *Informed* interface (message 1), tells the Solver to choose the locally optimal policy (1.1), decides on which action to take, and submits the planned observation likelihood through the *Fusing* interface (1.2). The negotiation process is iterative and the same sequence of operation is repeated (message groups 2.x, 3.x). Negotiation terminates when sufficient convergence is achieved. The resulting action plan is sent out through the *Controllable* interface (3.2).

The two diagrams demonstrate that the cooperative mode subsumes the coordinated one. The discussion below focuses on the cooperative synchronization mode since the other two can be viewed as special (and simpler) cases.

5.2.3 Planner Algorithms

This section covers detailed algorithmic steps inside the Solver and Negotiator parts. The diagrams in this section specify the high-level information flow in Figure 5.8.

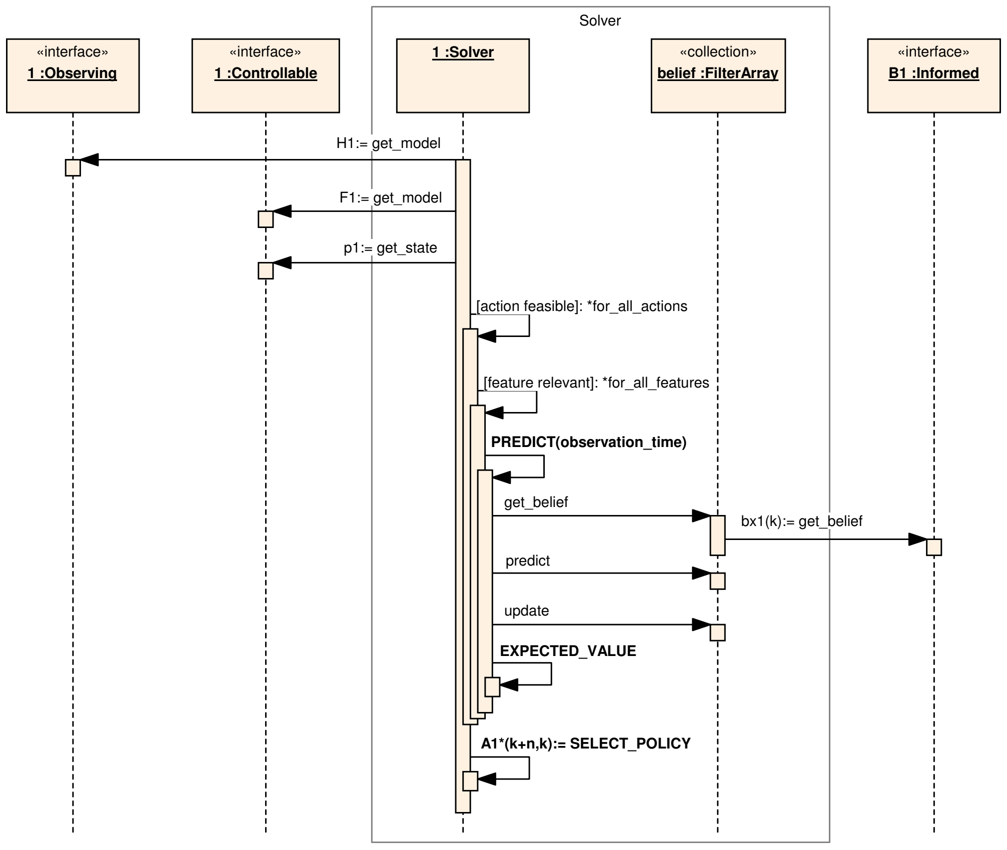

Solver
«interface» 1 :Observing
«interface» 1 :Controllable
1 :Solver
«collection» belief :FilterArray
«interface» B1 :Informed
H1:= get_model
F1:= get_model
p1:= get_state
[action feasible]: *for_all_actions
[feature relevant]: *for_all_features
PREDICT(observation_time)
get_belief
bx1(k):= get_belief
predict
update
EXPECTED_VALUE
A1*(k+n,k):= SELECT_POLICY

Figure 5.9: Information flow in the Solver on local re-plan. This is a more detailed version of Figure 5.7.

Local Policy Selection. Internal steps inside Solver are shown in Figure 5.9. At decision time (triggered by an external event), the Solver collects the necessary information: sensor model from the *Observing* interface, actuation model and current actuator state from the *Controllable* interface. For all feasible actions and all relevant features the Solver evaluates the expected utility. The current belief for each feature is obtained through the *Informed* interface. The action with the highest expected value is chosen.

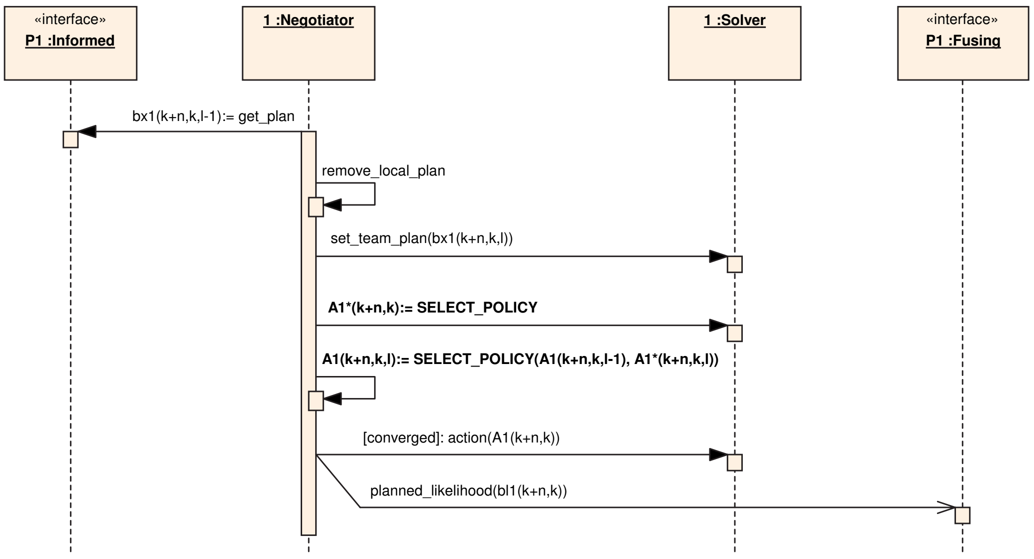

«interface»
P1 :Informed
1 :Negotiator
1 :Solver
«interface»
P1 :Fusing
bx1(k+n,k,l-1):= get_plan
remove_local_plan
set_team_plan(bx1(k+n,k,l))
A1*(k+n,k):= SELECT_POLICY
A1(k+n,k,l):= SELECT_POLICY(A1(k+n,k,l-1), A1*(k+n,k,l))
[converged]: action(A1(k+n,k))
planned_likelihood(bl1(k+n,k))

Figure 5.10: Information flow in the Negotiator. This is a more detailed version of Figure 5.8.

Team Policy Negotiation. Internal steps inside Negotiator are shown in Figure 5.10. First, the updated team plan is obtained. It contains the previous local plan which has to be subtracted. Based on the latest team plan, the Solver generates the action plan with the highest expected value. This action is sometimes called *best response.* Best response procedure in a distributed environment suffers from several problems with local minima, poor convergence, and oscillations. A *better response* algorithm proposed by Grocholsky [70] adds some damping on action update between the stages of negotiation and adds some noise to the process of action selection. The update parameters may be varied during a single optimization run.

5.2.4 Decentralized Decision Making Primitives

A closer examination of the sequence diagrams in Figures 5.9–5.10 reveals that the ASN decision making algorithm can be expressed in terms of three basic operations:

1. Prediction of platform state
2. Expected value evaluation
3. Policy selection

These three basic operations will be referred to as *DDM primitives*. In the diagrams, these three operations are shown in all capitals.

Prediction of platform state. This step involves predicting the platform state forward in time. This part is the same as in the prediction step of the information fusion task as described in Section 4.2.5

$$P(p_i^k \mid Z_i^{k-1}) = \int P(p_i^k \mid p_i^{k-1} = p) P(p_i^{k-1} = p \mid Z_i^{k-1}) \, dp. \tag{5.1}$$

Expected value evaluation. The expected partial value was defined in Section 3.3.4 as

$$\tilde{V}_i^K(\pi_{iA}) \approx \tilde{\rho}_{iB}^{k+K,k} + \sum_{t=0}^{K} \tilde{\rho}_{iA}^{k+t,k} + \sum_{t=0}^{K} \tilde{\rho}_{iM}^{k+t,k}. \tag{5.2}$$

For systems solving the DIG problem, quality of information about the environment, captured in $\tilde{\rho}_{iB}^{k+k,k}$, makes up a major portion of the overall utility. The question remains on how this utility should be calculated. Note that the utilities of action and communication, $\tilde{\rho}_{iA}^{k+t,k}$ and $\tilde{\rho}_{iM}^{k+t,k}$, are application specific.

It was shown by Manyika [145] that the utility based on entropy of the belief satisfies the rationality axioms[1] guaranteeing a preference ordering $U_B = \mathcal{E}n(x_i^k)$. Entropy is given by $\mathcal{E}n(x) = \int_{\infty}^{\infty} P(x) \log P(x) \, dx$. The entropic form of Bayes theorem shown in Appendix B.3 is repeated here

$$\mathcal{E}n(x \mid z) = \mathcal{E}n(z \mid x) + \mathcal{E}n(x) - \mathcal{E}n(z). \tag{5.3}$$

[1]Proof that the axioms imply the existence of a utility function can be found in [16].

Being able to describe changes in entropy as a result of observations leads naturally to asking an important question: which observation is the most informative. This question may be answered through the concept of mutual information. The mutual information $\mathcal{I}(x, z)$ obtained about a random variable x with respect to a second random variable z is defined as

$$\mathcal{I}(x, z) = -E\left\{\log \frac{P(x \mid z)}{P(x)}\right\} = -E\left\{\log \frac{P(z \mid x)}{P(z)}\right\}. \tag{5.4}$$

Mutual information is an *a priori* measure of the information to be gained through observation. It is a function of the ratio of the density $P(x \mid z)$ following an observation to the prior density $P(x)$. The expectation is taken over all possible z and x, so the mutual information gives an average measure of the gain to be expected *before* making the observation. (See Appendix B.3 for a summary of the relationship between entropy and mutual information.)

In coordinated mode, the partial belief utility is found as a mutual information between the current local belief b^k_{xi} and the planned future observation $z_i^{k+n,k}$

$$\tilde{\rho}_{iB}^{k+K,k} = \mathcal{I}^{b^k_{xi}}(x^{k+n,k}, z_i^{k+n,k}). \tag{5.5}$$

In cooperative mode, the team plan $b_{x\alpha}^{k+n,k}$ (with the local plan subtracted) is used instead of the current belief

$$\tilde{\rho}_{iB}^{k+K,k} = \mathcal{I}^{b_{x\alpha}^{k+n,k}}(x^{k+n,k}, z_i^{k+n,k}). \tag{5.6}$$

Policy selection. The choice of appropriate policy selection mechanism is application dependent. Two approaches have already been mentioned: best response and better response. In the context of negotiation, Grocholsky [70] considers two practical algorithms: Generalized Jacobi and Generalized Gauss-Seidel algorithms. The former is synchronous, i.e. decisions at different platforms are made simultaneously, and the latter is sequential, i.e. decisions are made one at a time.

It is instructive to compare the decision making primitives identified in this section to the DDF Primitives in Section 4.2.5. Only one aspect of the DDM primitives, evaluation of belief utility, depends on the belief representation. In the decision making algorithm, more choices are left unspecified. Virtually all aspects of policy selection can be varied depending on the application and remain active research topics.

5.3 Gaussian Point Features

Gaussian point features were considered in Section 4.3 from the point of view of information fusion. Feature and sensor models for Gaussian distributions are described in Appendix C.1. This section extends the discussion into the task of decision making. The applications to active feature localization is discussed in Section 5.3.2.

5.3.1 DDM Primitives

Algorithms for Gaussian point features are well known and are summarized for reference in Appendix C. This section maps these algorithms to the DDM primitives.

The method of predicting future platform state is the same as in Chapter 4, using the Kalman or Information filter algorithms.

The information measures used in evaluating the belief utility U_B is described in Appendix C.4. In coordinated mode, the partial belief utility is found as a mutual information between the current belief $\mathbf{Y}_i^k$ and the belief at the planning horizon $\mathbf{Y}_i^{k+K,k}$

$$\tilde{\rho}^{k+K,k} = \frac{|\mathbf{Y}_i^{k+K,k}|}{|\mathbf{Y}_i^k|}. \tag{5.7}$$

In cooperative mode, the team plan $\mathbf{Y}_\alpha^{k+K,k}$ (without the local plan) is used instead of the current belief

$$\tilde{\rho}^{k+K,k} = \frac{|\mathbf{Y}_i^{k+K,k}|}{|\mathbf{Y}_\alpha^{k+K,k}|}. \tag{5.8}$$

If there is a finite discrete set of policy options then simple evaluation and ranking provides an obvious means of policy selection. Such an approach has been demonstrated in the context of feature-based SLAM for a single platform [20, 52]. The alternative is to consider the continuous motion problem when policy selection is cast as a deterministic optimization problem [70]. The *Information surfing* algorithm is a special case of this approach with one-step look-ahead planning horizon. It is described in Appendix C.5.

5.3.2 Application to Active Target Localization

The objective is to find where stationary targets are located. The sensing platforms can move in order to improve information quality. The next two examples show simulation and experimental results of coordinated control of sensing platforms.

Example 5.3

This example demonstrates coordinated solution for platform motion control.[2] *Let us consider two bearings-only sensor platforms localizing a single point feature. Planning with zero look-ahead provides a special case in coordinated multi vehicle control. This can be used to form simple approximate solutions where the sensor platforms are directed by the dynamics of the gradient field of the mutual information rate* $\mathcal{I}(t)$*. The Information Surfing algorithm used here for action selection is described in Appendix C.5.*

The trajectories of platforms in isolated and coordinated synchronization modes are shown in Figure 5.11. Three snapshots of the locally optimal trajectories, with and without belief synchronization are shown in panels (a)–(c). The evolution of feature information available to the platforms is plotted in panel (d). The PLANNER*s on both platforms implement local control laws which maximize their individual information gain from bearings-only observations given local belief. Coordination results from the DDF algorithm updating local beliefs from which the optimal action is generated.*

This comparison demonstrates the improvement in performance achieved simply by employing the DDF algorithm in the network of decision makers with local information seeking PLANNER*s.*

5.3.3 Experiments with Active Target Localization

This section shows experimental results of implementing coordinated control of sensing platforms within the ASN framework. It uses a small-scale network of indoor mobile platforms described in Section 4.3.3.

[2]This example has been previously published in [71].

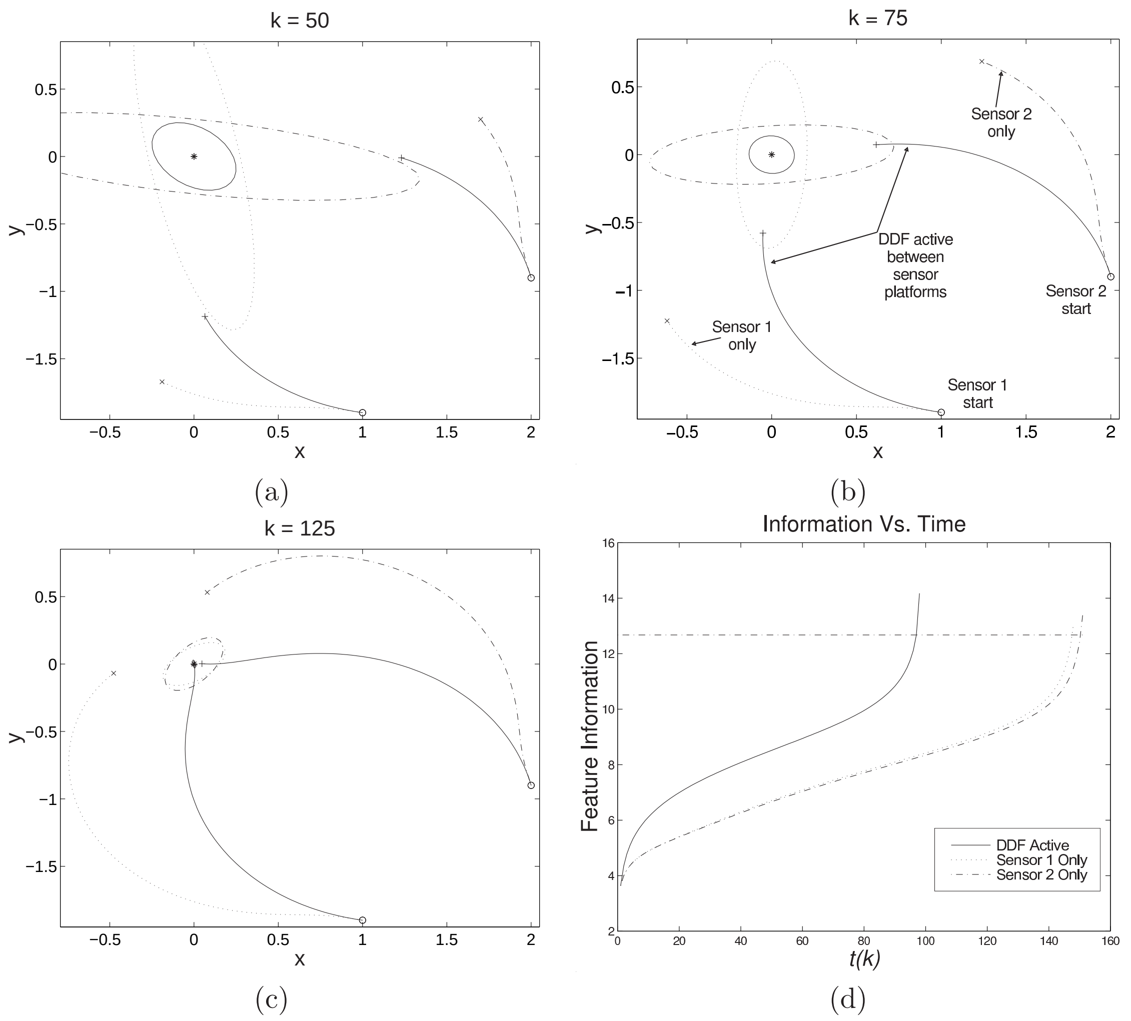
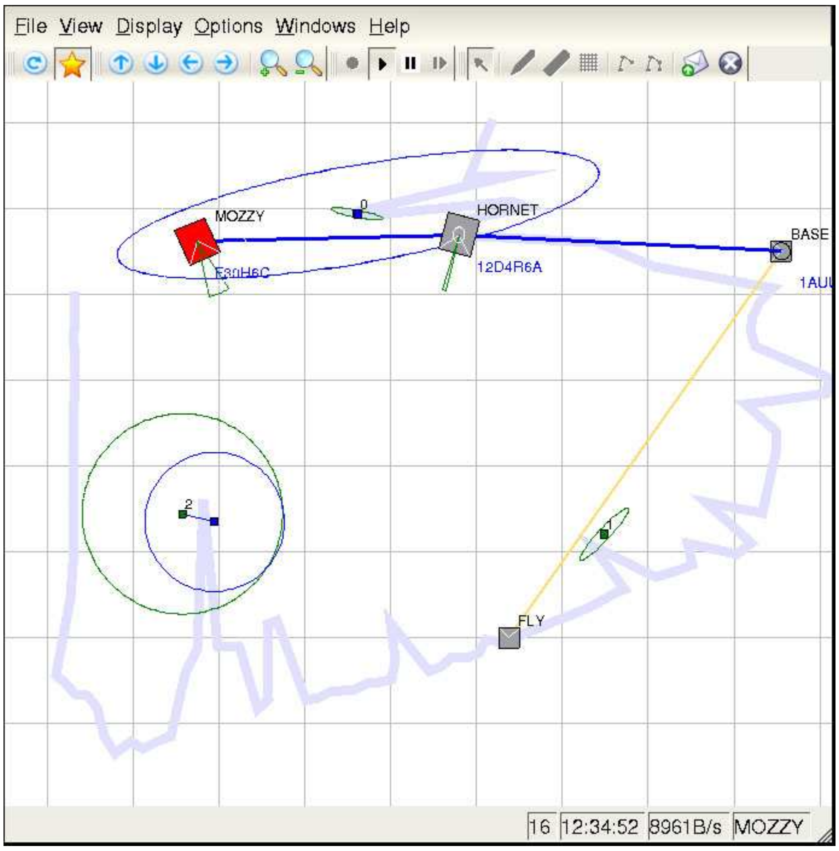
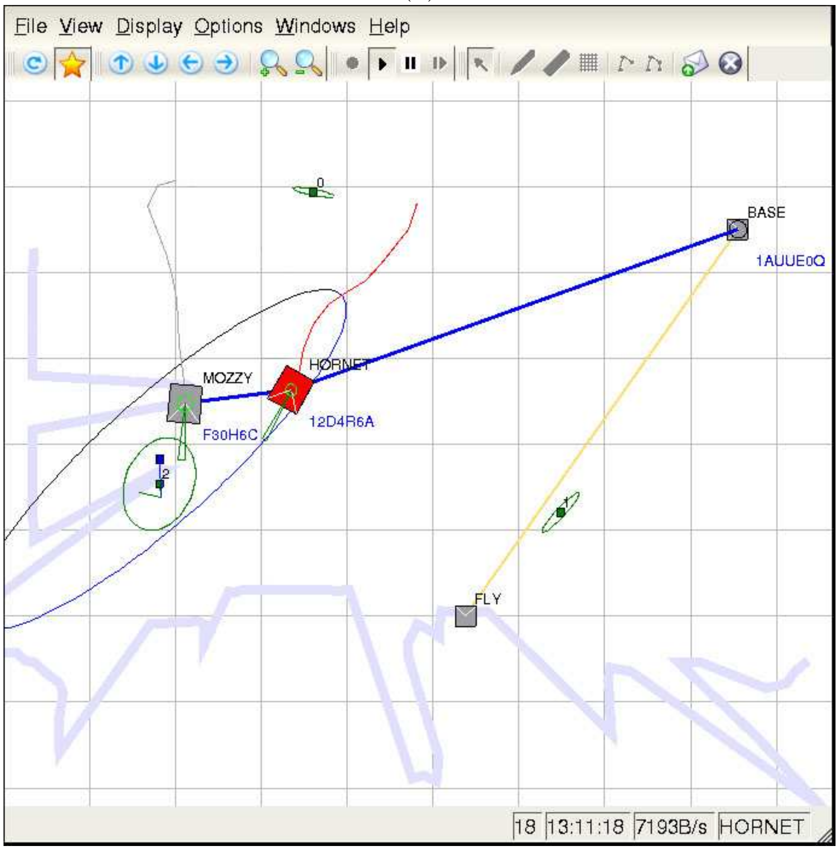


Figure 5.11: An example of coordinated target localization: (a)–(c) platform trajectories and (d) collected information (from [71]). Platform trajectories for isolated and coordinated modes are different from each other. At any given time, the platforms gather more information about the environment feature when operating in coordinated mode.

For the experiments involving decision making, each Pioneer robot executes four canonical components: a Pioneer FRAME, a Gaussian point NODE, a SENSOR which converts laser scans to observations of Gaussian point features, and a PLANNER which implements the Information Surfing algorithm described in Appendix C.5. The stationary sensor does not make any decisions so it does not need a PLANNER. All together, the network consists of 13 components on 5 hosts: three NODEs execute the Information filter algorithm and three SENSORs convert laser scans to Gaussian point observations.



(a)



(b)

Figure 5.12: Experiments with coordinated information surfing: (a) information available to the robots before the start of planning and (b) robots trajectories.

Features 0 and 1 in Figure 5.12(a) were acquired by the robotic SENSORs, feature 2 is outside of the platforms' sensor range and was entered by the operator as described in Section 4.3.3. The quality of information about the location of features 0 and 1 is too high to justify platform motion. Once a human-entered observation with large uncertainty becomes available to the platform, the expected utility exceeds the threshold and both platforms start to move. Platform trajectories are shown in Figure 5.12(b). Note that the estimated location of feature 2 has moved and its uncertainty has decreased due to the actual observations made by sensors mounted on the robots.

The experiment described in this section demonstrated application of the Gaussian version coordinate decision making to indoor active target localization. The experiments validate the basic approach but to be useful a much better integration with basic platform competence is necessary. One of the conclusions was that multi-step lookahead is required for any real-life scenario. A significant implementation effort required for robust optimization algorithms became the limiting factor for further work in this direction.

5.4 Discrete Grid Representations

Discrete grid feature representations were considered in Section 4.4 from the point of view of information fusion. Feature and sensor models for discrete distributions were described in Section 4.4.1. This section extends the discussion into the task of decision making. Sections 5.4.2 and 5.4.3 discuss two applications: feature identification and exploration with CG maps. Application to tracking is not discussed but some work in this direction can be found in [19, 21].

5.4.1 DDM Primitives

In coordinated mode, the utility of information gain is calculated by comparing the current belief with the planned belief based on local action plan $A_i^{k+n,k}$

$$\tilde{\rho}_{iB}^{k+K,k} = -E\left\{\log \frac{P(x^{k+K} \mid x^k, A_i^{k+n,k})}{P(x^k)}\right\}. \tag{5.9}$$

In cooperative mode, the basis for comparison is the planned belief based on team action plan $A_\alpha^{k+n,k}$

$$\tilde{\rho}_{iB}^{k+K,k} = -E\left\{\log \frac{P(x^{k+K} \mid x^k, A_i^{k+n,k})}{P(x^{k+K} \mid x^k, A_\alpha^{k+n,k})}\right\}. \tag{5.10}$$

Analytical control solutions are intractable with this representation. Numerical optimization techniques are possible but have not been demonstrated. Policy selection approaches have been limited to enumeration of options and selection of the one with highest expected value.

5.4.2 Application to Feature Identification

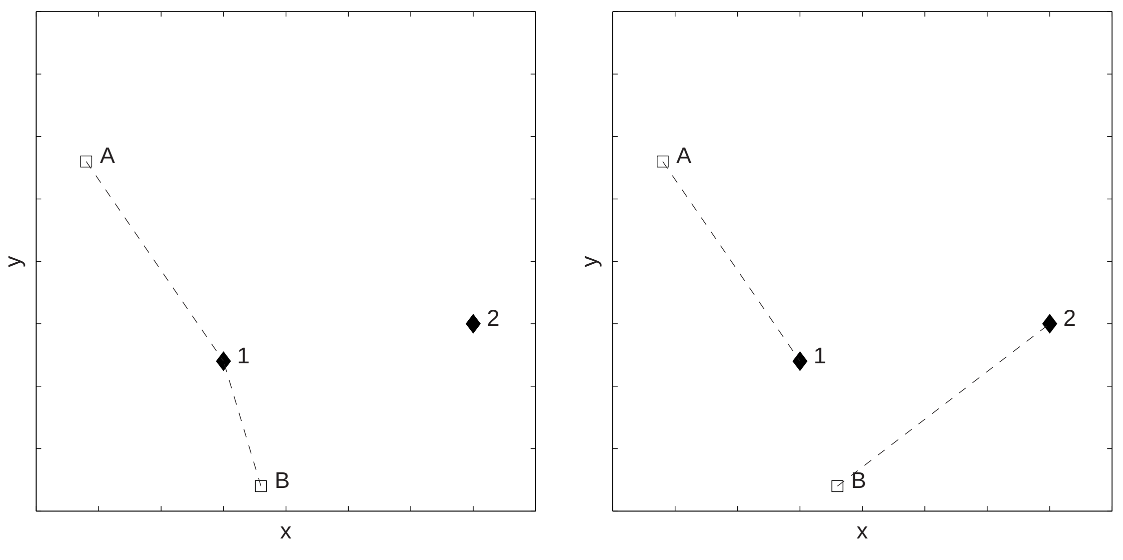
The following example demonstrates cooperative feature identification in simulation. The decision corresponds to target assignment. Extensive experimental work on this topic is described in [145].

Example 5.4 ———————————————————

This example uses the temperature and motion sensors described in Example 4.5 and considers the task of sensor management with two sensing platforms and two features. The system composition is the same as in Figure 5.4. The platform and feature locations are shown in Figure 5.13. The list of feasible actions contains three options: observe feature 1, observe feature 2, and make no observations. Both platforms can choose to observe the same feature.

Both the current and planned beliefs are initialized with a uniform distribution. The coordinated synchronization mode is considered first for comparison. Since both feature beliefs are equally uninformative, both observation actions are equally attractive for both platforms. If distance to the feature is part of the action utility (i.e. platforms must approach the feature for identification), then both platforms select to observe feature 1. This outcome is shown in Figure 5.13(a). The result is suboptimal compared to the cooperative mode which is considered next.

Information exchange between platforms follows the pattern shown in Figure 4.11, but, instead of the current beliefs, the platforms synchronize their plans. Platform decisions are made sequentially following the Generalized Gauss-Seidel algorithm.



(a) (b)

Figure 5.13: Target assignment in the feature identification example: (a) coordinated and (b) cooperative synchronization modes.

Evolution of the planned belief state is shown in Figure 5.14, starting with an uninformative distribution. At t_1*,* PLANNER *A considers which action to choose. It makes a decision to observe feature 1, because the information gain utility* $\tilde{\rho}^k_{iB}$ *is the same and feature 1 is closer. To inform the rest of the team about its decision, it must form a planned observation likelihood. The first step is to find the most likely observation using the current prior (uniform) and the observation model for* SENSOR *A. The probability of all four possible observations are* $[0.10, 0.23, 0.27, 0.40]$*, with a zero-speed observation the most likely. The observation likelihood assuming the most likely observation is* $[0.15, 0.45, 0.60]$*. When submitted to* NODE *A and combined with the prior, it results in the belief* $[0.13, 0.38, 0.50]$ *as shown in the top-left panel of Figure 5.15.*

NODES *A and B exchange information using standard BDDF algorithm. In addition to receiving the plan from* NODE *A,* NODE *B makes its own decision. The expected reduction in entropy after making a single observation for features 1 and 2 are 0.136 and 0.253 respectively. Thus observing feature 2 is preferred because, on average, the observation will be more informative. To form a planned observation likelihood, the most likely observation must be found. The probability of two possible observations are* $[0.33, 0.67]$*, with cold observation more likely. The observation likelihood for the more likely observation is* $[0.20, 0.90, 0.90]$*.*

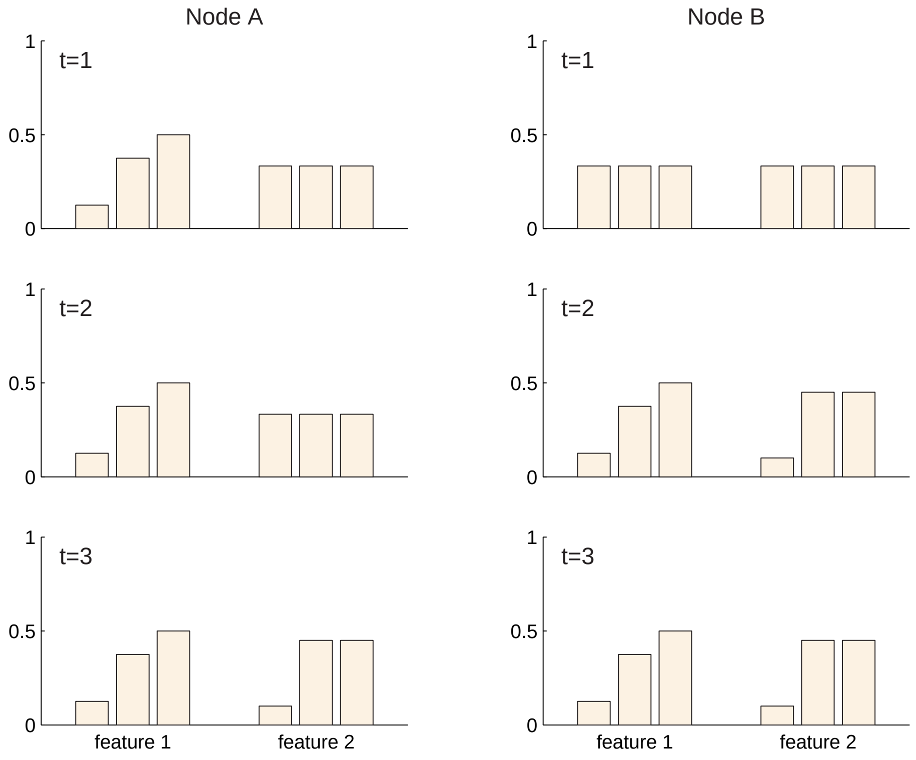


Figure 5.14: Evolution of the observation plan in the cooperative identification example. The graphs show the planned belief at the planning horizon, $b_{xi}^{k+1,k}$. At t_1, platform A makes a decision to observe feature 1. At t_2, platform B responds with a decision to observe feature 2. At t_3, the team plans are synchronized and ready for execution.

When combined with the belief by NODE *B, it results in* $[0.10, 0.45, 0.45]$ *as shown in the mid-right panel of Figure 5.15. By* t_3*, the plans on both platforms are synchronized and ready for execution.*

So far, the decisions made are based solely on the utility of belief quality. Utility of action can be a function of the distance to feature in the case when a platform has to move to a feature to perform observations. The two partial utilities are combined as follows

$$\tilde{V}_i^K(\pi_{iA}) \approx (1-\lambda)\tilde{\rho}_{iB}^{k+K,k} + \lambda\tilde{\rho}_{iAN}^{k+K,k} + 0, \tag{5.11}$$

where λ *is a weighting coefficient and the utility of communication is zero. Consider again the decision process of platform B at* t_2*. Figure 5.15 shows the change in expected value of the two actions as a function of the weighting coefficient* λ*. Based on the utility of information gain alone (*$\lambda = 0$*), the* PLANNER *B chose to observe feature 2. For* $\lambda > 0.71$*,*

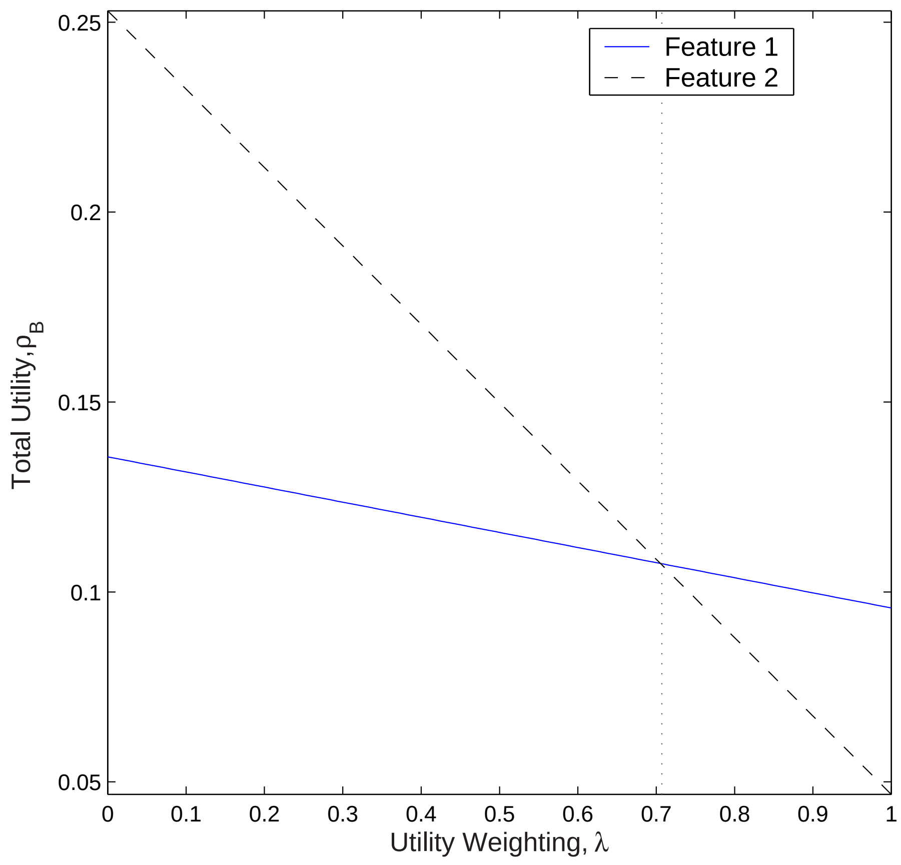


Figure 5.15: Expected value as a function of the weighting coefficient. Platform B prefers to observe feature 2 when the distance to target is not taken into account. Observing feature 1 becomes preferred for $\lambda > 0.71$.

the penalty on driving to a more distant feature 2 outweighs the higher expected information gain, and observing feature 1 becomes preferred.

5.4.3 Experiments with Exploration Using Certainty Grid Maps

The Certainty Grid (CG) representation was described in Section 4.4.4 in the context of information fusion. This section considers CG from the point of view of decision making. Experiments were conducted with a single platform performing exploration of an indoor office environment. In ASN terms, this is equivalent to a platform acting in the *isolated synchronization mode*. The same exploration algorithm can be used by several platforms in coordinated mode.

The expression for expected value was given in Equation 5.2. In the exploration experiments, the utility of action is further divided into the integral utility of locomotion and the terminal utility of localization quality. The latter takes into account the expected quality of localization at the proposed destination point. The total expected value is calculated as

$$\tilde{V}_i^K(\pi_{iA}) \approx (1-\lambda)\tilde{\rho}_{iB}^{k+K,k} + \lambda \left(\sum_{t=0}^{K} \tilde{\rho}_{iAN}^{k+t,k} + \tilde{\rho}_{iAL}^{k+K,k} \right) + 0, \tag{5.12}$$

where λ is a weighting coefficient. Note that, since a single platform was used, the utility of communication is zero. Details of utility definitions are provided for reference in Appendix D.

The individual and combined effect of the three utilities can be observed in Figure 5.16. The belief utility favors regions near the edges of the known map, the navigation utility prefers the vehicle to stay close to current location, and the localization utility leads the robot towards the well mapped regions. From the list of proposed destinations, the location with the highest total utility value is selected as the next destination.

The exploration algorithm works as follows. A list of destinations is first proposed. For each destination, the expected value is evaluated and the destination with the highest value is selected. The action plan in this case is a list of way points to the chosen destination. Details of the Integrated Exploration algorithms are provided for reference in Appendix D.1.

The map built by the robot is shown in Figure 5.17. Exploration time was approximately 20 minutes. The size of the map is 25 by 25 meters. A single laser was used for sensing the environment. Platform localization was performed using a feature-based SLAM algorithm [240]. The features were artificial light-reflecting markers placed throughout the environment.

One novel feature of the exploration algorithm is the inclusion of the utility of localizability. To demonstrate its effect on the platform behavior, the light-reflecting markers used for localization were not distributed uniformly, with fewer of them placed around the periphery of the building. With utility of localizability disabled (its relative weight set to zero), the platform quickly became lost in areas with few beacons. When the localizability utility taken into account, the platform does not venture out into territory far from well marked

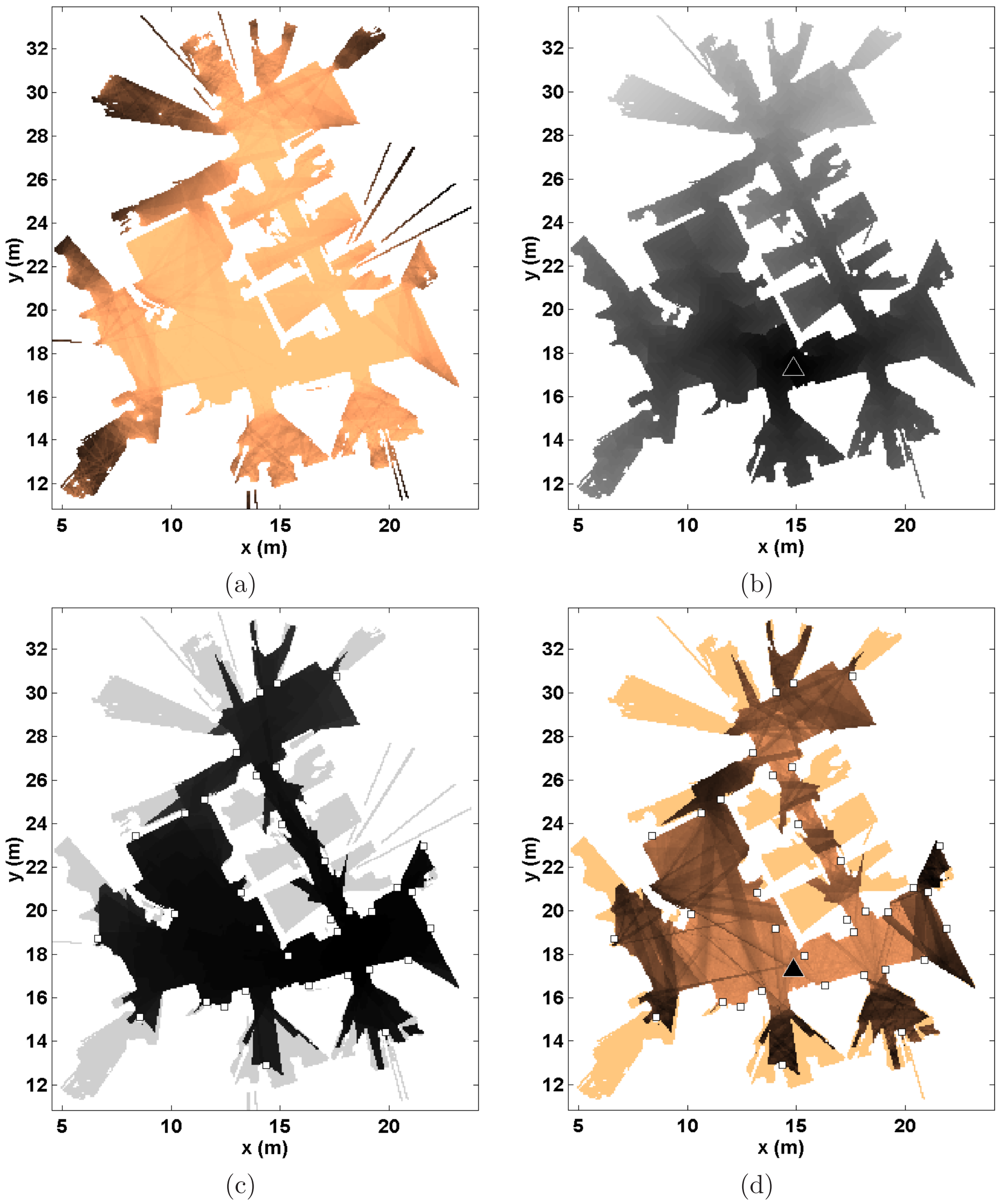


Figure 5.16: Illustration of the planning process for a section of an experimentally obtained map: (a) information gain utility, (b) navigation utility, (c) localization utility (for $k = 10$) and (d) total utility. All utilities favor dark colored regions. Also shown where relevant: current vehicle position ($\triangle$), landmarks ($\square$) and potential destinations ($*$). All utilities are calculated off-line for illustration only.

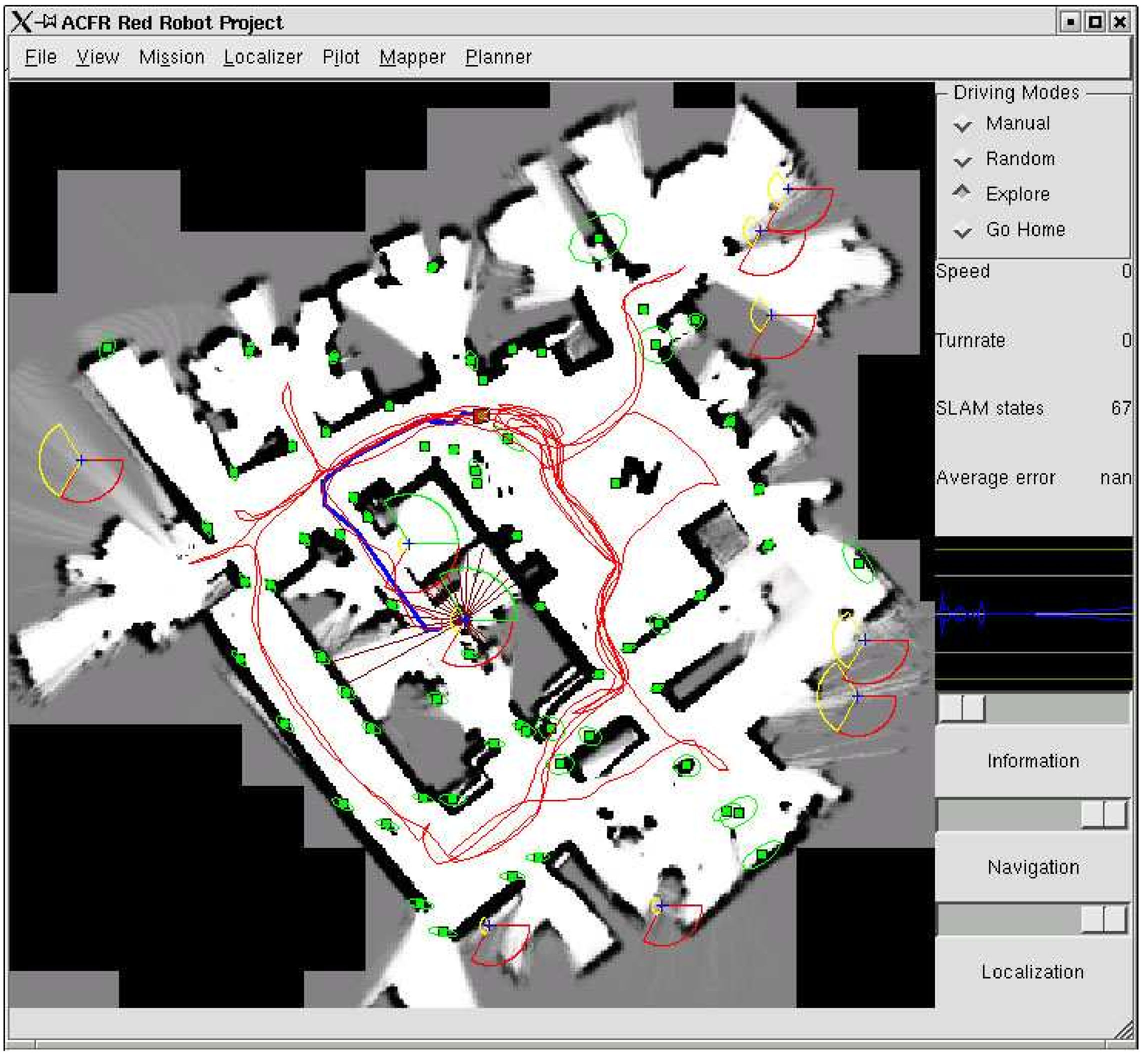


Figure 5.17: A Certainty Grid map autonomously constructed by a single mobile robot.

areas. As a result an acceptable level of localization quality is maintained.

The exploration experiment described in this section demonstrated application of the CG version of the BDDF algorithm to an individual mobile robot. In terms of ASN, the transition from isolated to coordinated decision making is trivially accomplished by enabling the inter-NODE synchronization (demonstrated in Section 4.4.5). In practice, coordinated exploration was not performed for lack of suitable multi-platform localization algorithm.

5.5 Summary

This chapter described how the task of decision making is performed in ASN. First, a general framework for decentralized Bayesian decision making was presented. Similar to the information fusion case, several primitive operations were identified which define the decision making algorithm of ASN called Bayesian Decentralized Decision Making (BDDM) algorithm. Well known algorithms for the Gaussian representation have been shown to implement various aspects of the BDDM algorithm. For the first time, experimental validation of decentralized coordinated platform control was performed.

Decision making in the context of CG map building led to the development of the Integrated Exploration algorithm. A novel feature of this algorithm is in the inclusion of the utility of localizability into the total utility for choosing destinations. It was shown experimentally that when the localizability utility is taken into account, the platform avoids territory where localization is expected to be poor. As a result, an acceptable level of localization quality is maintained and higher quality maps can be obtained. Although demonstrated in an environment with artificially placed navigation beacons, the same approach is applicable in uninstrumented environment as long as the expected localization quality can be predicted in advance. This exploration algorithm is readily extendible to multi-platform case in coordinated and, possibly, cooperative modes.

This chapter concludes presentation of ASN algorithms. The next chapter describes the concrete implementation of the ASN framework used in performing experiments described in this thesis.

Chapter 6

Implementation

6.1 Introduction

This chapter focuses on the implementation aspect of ASN. Two developments from Chapter 3 serve as departure points: the Execution architecture presented in Section 3.8 and the logical breakdown of the system into canonical components described in Section 3.6. Using these as a foundation, this chapter describes the software and hardware used in implementing the current version of ASN.

Section 6.2 describes the hardware used in the experiments including sensors, actuators, and two types of operator stations: stationary desktop computers and mobile hand-held devices. All experiments were conducted indoors, using both mobile and stationary platforms.

Software is described in Section 6.3. The description contains a list of components currently implemented in ASN and used in the experiments mentioned in the thesis. The User Interface plays a special role in the design and debugging of a large distributed system. Component communication, deployment, and configuration are also discussed.

Examples of typical platforms and networks are listed in Section 6.4. The examples serve to illustrate how the modular approach of ASN is applied to real-life situations. Some of the platforms appearing here were used in the experiments which are not described in the thesis. In addition to the networks actually implemented in ASN, hypothetical cases are considered drawn from the literature.

6.2 Hardware

The hardware description in this section starts with platforms, proceeds to sensors and actuators, and finishes with details of operator stations.

6.2.1 Platforms

Commercial indoor robots Pioneer 2 shown in Figure 6.1(a) were used as mobile platforms. Standard Intel Pentium III boards were used as on-board computers.

Stationary sensor platforms were custom-built. On-board processors included low-power Intel Pentium III on micro boards and PC-104 form factor. The units housing lasers could be deployed independently as well as on the Pioneer robots.

Standard wired and wireless Ethernet was used to connect the platforms into a network.

6.2.2 Sensors and Actuators

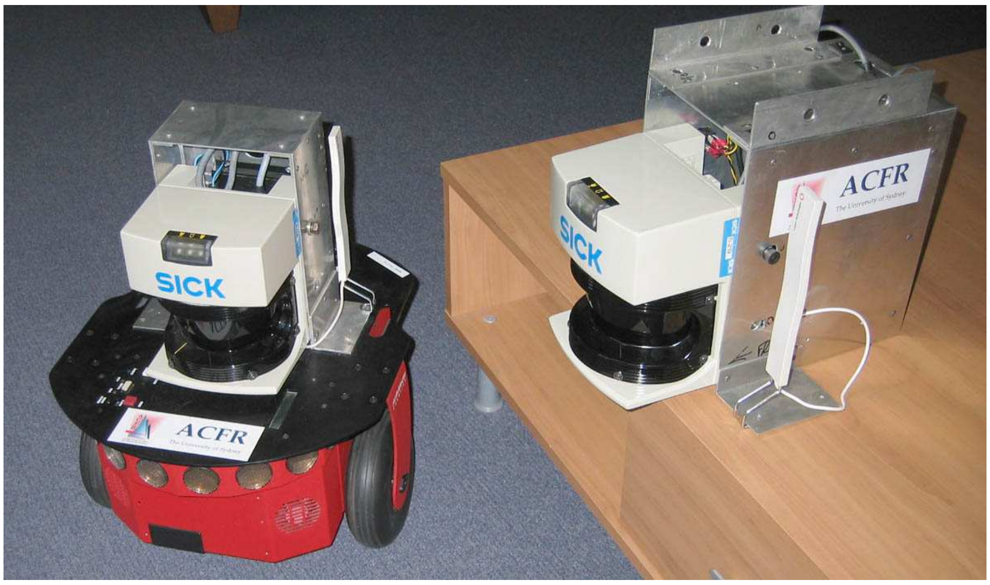
The most commonly used sensors were SICK LMS-291 laser range finders with a scan angle of 180 degrees. The lasers were configured to low-range (8 m) high-resolution operating mode more suitable for indoor applications. Scan frequency was limited to 5 Hz which was considered adequate for static and slow-moving environments.

A ring of range-finding sonars built into the Pioneer robots were used in some experiments.

Three types of cameras were used: high quality SONY DFW-VL500 FireWire, pan-tilt-zoom analog SONY EVI-D100P, and low-cost Creative NX Pro USB webcams. All three types are shown in Figure 6.1(b).

The wheeled base of the Pioneer 2 robot shown in Figure 6.1(a) was the most commonly used actuator. Motion control is achieved by specifying forward speed and the rate of turn. A built-in micro controller closes a low-level feedback loop from optical encoders to the motor drive.

Pan-tilt actuators in the base of the camera shown in Figure 6.1(b) were used in some experiments. Motion control is achieved by specifying the desired pan and tilt angles. A built-in micro-controller commands the base to move to the specified attitude.


(a)

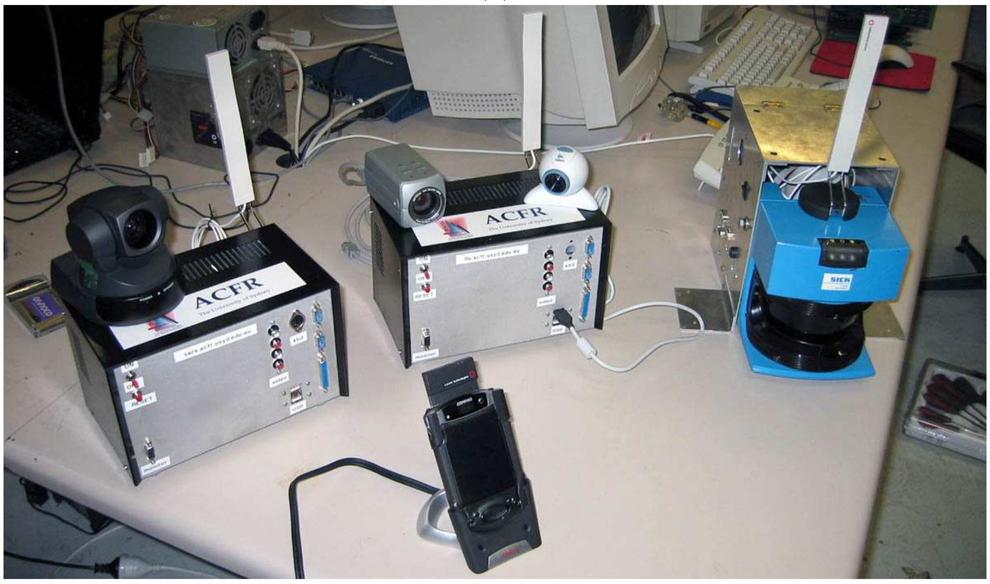

(b)

Figure 6.1: Mobile and stationary indoor sensor platforms: (a) mobile Pioneer platform and stationary laser sensor, and (b) custom-built PC-104 boxes used with video cameras.

6.2.3 Operator Stations

Operator stations serve as hardware access points for connecting to the ASN. Three types of operator stations were used: desktops, laptops, and handheld computers. The difference is in the available computing power, communication type, screen size, and input methods. Smaller less capable devices were useful only in the final stages of development and deployment.[1]

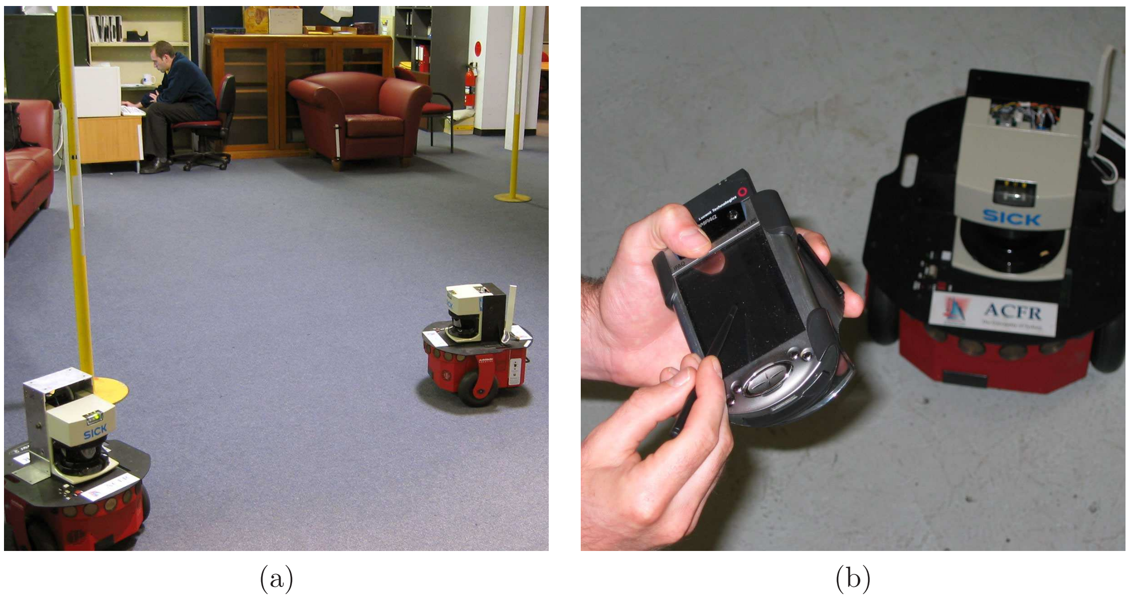


Figure 6.2: Two types of operator stations: (a) stationary version on a desktop computer and (b) a mobile one on a hand-held computing device.

An example of using UI running on a desktop computer is shown in Figure 6.2(a). An example of the handheld device in operation is shown in Figure 6.2(b). The mobile UI was executed on a Hewlett Packard iPaq 3870 running the Linux operating system. The connection to the network is wireless.

6.3 Software

All software is executed on the Linux operating system. All software components run as separate applications and communicate with each other using a common communication library. Communication is supported using different mechanisms: on the same processor and networked, both wireless and connected by a physical communication bus. Inter-component communication is described in Section 6.3.2. Execution and deployment is discussed in Section 6.3.4.

[1]The newly available trackpad devices with full-size screens, large batteries, and stylus-based input is a promising form factor for this application. These computers were not used in the experiments because the Linux support was lacking at the time.

Hardware abstraction is an important aspect of modularity and, in the context of indoor robots, it is achieved by using Player [63, 176], an open source device server developed at the University of Southern California. Player is used to access the Pioneer mobile robot, the SICK laser, and the SONY pan-tilt-zoom camera.

6.3.1 Component Implementation

ASN is a *component framework*[2] implemented as a set of C++ libraries. The system consists of approximately 50 classes implemented in 50,000 lines of source code. Roughly one third of the code implements the base classes which provide generic functionality (partial implementation) and infrastructure. The rest of the source is devoted to the implementation of specific components types.

At the time of writing, the ASN is comprised of 21 components: two NODEs, five FRAMEs, ten SENSORs, two PLANNERs, and two UIs. The twelve components mentioned in this thesis are listed and briefly described below.

Gauss Point Node. (`PointNode`) A NODE component for Gaussian point feature representation. It implements information fusion algorithms described in Section 4.3.

Certainty Grid Node. (`OgNode`) A NODE component for CG feature representation. It implements information fusion algorithms described in Section 4.4.4.

Pioneer Frame. (`Pioneer`) A FRAME component for use with Pioneer mobile robots. Localization of the mobile platforms was performed using beacon-based navigation. Light-reflecting strips placed around the office served as beacons. The beacon map was obtained *a priori* with a single Pioneer robot in teleoperation mode and executing the Constrained Local Sub-Map version of the SLAM algorithm [240]. For safe locomotion and obstacle avoidance, the VFH+ algorithm [230] was implemented.

Box Frame. (`Box`) A simple FRAME component for use with stationary platforms. The platform position was either supplied on startup (measured *a priori*) or estimated on line using the approach described in [23].

Laser to Point Sensor. (`Laser2Point`) A SENSOR component which converts laser range-bearing scans to point features with Gaussian position uncertainty. Range uncertainty of the

[2]Component framework is "a set of interfaces and rules of interactions that govern how the components 'plugged into' the framework may interact" [224].

SICK laser is approximately Gaussian with standard deviation of 5 mm. In order to obtain non-trivial trajectories in the experiments described in Section 5.3.3, range uncertainty was artificially increased to make observations almost bearing only.

Laser to Occupancy Grid Sensor. (`Laser2Og`) A SENSOR component which converts laser range-bearing scans to likelihoods of indoor space traversability. Occupancy Grid is a Certainty Grid which contains the probability of occupancy of the environment. CG techniques for line-of-sight range-bearing sensors are well established in the literature [47, 48].

Laser to Motion Grid Sensor. (`Laser2Motion`) A SENSOR component which converts laser range-bearing scans to likelihoods of moving objects in the environment. Motion Grid is a Certainty Grid which contains the probability of motion in the environment. Simple scan-differencing technique was used to detect a moving object. Only stationary platforms were used in these experiments so the more difficult problem of subtracting ego-motion was not encountered.

Vision to Point Sensor. (`Vision2Point`) A SENSOR component which converts visual observations to certain point features with Gaussian position uncertainty. The algorithm is limited to tracking human motion [24]. It uses a combination of background subtraction, frame differencing, and floor interception to estimate the position of the feet of the human in the global cartesian frame of reference.

Vision to Motion Grid Sensor. (`Vision2Motion`) A SENSOR component which converts visual observations to likelihoods of moving objects in the environment. The algorithm is limited to tracking human motion and is described in [22].

Point Target Information Surfing Planner. (`SurfController`) A PLANNER component which generates robot motion commands to maximize instantaneous information gain about environmental feature represented by Gaussian points. The algorithm is described in [71] and reviewed in Appendix C.5.

Occupancy Grid Exploration Planner. (`OgExplore`) A PLANNER component which generates robot motion commands to maximize the expected value comprised of the utilities of information gain, locomotion, and localization quality. The utilities for exploration of an indoor environment using an CG map are described in Section 5.4.3. The component also implements the Frontier exploration algorithm [242] and the Navigation Function path planning algorithm [118].

Desktop Graphical User Interface (GuiDesktop). A UI component for operating and debugging an ASN system. Figure 6.3 shows two screen-shots of the desktop version of the ASN GUI. Two views are available: the world view and the network view. In world view mode, the GUI provides environmental information in two representations: point features and Certainty Grid. An example of the CG display is shown in Figure 6.3(a). The GUI allowed operator input in two modalities: entering point feature information as described in Section 4.3.3 and direct teleoperation of Pioneer platforms. In network view, the GUI compiled information on all network components within communication range and assembled the global network topology. An example of network topology view is shown in Figure 6.3(b).

The handheld version of the GUI is similar to the desktop one, with minor changes to account for smaller screen area and the fact that the stylus is the only mode of information input.

The experience of implementing and using ASN showed that a Graphical User Interface is critical in the development stage of a large distributed system. The process of debugging is more difficult in a distributed environment and a GUI provides a convenient access to the state of individual components regardless of their physical location. The ability to record and playback all incoming messages was also a feature which was found to be very useful.

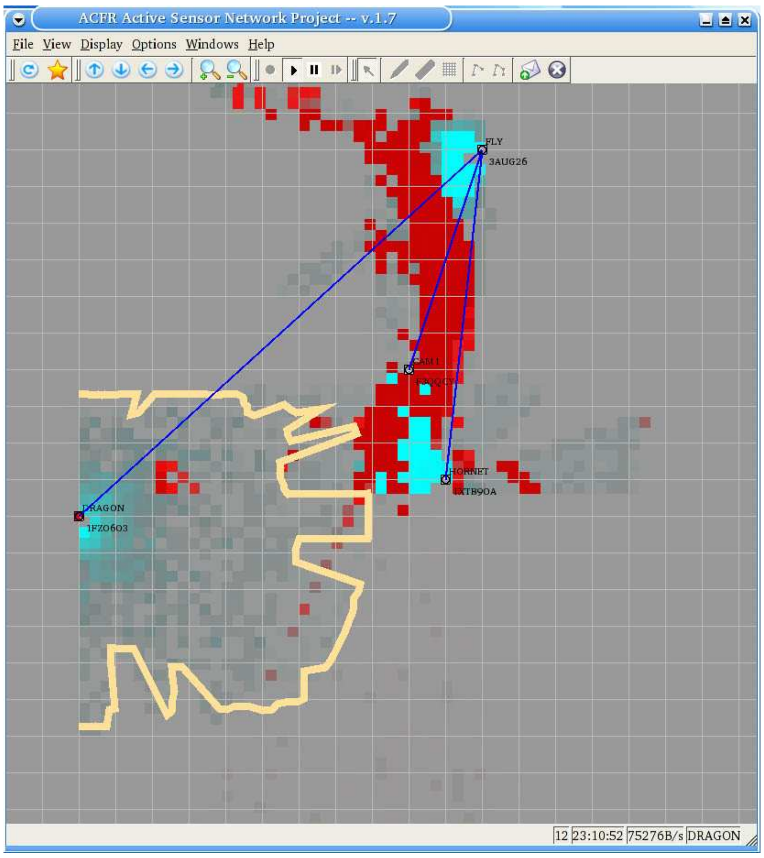
6.3.2 Inter-Component Communication

All ASN interfaces are implemented as a set of messages. Message format is known to all components and does not change. Message classes are responsible for their own serialization.

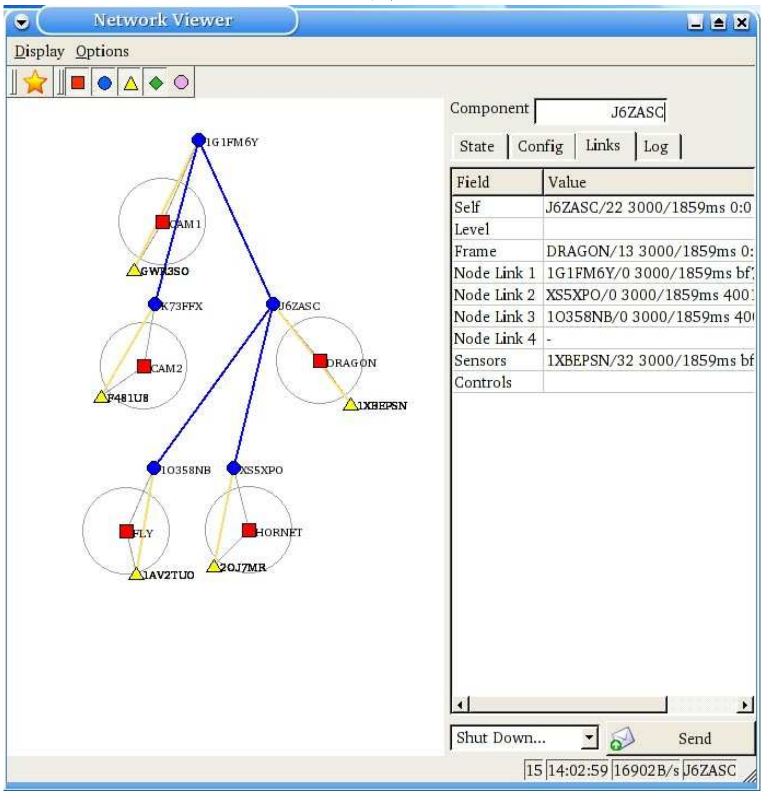
In terms of distribution scope, ASN, similar to [220], identifies three communication types listed below in order of increasing distribution scope:

1. *Component–Component.* Addressed component-to-component communication forms the backbone of our system. This communication type is used to implement the *Fusing*, *Linkable*, *Informed*, and other interfaces.

2. *Component–Neighborhood.* Local distribution is implemented with broadcast which may be further constrained by the use of communication channels. Certain types of


ACFR Active Sensor Network Project -- v.1.7
File View Display Options Windows Help
FLY
3AUG26
CAM1
HORNET
1XTB9OA
DRAGON
1FZO603
12 23:10:52 75276B/s DRAGON

(a)



Component J6ZASC

Field	Value
Self	J6ZASC/22 3000/1859ms 0:0
Level	
Frame	DRAGON/13 3000/1859ms 0:
Node Link 1	1G1FM6Y/0 3000/1859ms bf
Node Link 2	XS5XPO/0 3000/1859ms 400
Node Link 3	1O358NB/0 3000/1859ms 40
Node Link 4	-
Sensors	1XBEPSN/32 3000/1859ms bf
Controls	

(b)

Figure 6.3: The Graphical User Interface. The operator has a choice of (a) the world view and (b) the network view. Connection details for the selected component are shown on the right. These were often useful for debugging.

components may sign up for channels organized by information type. This communication type is used for system configuration.

3. *Component–Network.* Global distribution is facilitated by NODEs. This communication type is used to propagate environment information and team control priorities.

The first two communication types are implemented with a communication library used by ASN. The last type, responsible for global information distribution, is implemented on the ASN component level.

Both addressed and broadcast communication types are facilitated by a single custom library which works as follows. Each host runs a daemon process which is responsible for message routing. When a new component process starts up, it registers with the daemon and subscribes to receive messages on predetermined channels which depend on the component type. The daemon at each host maintains a table of other daemons within communication range and the channels to which they are subscribing. This *infrastructure communication* [227] is performed using a custom low-bandwidth broadcast protocol.

The daemon is indistinguishable from any other communicating process except that it provides a "gateway" (a forwarding facility) to remote hosts. Messages are sent from one component process to another in the following manner. If the destination component is local, the process simply writes its message contents to the addressees shared memory. If the message is inter-host, the process first writes to the local daemons shared memory which then sends an addressed UDP packet to the daemon on the receiving host which, in turn, forwards it to the appropriate process. An unaddressed (broadcast) message is sent to all daemons in the neighborhood subscribing to the particular channel who distribute it to all subscribing processes on the same host.

Due to the algorithm-intensive nature of information fusion and control, the inter-component communication link can be subject to effective optimization. Figure 6.4 shows a link between a component and its *remote proxy.*[3] As an illustration, consider the case when components A and B are executed on different platforms (hosts) and have to communicate over a radio link. Component B needs some information from component A. It expects

[3]"A remote proxy provides a local representation for an object in a different address space" [61]. In the context of ASN, a SENSOR, instead of talking to the FRAME which may be on a different host, may talk to a local proxy which provides access to the same localization information.

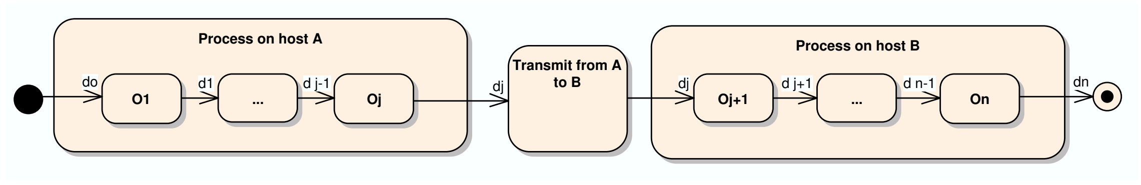


Figure 6.4: A chain of operations between a component and its remote proxy. Operations O_1 through O_n transform the data into potentially different representations d_0 through d_n. Operations O_1 through O_j are performed on host A. The data is then transmitted to host B in d_j representation, and the rest of the operations are performed on host B. The relative costs of computation and communication determine the optimal choice of communication step j.

to receive it in the form of representation d_n. Component A has the data available in the form of d_0. There are n operations which must be applied to the data, each one potentially changing its representation. After a certain number of operations the data must be transmitted to component B. The code for performing all operations is available at both the actual component and its proxy. The total cost of delivering data d_n to component B is

$$C_{AB} = \sum_{j=0}^{i} C_A^{\text{comp}}(O_j) + C_{AB}^{\text{comm}}(d_i) + \sum_{j=i+1}^{n} C_B^{\text{comp}}(O_j). \tag{6.1}$$

It is a simple optimization problem to find the step i for data transmission which minimizes the cost C_{AB}. The computational and communication costs can be either specified or estimated at run time.

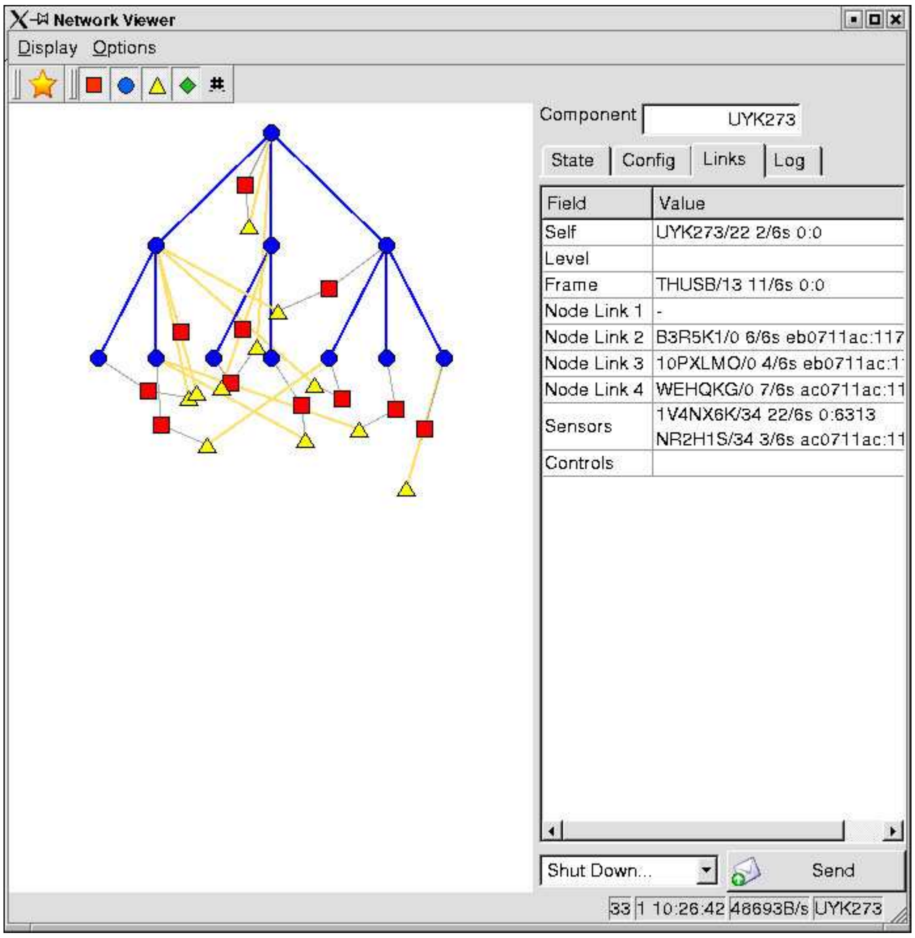
For example, the data size of a range-bearing contact-based observation is small compared to an observation likelihood defined on a fine spatial grid. It may be beneficial, in this case, for a SENSOR to communicate the observation z_i^k to the NODE where the SENSOR-proxy will transform the observation into a likelihood b_{li}^k. As a counter example, the data size for a vision-based system typically outweighs the corresponding likelihood in any representation.

6.3.3 System Configuration

Service discovery is performed at run time using a ping-and-reply mechanism. The algorithm is decentralized on the component level, i.e., each component is responsible for looking for the services it requires. Depending on the interface, selection from multiple replies is made randomly or based on information supplied with the replies.

Addressing between components is implemented using unique 32-bit identifiers based on the IP address of the host and the process ID of the component.

After a service is initialized, the link is monitored and the process of discovery and connection is repeated if the service is lost.



Component UYK273

Field	Value
Self	UYK273/22 2/6s 0:0
Level	
Frame	THUSB/13 11/6s 0:0
Node Link 1	-
Node Link 2	B3R5K1/0 6/6s eb0711ac:117
Node Link 3	10PXLMO/0 4/6s eb0711ac:1
Node Link 4	WEHQKG/0 7/6s ac0711ac:11
Sensors	1V4NX6K/34 22/6s 0:6313 NR2H1S/34 3/6s ac0711ac:11
Controls	

33 1 10:26:42 48693B/s UYK273

Figure 6.5: Topological view of an experimental sensor network described in Section 4.4.5. The diagram shows FRAMEs (□), NODEs (○), and SENSORs (△). A total of 33 components are currently connected into a network. The NODEs clearly form a tree network.

Figure 6.5 shows the network topology of an experimental system from Section 4.4.5. The system ran for several days with individual platforms turning on and off.

6.3.4 Component Deployment

Deployment refers to the mapping of executable software units to physical hardware. In general, distributed architectures offer an opportunity to implement a flexible and adaptive deployment strategy although such considerations are beyond the topic of this thesis. Perhaps the main issue in component deployment, and the only one discussed here, is the quality of communication links between components. Communication has many aspects, including reliability, delays, energy consumption, and security. An obvious guideline is to deploy components in such a way that the mission-critical bandwidth-hungry links correspond to reliable high-bandwidth low-power communication lines. Such lines of communication typically exist *inside* platforms.

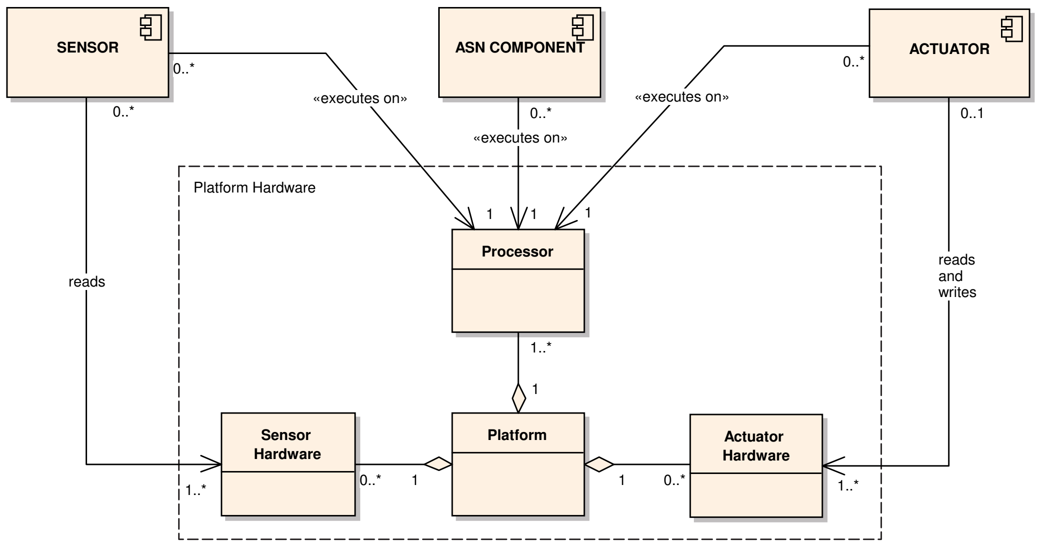


Figure 6.6: Relation between the platform hardware and software. Compare with the diagram of platform hardware in Figure 2.5.

This is why the term *platform* remains useful despite the fact that the ASN framework is fully distributed and, in principle, any component can be deployed anywhere as long the network connection is maintained. The "membrane" separating the platform from the outside world makes the environment inside it more sheltered. This is true for mechanical systems: stable offsets of sensors compared to the motion of the platform. This is also true for computer systems: low communication costs, short delays, and low probability of failure compared to the inter-platform communication. To exploit this benefit, it is advisable to keep some component interactions local.[4] The relationship between platform hardware and software is illustrated in Figure 6.6. It is apparent that, if the rule of local hardware access is adopted, then at least two types of components, SENSOR and ACTUATOR, must be deployed at the platform to which the hardware is attached.

Having a large number of components running on physically distributed hardware presents a management problem which is not fully resolved in ASN. The partial solution involves a combination of Unix scripts which start up the specified components and background daemons which monitor the status of the running software. Software distribution, versioning, etc. remain real problems.[5]

6.3.5 Configuration and Deployment in the Experiments

This section describes deployment strategies for the two experiments described throughout this thesis.

Experiments with active target localization were described in Section 5.3.3 and the overall view of the system was shown in Figure 4.15. The experiment involves five platforms: two mobile Pioneer robots, a stationary sensor module, and two operator stations, one stationary and one mobile.

Figure 6.7 shows a deployment diagram for this system. Each of the two Pioneers executes a custom component VEHICLE (see Figure 3.38) and three canonical components: a Gaussian point NODE, a laser-to-point SENSOR, and an information surfing PLANNER. By executing all four components internally, the mobile platform is made relatively independent from the rest of the team. Even if all external communication links are lost, the robot will continue to control its wheel base, sense the environment, maintain its local belief, and make local decisions.

The stationary sensor unit *fly* executes the same SENSOR, but its Box FRAME is much simpler. A SENSOR executes on *fly* for local access to its laser. *Fly's* processing power is insufficient to run NODE so its SENSOR connects to a remote NODE which happens to be on the ground station *base*.

[4] To quote the designers of Hive "Things have places!" [155].

[5] These problems are not unique to robotics and all commercial component models address them to a certain degree [131]. In a research setting, an interesting application of *mobile code* approach to solving the problem of software distribution is described in [155].

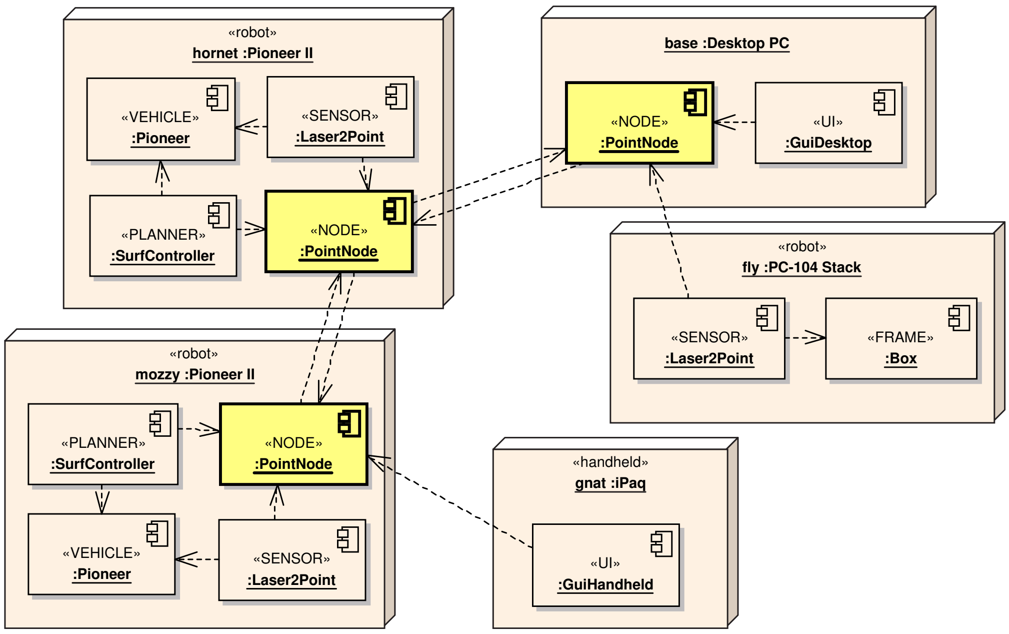


Figure 6.7: Deployment diagram for the active localization experiments in Section 5.3.3. Each UML Node element represents a host, a physical piece of equipment (e.g. a desktop PC called *base*).

Platform *base* has no local sensors of its own. It executes a desktop version of the GUI software and an instance of Gauss point NODE. The mobile operators station *gnat* can only run a handheld version of GUI which connects to a remote NODE on *mozzy*.

Experiments with motion mapping using Certainty Grid representation were described in Section 4.4.5 and the physical locations of sensor platforms was shown in Figure 4.26. The experiment ran for several days and involved up to 39 components on up to 11 platforms.[6]

Figure 6.8 shows a partial deployment diagram with six platforms. Compared with the previous example, the platforms here are more encapsulated. Most of inter-component connections are between NODEs using *Linkable* interface. The only exception is desktop UI on laptop *cocoa* connected to a remote NODE on *hornet*.

The composition of the software running on all Pioneers is the same. Each executes three components: a CG NODE, a Box FRAME, and a laser-to-motion SENSOR.

[6]As a side note, a comparison of the two sets of experiments shows the improvement in reliability due to a year of implementation work. The first version, used in the point target localization experiments, ran for several hours before something would go wrong. This version, a year later, ran for several days at a time.

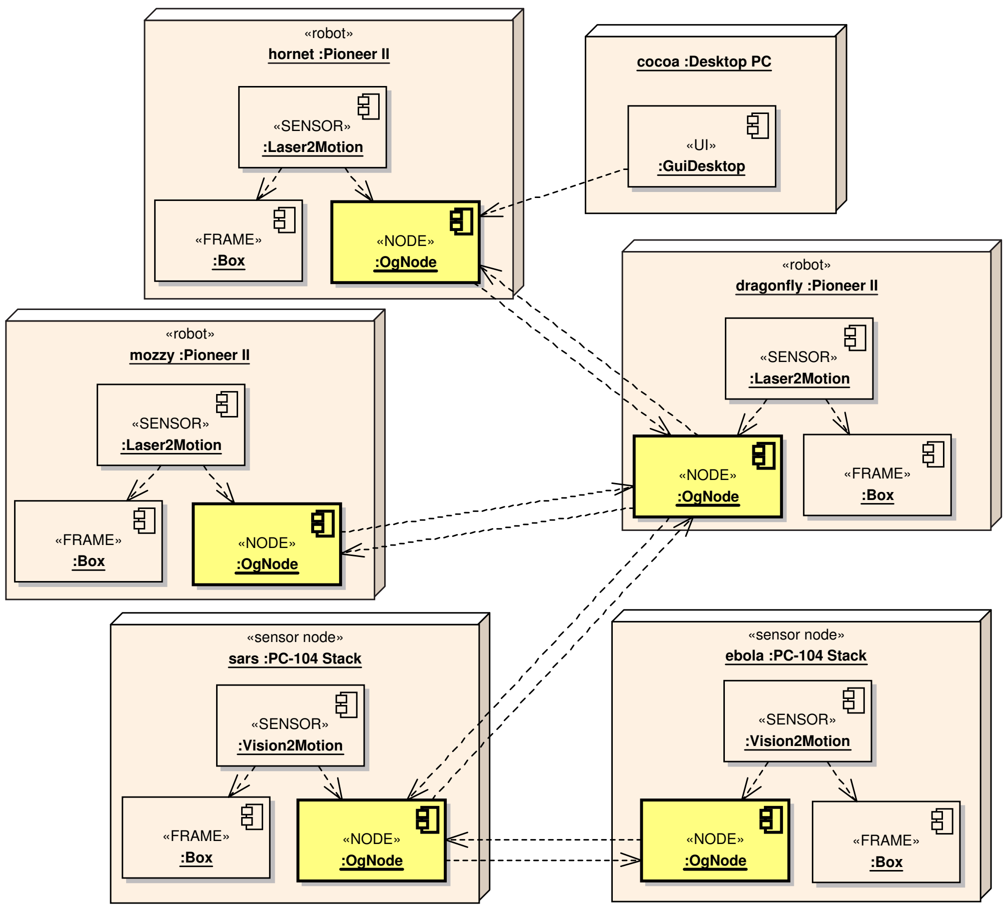


Figure 6.8: Partial deployment diagram for the motion mapping experiments in Section 4.4.5.

Each of the two stationary sensor platforms is equipped with a video camera and executes three components: a Box FRAME, an CG NODE, and a vision-to-motion SENSOR which extracts motion information from video frames.

Figure 6.9 shows a deployment diagram for the exploration experiment described in Section 5.4.3. The experiment involved a single platform and an operator station. All software necessary for exploration was executed on the platform itself.

These experiments demonstrate the flexibility in component deployment offered by the ASN framework. It seems that the biggest benefit to be gained from this flexibility is in the upcoming multi-host robotic platforms. For example, the on-board computing resources of the CMU entry in the 2004 Grand Challenge competition included 8 hosts with the total

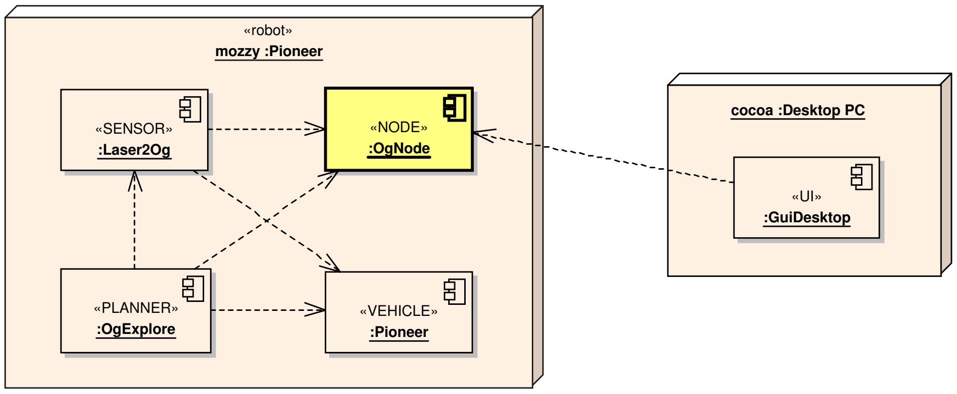


Figure 6.9: Deployment diagram for the exploration experiments in Section 5.4.3.

of 14 processors [231]. All hosts were interconnected with 1 Gb Ethernet internally and a wireless external link was available. This configuration is ideal for deploying a decentralized ASN-based system.

6.4 Examples

This section considers several examples of platforms and sensor networks. The objective is to illustrate the modular breakdown and component interaction in the ASN framework. Most of the platforms used in the examples were actually implemented using the ASN framework and used in the experiments described throughout the thesis. Network examples are both experimental and hypothetical showing how the ASN approach could be applied to the systems described in the literature.

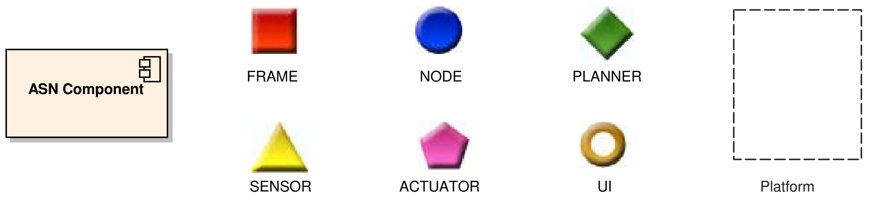
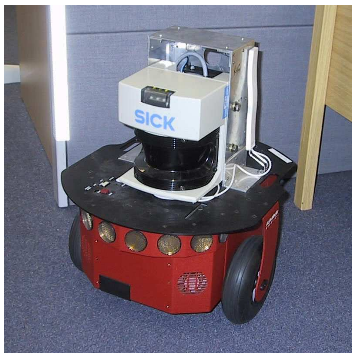


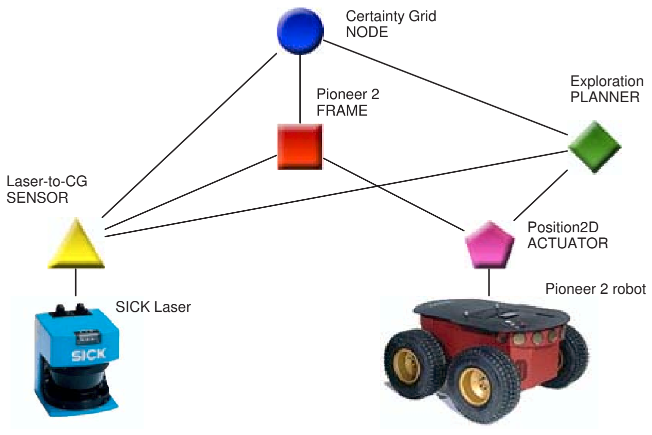
Figure 6.10: Symbols used for the canonical component types. The standard UML symbol for a component is shown for comparison on the left. The platform boundaries are marked with a dash line.

The instances of platform and network architectures are described with UML component diagrams. To improve readability of component diagrams, each canonical component type is coded with a distinctive shape and color. Figure 6.10 shows the legend of graphical symbols.

6.4.1 Typical Platforms

To illustrate how the modular breakdown applies to typical robotic sensing platforms several examples are considered. All platforms except for the UAV have been implemented using the ASN framework.


(a)



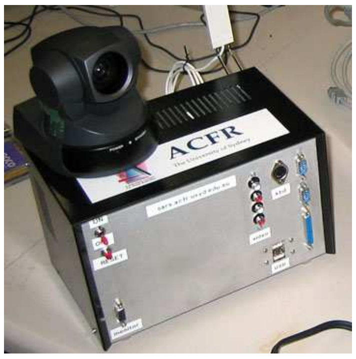
(b)

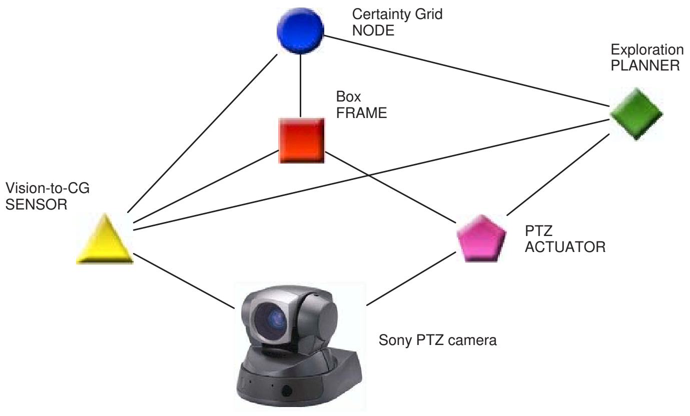
Figure 6.11: A Pioneer 2 indoor robot with a mounted laser range finder: (a) the physical view and (b) the component diagram.

A Pioneer 2 with a mounted laser range finder is a typical indoor mobile robotic platform. Figure 6.11 shows its physical view and the component diagram. The ACTUATOR component represents the mobility of a wheeled base. The SENSOR converts raw observations of the laser range finder to the chosen feature representation, in this case Certainty Grid map.

A Sony pan-tilt-zoom camera is a simple articulated sensor platform. Figure 6.12 shows its physical view and the component diagram.

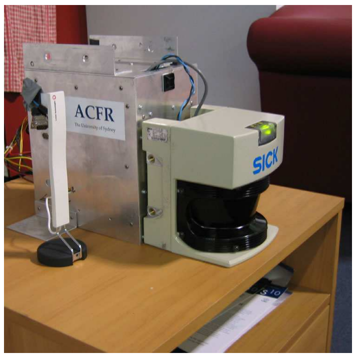
A SICK laser range finder is used as an example of a stationary uncontrolled platform. Figure 6.13 shows its physical view and the component diagram. No PLANNERs are present. The processor is capable of running the NODE.

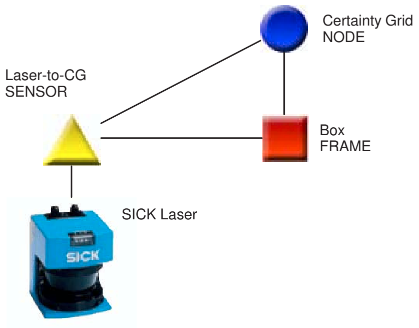

(a)



(b)

Figure 6.12: A PTZ camera with a processing unit: (a) the physical view and (b) the component diagram.


(a)

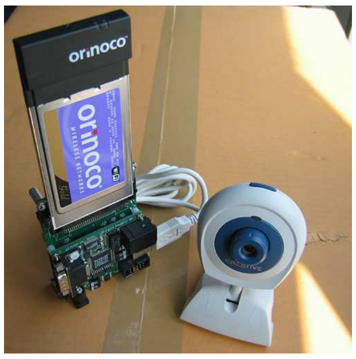


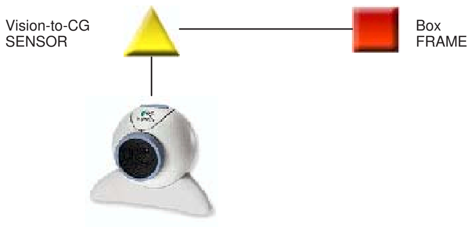
(b)

Figure 6.13: A stationary laser range finder with a processing unit: (a) the physical view and (b) the component diagram.

A low-cost web camera and a small-size processing board is used as an example of a stationary low-power platform. Figure 6.14 shows its physical view and the component diagram. No PLANNERs are present. The computer does not have enough processing power to execute the NODE component so the observations are sent to a remote NODE.

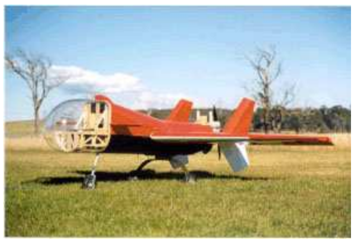
Brumby Mk.III UAV is used as an example of an advanced flying platform. Figure 6.15 shows its physical view and the hypothetical component diagram. Each sensor payload includes a processor and both execute the NODE component.

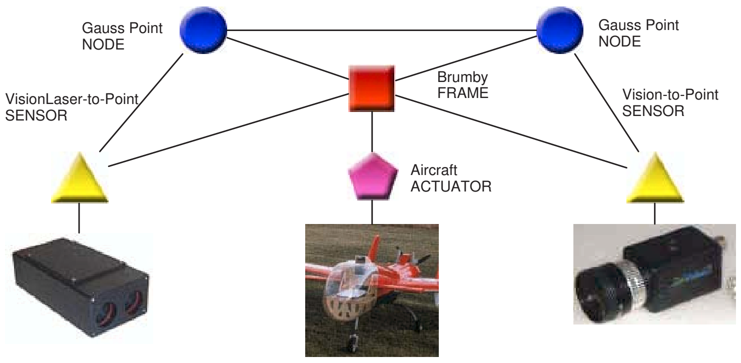

(a)



(b)

Figure 6.14: A stationary webcam with a processing unit: (a) the physical view and (b) the component diagram.


(a)



(b)

Figure 6.15: An UAV Brumby Mk.III with a camera and a laser: (a) the physical view and (b) a possible component diagram.

6.4.2 Typical Networks

This section considers several typical systems prevalent in the literature and shows how they can be implemented using ASN framework. In particular, centralized and hierarchical system can be implemented as special cases of a fully decentralized architecture. Some of the algorithms underlying the actual systems are implemented within the Bayesian framework, some are not. Casting these literature examples into the ASN mold does not make claims about the algorithmic compatibility, the diagrams simply refer to connectivity and information flow.

FireMapper is a commercial airborne fire-tracking system [87]. It employs centralized in-

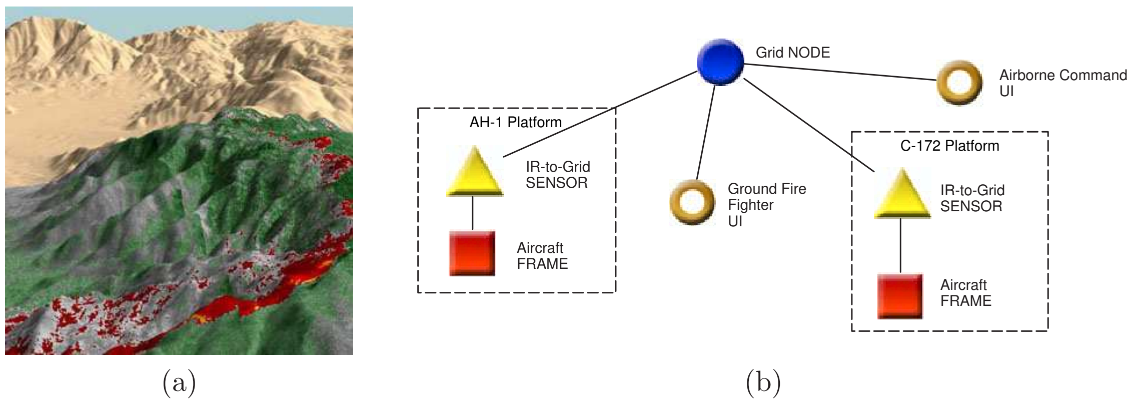


Figure 6.16: Component diagram for a hypothetical fire tracking system: (a) the composite view of the environment [87] and (b) the component diagram of a possible ASN implementation.

formation fusion of observations and decentralized human control of sensing platforms. Figure 6.16 shows a hypothetical implementation of the system using the ASN framework.

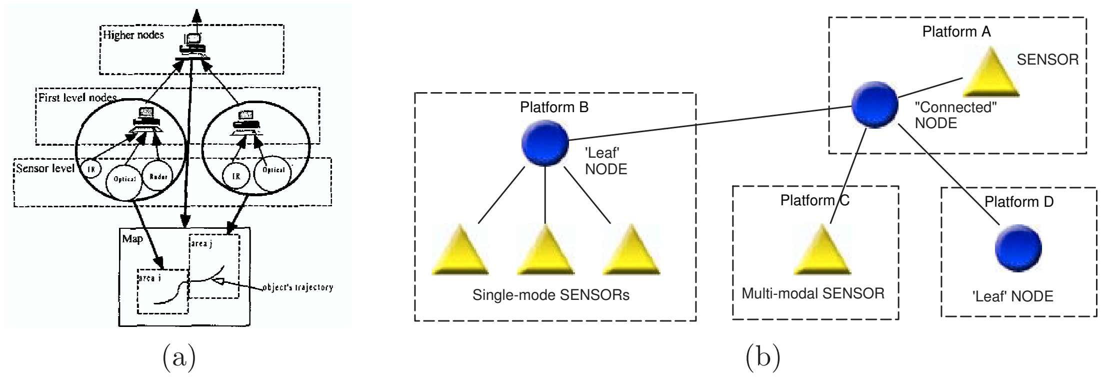
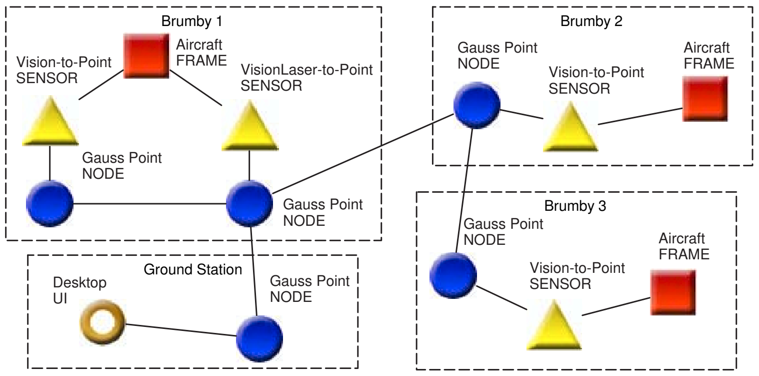


Figure 6.17: Hierarchical sensor network (a) a hierarchical system [58] and (b) the component diagram of a possible ASN implementation.

Hierarchical information fusion architectures are often proposed and implemented [28, 66, 202]. Cerpa et al. [28] describe a system with “inexpensive sensor nodes” or “tags” which send information to “larger more capable nodes”. The former can be implemented as ASN SENSORs and the latter as ASN NODEs. Foresti and Snidaro [58] describe a hierarchical system in which sensors connect to nodes at different hierarchical levels. The information fusion method is not Bayesian. Figure 6.17 shows the original architecture structure and the corresponding ASN interpretation. The figure shows four platform types: a fully-featured NODE with local sensors and multiple NODE-to-NODE links (platform A), a “leaf” NODE which connects local SENSORs and a single external NODE (platform B), a SENSOR without a NODE (platform C), and a “leaf” NODE without local SENSORs (platform D).

(a)



(b)

Figure 6.18: An airborne sensor network developed as part of the ANSER project [221]: (a) the aircraft and (b) a possible component diagram of the sensor network. Component diagrams for individual platforms are shown in Figure 6.15.

Fixed-wing Unmanned Aerial Vehicles were applied to tracking ground targets [221]. Up to three of them were flown simultaneously during a three year flight program with up to six fusion nodes (precursors to ASN NODEs) operating simultaneously and in real time. Figure 6.18 shows the aircraft used in the experiments and a hypothetical ASN implementation of the sensor network.

6.5 Summary

This chapter has described the experimental implementation of the ASN framework. The framework has proven to be flexible enough to accommodate a variety of system topologies, platform and sensor hardware, and environment representations.

The implementation described in this Chapter was created strictly for research purposes and many functions are missing. For example, the issues of timing, deployment, versioning, and many others are not addressed.

Nevertheless, the practical experience with the system validated the fundamental premise of the approach: modularity and decentralization. Modular component-based design made it possible to develop components individually and to add capability incrementally. Several developers could easily write code simultaneously when working on different components which is an important feature for large projects. For example, the information fusion task in the experiments with localizing point features (Section 4.3.3) was developed first and decision making functionality was added later (Section 5.3.3). While this property could be expected from several implementation strategies, the CBSE approach made it particularly simple.

Decentralization allowed easy addition of network capacity by simply deploying more platforms. For example, in the experiments with motion mapping (Section 4.4.5), the area coverage was gradually increased from a few square meters to nearly the entire floor of roughly 400 square meters. Such growth could not have been possible with a centralized system without careful planning.

Chapter 7

Conclusions

This chapter summarizes the principle contributions of this thesis and provides suggestions for future research directions.

7.1 Summary of Contributions

This thesis is concerned with the development of distributed information gathering systems. The principle contribution of the thesis is in the development of the ASN framework which includes architecture design, new algorithm formulation, and a concrete system implementation.

7.1.1 Probabilistic Problem Statement

The probabilistic reference model of the DIG problem is a significant contribution of this thesis. The reference model established the structure of the DIG problem and helped to categorize the solution space by listing a number of options available to the designer. A quantitative survey of related work suggested three types of systems commonly used in solving the DIG problem: Micro Sensor Networks, Macro Sensor Networks and Multi-Robot Systems. This taxonomic breakdown elucidated the relationship between common technical requirements and the solution strategies widely used in the field. This thesis has presented a system which satisfies the requirements for Macro Sensor Networks.

7.1.2 Solution Framework

The main contribution of this work is in designing a comprehensive framework, called Active Sensor Network (ASN), which addresses the tasks of information fusion, decision making, system configuration, and user interaction in MSN. The framework is designed to accommodate a large number of mobile autonomous heterogeneous platforms acting as a team to collect information about the environment.

The framework has been described from three complementary points of view: architecture, algorithms, and implementation. This should not obscure the fact that ASN is a unified comprehensive solution. Several guiding principles were identified and applied consistently across the board and at all levels of the framework. There is a certain value in such holistic treatment which goes beyond the individual contributions in the architecture, algorithms, and implementation which are listed below. Such an approach is necessary to ensure that no aspect of the solution becomes a performance bottleneck as the system scales up in size.

7.1.3 Architecture

A key aspect of the ASN framework is its architecture which is a necessary consideration for a large distributed system. Architectural design was performed at multiple levels, starting with the Meta-architecture which established the three guiding design principles: decentralization, modularity, and locality of interactions.

The Conceptual and Logical architectures partitioned the system into a set of manageable elements. The Logical architecture, in particular, defined a detailed architecture specification, including canonical component definition and inter-component service interfaces. The six representative component types were described in detail and used in the implementation of experimental systems in this thesis. The Algorithmic and Execution architectures ensure that the Meta-architecture principles are adhered to on the algorithmic and implementation levels respectively.

The architecture design has emphasized interactions between components rather than the definition of the components themselves. The architecture specified a small set of interfaces sufficient to implement a wide range of information gathering systems. By placing emphasis on the interfaces, the ASN architecture is made flexible and extendable, with custom components extending and refining the functionality provided by canonical components.

7.1.4 Algorithms

The main contribution of this work in the area of algorithms is in the extension of the Decentralized Data Fusion algorithm to the general Bayesian case. The thesis presented the Bayesian Decentralized Data Fusion (BDDF) algorithm formulated for environment features represented by a general probability density function. Several specific representations were also considered: Gaussian, discrete, and the Certainty Grid map. Well known algorithms for these representations have been shown to implement various aspects of the BDDF algorithm.

To make the process of applying the BDDF algorithm to a specific representation more straightforward, the DDF primitives were identified. These are a small set of information processing operations, which constitute the ASN information fusion algorithm. It is hoped that a clear general formulation will stimulate extensions to efficient non-parametric representations of arbitrary distributions. Stating the algorithm in Bayesian terms has an additional practical advantage of allowing a generic software implementation.

The Decentralized Certainty Grid algorithm presented in this thesis is a novel application of the DDF approach to a well known Certainty Grid environment representation. CG maps are popular in indoor robotics because they are easy to construct and use for path planning and navigation. Extending the DDF approach to this environment representation is an important practical contribution.

This thesis also makes a contribution in extending earlier efforts in decentralized sensor management and control to the general Bayesian case. Similar to the information fusion case, several primitive operations were identified which define the decision making algorithm of ASN called Bayesian Decentralized Decision Making (BDDM) algorithm. Well known algorithms for the Gaussian representation have been shown to implement various aspects of the BDDM algorithm. Additionally, the experimental results of decentralized coordinated information-theoretic control of sensing platforms are first of a kind.

Decision making in the context of building CG maps lead to the development of the Integrated Exploration algorithm. A novel feature of this algorithm is in the inclusion of the utility of localizability into the total utility for choosing destinations. A localizability metric for a particular Simultaneous Localization and Mapping (SLAM) algorithm has been derived. It was shown experimentally that when the localizability utility is taken into account, the platform avoids territory where localization is expected to be poor. As a result, an acceptable level of localization quality is maintained and higher quality maps can be obtained. Although demonstrated in an environment with artificially placed navigation beacons, the same approach is applicable in uninstrumented environment as long as the expected localization quality can be predicted in advance. This exploration algorithm is readily extendible to multi-platform case in coordinated and, possibly, cooperative modes.

7.1.5 Implementation

As part of this thesis work, a concrete implementation of the Active Sensor Network (ASN) framework has been developed and tested in an indoor environment. Practical experience with the system has validated the fundamental principles of the ASN approach. Modular component-based design made it possible to develop components independently and add capability incrementally. Decentralization allowed easy addition of network capacity by simply deploying more platforms.

Perhaps, the best measure of success of the implementation is reflected in its contribution to the work of other researchers. In fact, only half of the 21 ASN components have been implemented by the author. The rest were added by the fellow researchers who found the framework suitable for their needs and useful as a tool. As a result, the ASN framework, in addition to the experiments described in this thesis, was fully or partially used in experiments on human-robot interactions [108], human motion tracking [24], sensor self-localization [23], cooperative search [19], and in an interactive art installation [237].

7.2 Future Research

Within the confines of the DIG problem, the ASN offers a fairly complete framework for building large distributed information gathering systems. A significant additional effort is required to implement all of the described functionality and test the architectural decisions in various situations. Beyond that, several of numerous research directions are explored in this section.

7.2.1 Architecture

One of the most exciting properties of modular architectures is the possibility of system composition. The simulated and practical examples in this thesis provided numerous examples *manual* off-line system composition using canonical components. An important research direction is in *automatic* system composition which was not considered here at all. It can take many forms, both off- and on-line.

Off-line composition of both hardware and software given a task description was briefly discussed in Section 2.5 and illustrated in Example 2.1. It requires a quantitative metric of performance which is readily available in the task of information gathering. The pioneering work by Henderson et al. as part of the Logical Sensor Systems framework [40, 78] can be revisited in the context of decentralized multi-platform environment. This approach may require platforms to exchange quantitative models of their capabilities. Standardizing on the model types and formats may be the biggest difficulty in this area.

It is even more interesting to explore on-line adaptive system composition. In general, the self-configuration aspect of an ASN has not been addressed fully in this work. The concept of adaptive degree of decentralization was mentioned in Section 3.5.3 from the point of view of changing network topology. A more advanced concept is on-demand *activation* and *de-activation* of components as part of system's self-configuration. Changes in the makeup of system components executed on a particular platform would essentially change the role which the platform plays.

What is described above may be called Systems Engineering. The field of Systems of Systems Engineering (SoSE) is concerned with "complex systems which become a subsystem within a larger system of systems" [110]. This field straddles the line between engineering and management which is indicative of the types of problems presented by real-life large-scale systems.

7.2.2 Information Fusion Algorithms

The list of representations considered in this thesis was rather limited. Future research may focus on more expressive and efficient representations of general probability distributions, e.g. functional approximations, multi-Gaussian [188], particle filters [191], kernel and other non-parametric methods.

Maintaining a tree topology in a large network requires significant effort and, at present, is inefficient. New algorithms and experiments are needed. For many systems, the optimality offered by using Channel Filters in an acyclic network is unnecessary but more research is needed to find conservative fusion of general probability densities.

Interaction between human operators and the robotic sensor network is practically valuable and, at present, poorly understood. The few existing approaches are very different from the one in this thesis and are not closely tied to the information about the environment (c.f. [226]). Incorporation of human observation and knowledge into the network belief may require different sensor models from those currently used.

The fundamental difference in how the environment is perceived by humans and the current generation of robotic sensors suggests the use of multi-level environment representation. This would require making automatic inferences from the quantitative picture built by the SN. The use of Bayesian Network representation may also be explored in this context.

7.2.3 Decision Making Algorithms

In the discussion of decentralized decision making, this thesis made a fundamental assumption that the information quality, or more precisely the information gain resulting from an action, is the primary component of action utility. This allowed an approximation of utility fusion by information fusion.

A very interesting and well-developed approach which deals with full utility coupling can be found in the field of scheduling [126]. It utilizes the *task structure* defined as "a precise quantitative definition of the degree of achievement in terms of measurable characteristics such as solution quality and time" and which is essentially a utility. The task structure also contains information on task relationships that indicate how basic actions or abstract task achievement effect task characteristics (e.g. quality and time) elsewhere in the task structure. Therefore the task structure encapsulates the information about action coupling, which is a requirement for cooperation. The difficulty is that for mobile and geographically distributed platforms obtaining the task relationships is not easy.

Many research opportunities lie in the area of cooperative human-network decision making. Cooperative decision making as described in Section 5.2 can be easily extended to interaction between operators and robotic platforms. The operators and the PLANNER components would in this case as as peers. It would be interesting to investigate the possibility of defining a sliding scale of authority exercised by the operator to span the gap between peer-to-peer interaction and teleoperation. This approach is being actively pursued in the field of Human Robot Interaction under the name of *adjustable autonomy* [56]. This work has so far been limited to one-to-one human-robot communication.

7.2.4 Implementation

Having a large number of components executed on physically distributed hardware presents a management problem which has not been resolved in ASN. Software distribution, versioning, execution, status monitoring, etc. remain real problems and should be addressed comprehensively as part of the component model.

The ASN experience highlighted the practical complexity of implementing a heterogenous distributed system. Some of the factors contributing to this complexity include potential hardware, operating system, and programming language diversity; issues related to data serialization and type safety; name resolution, addressing, routing, and many other. Many of these issues are addressed in dedicated distributed middleware packages. One of several common middleware systems, CORBA [73], hides the location of remote objects, simplifying the application's interactions with these remote objects by allowing all operations to appear local. The ASN framework has essentially implemented its own middleware. The advantage of following the custom middleware route is in the ability to customize its feature to the specifics of the problem but, as the ASN experience has shown, the effort in doing so may be significant. It seems likely that the field of robotics will eventually adopt general distributed middleware products just like it is common today to use standard off-the-shelf components like processors, operating systems, and development tools.

The component-based approach advocated in this thesis can be pushed down from the sensor network level to the individual platforms. A good implementation would allow the low-level local components interact seamlessly with the decentralized ASN components. Several open source projects with similar goals are currently underway (c.f. [175]) and could be used in future systems solving the DIG problem.

7.3 Summary

Frameworks similar in scope to the one described in this thesis will certainly be needed to bring to life scalable, autonomous, and cooperative information gathering systems. This thesis has made a significant contribution to the art and science of designing such frameworks.

Appendix A

Quantitative Survey of Sensor Network Literature

This appendix provides details of the quantitative literature survey described in Chapter 2. The reported and estimated data is listed in Section A.1. The derived metrics are explained and listed in Section A.2. Projects are classified into SN types in Section A.3.

A.1 Data

Both experimental and selected simulated sensor networks are included. Most sources do not report detailed system metrics. In such cases, the numbers had to be estimated from secondary data or from figures and must therefore be treated as rough estimates.

Table A.1 lists primary infrastructure data about the project. These numbers are often directly quoted in the literature.

Table A.2 lists primary objectives data about the project. In most cases, the area is estimated from plots and maps. For robotics applications, a typical mission duration is often quoted. For simulated sensor networks, mission duration was equated to the often-stated objectives for long-lived embedded systems.

Table A.3 lists solution types used in each of the projects.

No.	Project	Unit mass	Team size	Active platforms	Hetero platforms
1	GEMS study [143]	2.5 mg	10^{10}		
2	*Berkley sim* [201]	100 g	25	✓	✓
3	Xerox PARC [51]	180 g	76		
4	*Umass sim* [90]	180 g	21		
5	*UCLA energy sim* [214]	180 g	45		
6	29 Palms [45]	500 g	61		
7	ASN (CG) [22]	1 kg	13		✓
8	UMN Scouts [194]	200 g	11	✓	✓
9	CMU millibots [66]	500 g	5	✓	✓
10	*USC sim* [93]	5 kg	100	✓	
11	Centibots [114]	5 kg	42–66	✓	✓
12	RoboCup [43]	10 kg	6	✓	
13	CMU mapping [211]	5 kg	3	✓	✓
14	ASN (Gauss) [137]	10 kg	3	✓	✓
15	ANSER [221]	45 kg	3	✓	
16	*UAV SAR sim* [21]	45 kg	6–10	✓	
17	FireMapper [87]	1000 kg	2	✓	✓

Table A.1: Problem metrics. Project names are shown in *italics* if only simulated data is reported.

No.	Project	Mission time	Area	High accuracy	Low latency
1	GEMS study [143]	1 week	–	–	–
2	*Berkley sim* [201]	year	540 m^2	–	–
3	Xerox PARC [51]	1 week	2.3 m^2	–	–
4	*Umass sim* [90]	days	–	–	–
5	*UCLA energy sim* [214]	TBD	density	–	–
6	29 Palms [45]	days	?	–	–
7	ASN (CG) [22]	days	600 m^2	✓	–
8	UMN Scouts [194]	hours	1400 m^2	–	✓
9	CMU millibots [66]	1 hour	30 m^2	–	–
10	*USC sim* [93]	5 min.	relative	–	–
11	Centibots [114]	30 min.	2790 m^2	✓	✓
12	RoboCup [43]	90 min.	7350 m^2	✓	✓
13	CMU mapping [211]	10 min.	890 m^2	✓	✓
14	ASN (Gauss) [137]	1 hour	100 m^2	✓	–
15	ANSER [221]	45 min.	2 km^2	✓	–
16	*UAV SAR sim* [21]	hours	50 km^2	✓	–
17	FireMapper [87]	days	$\sim km^2$	✓	–

Table A.2: Mission requirements.

No.	Project	Arch.	Info fusion	Utility	Policy selection
1	GEMS study [143]	?	?	–	–
2	*Berkley sim* [201]	dec.	–	–	–
3	Xerox PARC [51]	dec.	aggregation	–	–
4	*Umass sim* [90]	hierach.	deterministic	–	–
5	*UCLA energy sim* [214]	dec.	?	–	–
6	29 Palms [45]	?	CSP	–	–
7	ASN (CG) [22]	dec.	Bayes	–	–
8	UMN Scouts [194]	centr.	–	–	reactive
9	CMU millibots [66]	centr.	Bayes	–	teleop.
10	*USC sim* [93]	dec.	–	–	reactive
11	Centibots [114]	hierach.	Bayes	info gain	hierarch.
12	RoboCup [43]	centr.	Bayes	domain	domain
13	CMU mapping [211]	centr.	Bayes	info gain	central
14	ASN (Gauss) [137]	dec.	Bayes	info gain	dec.
15	ANSER [221]	dec.	Bayes	–	teleop.
16	*UAV SAR sim* [21]	dec.	Bayes	info gain	dec.
17	FireMapper [87]	centr.	deterministic	–	manual

Table A.3: Solution approaches.

A.2 Metrics

Sensor coverage. The majority of entries sense a 2D environment. Area coverage is an appropriate metric for such systems. For systems localizing or tracking discrete features, the ratio of observed to the total number of features is more appropriate

$$C_A = \frac{A_{observed}}{A_{total}}, \qquad C_F = \frac{F_{observed}}{F_{total}}.$$

The total area (number of features) is easier to estimate than the observed one. Information about sensor range can be used for the latter, but it can only be an estimate since, it depends on the environment and it is dynamic for mobile platforms.

In many micro sensor applications observations are very local. This is a rather special case of sampling networks. Rather than imply zero coverage, it is assumed that interpolation is adequate for the task and coverage is implicitly unity.

Platform mobility In terms of observing the environment, platform mobility may be approximated by *sensor mobility*: the ratio of dynamic space coverage to static sensor footprint

$$M_A = \frac{\sum A_s}{\sum a_s}, \qquad M_F = \frac{\sum N_f}{\sum n_f}.$$

No.	Project	Space coverage	Sensor mobility
1	GEMS study [143]	1.0	0
2	*Berkley sim* [201]	1.0	
3	Xerox PARC [51]	3.1	0
4	*Umass sim* [90]	3.65	0
5	*UCLA energy sim* [214]	?	0
6	29 Palms [45]	1.0	0
7	ASN (CG) [22]	0.5	0
8	UMN Scouts [194]	0.28	0.12
9	CMU millibots [66]	0.13	1e-3
10	*USC sim* [93]	3.0	0?
11	Centibots [114]	0.5	0.18
12	RoboCup [43]	1.0?	0.3?
13	CMU mapping [211]	0.13	0.20
14	ASN (Gauss points) [137]	0.2	0.09
15	ANSER [221]	2.7E-3	1.58
16	*UAV SAR sim* [21]	7.0E-4	0.087
17	FireMapper [87]	1.0e-2?	0.5?

Table A.4: Derived metrics.

Table A.4 lists derived metrics about the projects.

A.3 Classification

Table A.5 shows assignment of the projects to one of three sensor network types. Several project are hard to classify. In such cases, a question mark stands for the second possible classification. For example, the competing entries in the Centibots project [94, 114] have strong roots in Multi-Robot Systems in terms of application and solution approaches. The exploration and mapping tasks performed by the robots have a finite duration, typically under one hour. A limited number of robots (under 10) participated in this stage of the exercise. The second task is oriented towards long-term surveillance, redeployment, search. An unusually high number of robots involved at this stage (up to 66), requires solutions typical of Macro Sensor Networks.

No.	Project	μSN	MSN	MRS
1	GEMS study [143]	✓		
2	*Berkley sim* [201]	✓		
3	Xerox PARC [51]	✓		
4	*Umass sim* [90]	✓		
5	*UCLA energy sim* [214]	✓		
6	29 Palms [45]	✓		
7	ASN (CG) [22]		✓	
8	UMN Scouts [194]			✓
9	CMU millibots [66]			✓
10	*USC sim* [93]		✓	?
11	Centibots [114]		✓	?
12	RoboCup [43]			✓
13	CMU mapping [211]		?	✓
14	ASN (Gauss) [137]		?	✓
15	ANSER [221]			✓
16	*UAV SAR sim* [21]			✓
17	FireMapper [87]			✓

Table A.5: Sensor network type assignment.

Appendix B

Bayesian Algorithms

B.1 Bayesian Filtering

Following [219], nonlinear Bayesian filtering equations are derived for a single feature and a single sensing platform. Let $x^k = x(t_k)$ be feature state at time t_k. The feature state is unknown and needs to be estimated. Let $u^k = u(t_k)$ be control inputs applied to the feature at time t_k. A history of internal feature inputs up to time t_k may be stored as a sequence $U^k = \{u^1, u^2 \dots u^k\}$. Define the state transition model at time $t_k \geq 1$

$$G_k = P(x^k \mid x^{k-1}, u^k). \tag{B.1}$$

This definition assumes that the state has Markov property: state transition function depends only on the state of the feature at the time of transition.

Let $z^k = z(t_k)$ be an observation of the feature described by a state vector x^k taken at time t_k. A history of observations up to time t_k may be stored as a sequence $Z^k = \{z^1, z^2 \dots z^k\}$. Define the sensor likelihood function at time t_k

$$L_k = P(z^k \mid x^k). \tag{B.2}$$

This definition assumes that the sensor likelihood function depends only on the state of the feature at the time of the observation. This assumption is necessary to allow recursive computation of the posterior but is quite reasonable in practice.

The problem is formulated as follows: find the posterior probability at time t_k based on the initial information and the histories of inputs and observations

$$P(x^k \mid Z^k, U^k, x^0). \tag{B.3}$$

The solution uses the recursive form of Bayes' theorem

$$P(x^k \mid Z^k, x^0) = \frac{P(z^k = z \mid x^k) P(x^k \mid Z^{k-1}, x^0)}{P(z^k = z \mid Z^{k-1})}, \tag{B.4}$$

where the *prior* is predicted (recursively) based on the information available at time step t_{k-1}

$$P(x^k \mid Z^{k-1}, U^k, x^0) = \int P(x^k \mid x^{k-1}, u_i^k) P(x^{k-1} \mid Z^{k-1}, U^{k-1}, x^0)\, dx^{k-1}. \tag{B.5}$$

B.2 Bayesian Communication

Consider a sensing platform i whose belief about the feature state is described by Equation B.3. The information set Z_i^k denotes all information available to platform i, both locally observed and communicated. Omitting internal inputs and the initial prior for clarity, $P(x^k \mid Z_i^k)$ defines the posterior probability on the feature state given all locally available information.

If there are two communicating platforms i and j, with information sets Z_i^k and Z_j^k respectively, it is desirable to determine the common information set $Z_i^k \cup Z_j^k$ found as a union of the two local information sets. This may then be used to find a posterior distribution $P(x^k \mid Z_i^k \cup Z_j^k)$ for the state based on both sources of information. If the two information sets have no information in common $Z_i^k \cap Z_j^k = \varnothing$, then it is simple to construct the union of the sets from a sum. However if the two sets do have observations in common $Z_i^k \cap Z_j^k \neq \varnothing$ then key to constructing the union $Z_i^k \cup Z_j^k$ is finding the *common information* $Z_i^k \cap Z_j^k$.

The most general solution to the common information problem is in the form of Bayes theorem. Consider the interaction of pairs of platforms. For each communicating pair (i, j), the required probability is $P(x^k \mid Z_i^k \cup Z_j^k)$. Let the union of the individual observation information sets be partitioned into disjoint sets as

$$Z_i^k \cup Z_j^k = Z_{i\backslash j}^k \cup Z_{j\backslash i}^k \cup Z_{ij}^k, \tag{B.6}$$

where

$$Z_{i\backslash j}^k = Z_i^k \setminus Z_{ij}^k, \qquad Z_{j\backslash i}^k = Z_j^k \setminus Z_{ij}^k, \qquad Z_{ij}^k = Z_i^k \cap Z_j^k,$$

and where the notation $p\backslash r$ (the restriction operation) means elements of the set p excluding those elements that are also in set r. Note also that

$$Z_{i\backslash j}^k \cup Z_{ij}^k = Z_i^k, \qquad Z_{j\backslash i}^k \cup Z_{ij}^k = Z_j^k.$$

Then

$$P(Z_i^k \cup Z_j^k \mid x^k) = \frac{P(Z_i^k \mid x^k)P(Z_j^k \mid x^k)}{P(Z_i^k \cap Z_j^k \mid x^k)}. \tag{B.7}$$

Substituting into Bayes theorem and simplifying the result gives

$$\begin{aligned} P(x^k \mid Z_i^k \cup Z_j^k) &= \frac{P(Z_i^k \cup Z_j^k \mid x^k)}{P(Z_i^k \cup Z_j^k)} \\ &= \frac{1}{C}\frac{P(x^k \mid Z_i^k)P(x^k \mid Z_j^k)}{P(x^k \mid Z_i^k \cap Z_j^k)}, \end{aligned} \tag{B.8}$$

which states how to determine the *joint belief* given information from both platforms, $P(x^k \mid Z_i^k \cup Z_j^k)$, when the individuals beliefs, $P(x^k \mid Z_i^k)$ and $P(x^k \mid Z_j^k)$, and the information common to the two platforms, $P(x^k \mid Z_i^k \cap Z_j^k)$, are available.

An alternative expression for the joint belief is possible, expressed in terms of local belief and the *new information* contained in the external belief

$$P(x^k \mid Z_i^k \cup Z_j^k) = \frac{1}{C}P(x^k \mid Z_i^k)P(x^k \mid Z_j^k \setminus Z_i^k), \tag{B.9}$$

where the new information is found as

$$P(x^k \mid Z_j^k \setminus Z_i^k) = \frac{P(x^k \mid Z_j^k)}{P(x^k \mid Z_i^k \cap Z_j^k)}. \tag{B.10}$$

For the full derivation, refer to [67].

B.3 Information Measures

It was shown by Manyika [145] that the utility based on entropy of the belief satisfies the rationality axioms guaranteeing a preference ordering. Proof that the axioms imply the existence of a utility function can be found in [16]. An appropriate definition of utility is now provided by the log-likelihood

$$U(x) = \log P(x). \tag{B.11}$$

The expected utility is known as *entropy*, defined as follows

$$\mathcal{E}n(x) \overset{\triangle}{=} -E\left\{\log P(x)\right\} = \int_{-\infty}^{\infty} P(x)\, \log P(x)\, dx. \tag{B.12}$$

With the definitions of entropy, it is possible to write an "information form" of Bayes theorem. Taking logs and expectations of Equation B.4 with respect to both the state x and the observation z gives

$$\mathcal{E}n(x \mid z) = \mathcal{E}n(z \mid x) + \mathcal{E}n(x) - \mathcal{E}n(z). \tag{B.13}$$

This equation describes the change in entropy following an observation from a sensor modelled by the likelihood $L(z \mid x)$. Being able to describe changes in entropy leads naturally to asking an important question: which of the possible observations is the most informative? The objective is to take an observation which *on average* will reduce entropy of the posterior distribution the most. This question may be answered using the concept of mutual information $\mathcal{I}(x, z)$ defined as follows

$$\begin{aligned} \mathcal{I}(x,z) &= \mathcal{E}n(x) + \mathcal{E}n(z) - \mathcal{E}n(x,z) \\ &= \mathcal{E}n(x) - \overline{\mathcal{E}n}(x \mid z) \\ &= \mathcal{E}n(z) - \overline{\mathcal{E}n}(z \mid x). \end{aligned} \tag{B.14}$$

Mutual information is an *a priori* measure of the information to be gained through observation. It measures the average reduction in uncertainty about x that results from learning the value of z or vice versa. The relationship between the joint, marginal, and conditional entropies and mutual information is illustrated in Figure B.1.

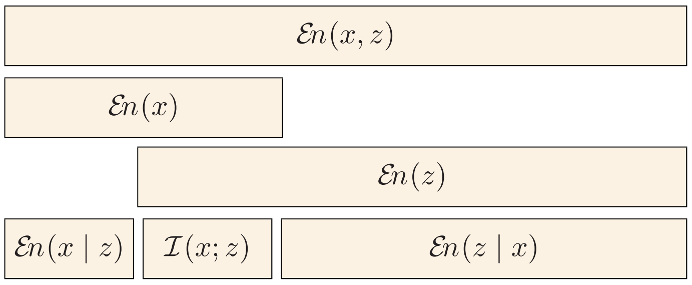


Figure B.1: Relation between joint entropy, marginal entropy, conditional entropy and mutual information (from [136]).

Alternatively, the mutual information $\mathcal{I}(x,z)$ can be written as the expectation of the ratio of probability distributions

$$\mathcal{I}(x,z) = -E\left\{\log\frac{P(x \mid z)}{P(x)}\right\} = -E\left\{\log\frac{P(z \mid x)}{P(z)}\right\}. \tag{B.15}$$

Equation B.14 is an *a priori* measure of the “compression” of the probability mass caused by an observation.

Appendix C

Algorithms for Gaussian Point Features

C.1 Model

Consider a linear (or linearized) discrete-time system described by the equation

$$\mathbf{x}^k = \mathbf{F}_k \mathbf{x}^{k-1} + \mathbf{B}_{ak}\mathbf{a}^k + \mathbf{B}_{wk}\mathbf{w}^k, \tag{C.1}$$

where $\mathbf{x}^k = [\mathbf{x}_f^k, \mathbf{x}_{pi}^k]^T$ is the combined feature and platform state vector at time t_k, $\mathbf{F}_k$ is the combined state transition matrix from time t_{k-1} to t_k[1], $\mathbf{B}_{ak}$ is the influence matrix for actions, $\mathbf{B}_{wk}$ is the influence matrix for process noise, and $\mathbf{w}^k$ is the associated process noise modelled as an uncorrelated white sequence with $E\left\{\mathbf{w}(i)\mathbf{w}^T(j)\right\} = \delta_{ij}\mathbf{Q}(i)$. The system uses a sensor with a linear (or linearized) observation model

$$\mathbf{z}^k = \mathbf{H}_k \mathbf{x}^k + \mathbf{v}^k, \tag{C.2}$$

where $\mathbf{z}^k$ is the vector of observations made at time t_k, $\mathbf{H}_k$ is the observation matrix, and $\mathbf{v}^k$ is the associated observation noise modelled as an uncorrelated white sequence with $E\left\{\mathbf{v}(i)\mathbf{v}^T(j)\right\} = \delta_{ij}\mathbf{R}(i)$.

[1]Note a slight change in notation in this section: for matrices, the time step is shown as a subscript to allow room for transcripts and inverses in the superscript.

The state estimate $\hat{\mathbf{x}}(i \mid j)$ and its variance $\mathbf{P}(i \mid j)$ at time t_i, given observations up to time t_j, are defined as

$$\hat{\mathbf{x}}(i \mid j) \triangleq E\left\{\mathbf{x}(t_i) \mid \mathbf{Z}^j\right\}, \quad \mathbf{P}(i \mid j) \triangleq E\left\{(\mathbf{x}(t_i) - \hat{\mathbf{x}}(i \mid j))(\mathbf{x}(t_i) - \hat{\mathbf{x}}(i \mid j))^T \mid \mathbf{Z}^j\right\}. \quad \text{(C.3)}$$

The mean $\hat{\mathbf{x}}(i \mid j)$ and the covariance matrix $\mathbf{P}(i \mid j)$ are state-space variables.

C.2 Information Filter Algorithm

The Information filter is equivalent to the Kalman filter [9, 150] but is more suitable for decentralization. Two "information" variables, the information vector $\hat{\mathbf{y}}(\cdot \mid \cdot)$ and information matrix $\mathbf{Y}(\cdot \mid \cdot)$, are defined as

$$\hat{\mathbf{y}}(i \mid j) \triangleq \mathbf{P}^{-1}(i \mid j)\hat{\mathbf{x}}(i \mid j), \qquad \mathbf{Y}(i \mid j) \triangleq \mathbf{P}^{-1}(i \mid j). \quad \text{(C.4)}$$

In addition, the information associated with an observation at time t_k is expressed in terms of two new variables, the information gain vector $\mathbf{i}(\cdot)$ and information gain matrix $\mathbf{I}(\cdot)$

$$\mathbf{i}(k) \triangleq \mathbf{H}_k^T\mathbf{R}_k^{-1}\mathbf{z}(k), \qquad \mathbf{I}(k) \triangleq \mathbf{H}_k^T\mathbf{R}_k^{-1}\mathbf{H}_k. \quad \text{(C.5)}$$

The prediction and update stages of the Information filter can now be stated.

Prediction. The prediction stage is more complicated than in state-space form

$$\begin{aligned} \hat{\mathbf{y}}(k \mid k-1) &= [1 - \mathbf{M}_k\mathbf{B}_{wk}\mathbf{\Sigma}_k^{-1}\mathbf{B}_{wk}^T]\mathbf{F}_k^{-T}\hat{\mathbf{y}}(k-1 \mid k-1) + \mathbf{Y}(k \mid k-1)\mathbf{B}_{uk}\mathbf{u}(k) \\ \mathbf{Y}(k \mid k-1) &= \mathbf{M}_k - \mathbf{M}_k\mathbf{B}_{wk}\mathbf{\Sigma}_k^{-1}\mathbf{B}_{wk}^T\mathbf{M}_k. \end{aligned} \quad \text{(C.6)}$$

where $\mathbf{\Sigma}_k = \mathbf{B}_{wk}^T\mathbf{M}_k\mathbf{B}_{wk} + \mathbf{Q}_k^{-1}$, $\mathbf{M}_k = \mathbf{F}_k^{-T}\mathbf{Y}(k-1 \mid k-1)\mathbf{F}_k^{-1}$. Equation C.6 is the Information filter equivalent of Equation B.5.

Belief update. The update state is simplified compared to the state-space form

$$\begin{aligned} \hat{\mathbf{y}}(k \mid k) &= \hat{\mathbf{y}}(k \mid k-1) + \mathbf{i}(k) \\ \mathbf{Y}(k \mid k) &= \mathbf{Y}(k \mid k-1) + \mathbf{I}(k). \end{aligned} \quad \text{(C.7)}$$

Equation C.7 is the Information filter equivalent of Equation B.4.

Channel update. Two beliefs are combined by adding the information in the individual beliefs and subtracting the information they have in common

$$\begin{aligned}\hat{\mathbf{y}}_{i\cup j}(k \mid k) &= \hat{\mathbf{y}}_i(k \mid k) + \hat{\mathbf{y}}_j(k \mid k) - \hat{\mathbf{y}}_{i\cap j}(k \mid k-1) \\ \mathbf{Y}_{i\cup j}(k \mid k) &= \mathbf{Y}_i(k \mid k) + \mathbf{Y}_j(k \mid k) - \mathbf{Y}_{i\cap j}(k \mid k-1). \qquad \text{(C.8)}\end{aligned}$$

Equation C.8 is the Information filter equivalent of Equation B.8.

The full derivation and discussion of the algorithm can be found in [67, 68] and more recently in [163, 164].

Data Association with Information Gate. The information gate [55] can be used for data association with the Information filter. It is the information equivalent of the state space innovation gate. The test is

$$\mathbf{v}_2^T(k)\mathbf{B}_2(k)\mathbf{v}_2(k) < \gamma, \qquad \text{(C.9)}$$

where γ is selected from a Chi-Squared distribution with the number of degrees of freedom of the observation and

$$\begin{aligned}\mathbf{B}_1(k) &= \mathbf{I}(k)[\mathbf{I}(k) + \mathbf{Y}^{-1}(k \mid k-1)]\mathbf{I}(k) \\ \mathbf{B}_2(k) &= \mathbf{H}_k^T[\mathbf{H}_k\mathbf{B}_1(k)\mathbf{H}_k^T]\mathbf{H}_k \\ \mathbf{I}_1(k) &= \mathbf{H}_k^T[\mathbf{H}_k\mathbf{I}(k)\mathbf{H}^T(k)]\mathbf{H}_k \\ \mathbf{v}_2(k) &= \mathbf{I}(k)[\mathbf{I}_1(k)\mathbf{i}(k) - \mathbf{Y}^{-1}(k \mid k-1)\hat{\mathbf{y}}(k \mid k-1)]. \qquad \text{(C.10)}\end{aligned}$$

A complete derivation of the information gate can be found [55] and in an Appendix of [163].

C.3 Covariance Intersect Algorithm

If two available estimates are known to be independent, then the Kalman filter update gives the combined estimate

$$\begin{aligned}\hat{\mathbf{y}}_c(k \mid k) &= \hat{\mathbf{y}}_a(k \mid k) + \hat{\mathbf{y}}_b(k \mid k) \\ \mathbf{Y}_c(k \mid k) &= \mathbf{Y}_a(k \mid k) + \mathbf{Y}_b(k \mid k). \end{aligned} \tag{C.11}$$

If the assumption of independence is not valid than this estimate double counts information and is too optimistic. Instead the CI algorithm computes an updated covariance matrix as a convex combination of the two initial covariance matrices in the form

$$\begin{aligned}\hat{\mathbf{y}}_c(k \mid k) &= \omega\hat{\mathbf{y}}_a(k \mid k) + (1-\omega)\hat{\mathbf{y}}_b(k \mid k) \\ \mathbf{Y}_c(k \mid k) &= \omega\mathbf{Y}_a(k \mid k) + (1-\omega)\mathbf{Y}_b(k \mid k), \end{aligned} \tag{C.12}$$

where ω is selected based on a heuristic. The most commonly used heuristic is to select ω which minimizes the determinant of the resulting covariance matrix. Figure C.1 shows an example of application of the algorithm.

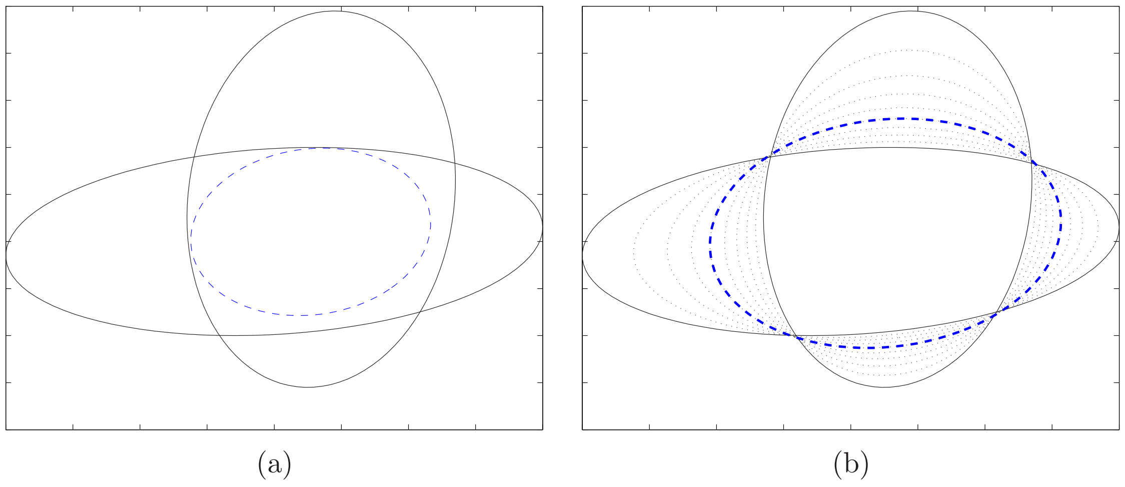


Figure C.1: Covariance Intersect updates for two estimates: (a) optimal fusion when common information is known and (b) conservative fusion when it is unknown. A set of possible estimates is shown, with the least uncertain one highlighted.

A proof and a discussion of the conservative nature of the CI algorithm can be found in [105].

C.4 Information Measures

Of particular interest in the following is the entropic information on an n-dimensional state $\mathbf{x}$ modelled by a Gaussian with a covariance matrix $\mathbf{P}$ [37]

$$\mathcal{E}n(\mathbf{x}) = \frac{1}{2}\log\left[(2\pi e^{n})|\mathbf{P}|\right]. \tag{C.13}$$

Entropy is a measure of the volume enclosed by the covariance matrix and consequently the compactness of the probability distribution. Equation C.13 is the Gaussian version of Equation B.12.

For a feature described by Gaussian uncertainty, mutual information is calculated as

$$\mathcal{I}(x^k, z^k) = -\frac{1}{2}\log\frac{|\mathbf{P}(k \mid k)|}{|\mathbf{P}(k-1 \mid k-1)|} = \frac{1}{2}\log\frac{|\mathbf{Y}(k \mid k)|}{|\mathbf{Y}(k-1 \mid k-1)|}, \tag{C.14}$$

where $\mathbf{P}$ is the covariance and $\mathbf{Y}$ is the information matrices respectively. Equation C.14 is the Gaussian version of Equation B.15.

C.5 Information Surfing Algorithm

Planning with zero look ahead provides a special case in coordinated multi vehicle control. This can be used to form simple approximate solutions where the sensor platforms are directed by the dynamics of the mutual information rate gradient field. The Fisher information evolution in continuous linearized filtering is given by the information form of the Ricatti equation

$$\dot{\mathbf{Y}}(t) = -\mathbf{F}_k\mathbf{Y}(t) - \mathbf{F}_k^T\mathbf{Y}(t) - \mathbf{Y}\mathbf{B}_{wk}\mathbf{Q}_k\mathbf{B}_{wk}^T\mathbf{Y}(t) + \sum_{i=1}^{n}\mathbf{H}_{ik}^T\mathbf{R}_{ik}^{-1}\mathbf{H}_{ik}. \tag{C.15}$$

Using matrix calculus identities, the instantaneous rate of change of entropy, or *mutual information rate* is

$$\mathcal{I}(t) = \frac{1}{2}\frac{d}{dt}\log|\mathbf{Y}(t)| = \frac{1}{2}\mathrm{trace}(\mathbf{Y}^{-1}(t)\dot{\mathbf{Y}}(t)). \tag{C.16}$$

Equation C.16 represents a time varying vector field. It shows that the mutual information rate is determined by the current Fisher information and Fisher information rate from Equation C.15. The gradient of the mutual information rate relates changes in the system state and control to changes in the rate of change of entropic information. Since $\mathbf{Y}(t)$ is not an explicit function of $\mathbf{x}$ or $\mathbf{a}$, the gradient field is calculated as

$$\nabla_{\mathbf{x}}\mathcal{I}(t) = \frac{1}{2}\text{trace}(\mathbf{Y}^{-1}(t)\nabla_{\mathbf{x}}\dot{\mathbf{Y}}(t)). \tag{C.17}$$

This allows evaluation of the gradient field in terms of the current Fisher information and the partial derivatives of Equation C.15. Actions can be scheduled according to the direction and magnitude of the local gradient field. In order to maximize the information rate Equation C.16, the platform should head in the direction of the gradient vector of information rate with respect to the vehicle state $\{x, y\}$. The best action with zero look ahead is the direction of the gradient vector of information rate with respect to the vehicle state $\{x, y\}$

$$\psi^{\star}(t) = \arctan\left(\frac{\nabla_y \mathbf{I}(t)}{\nabla_x \mathbf{I}(t)}\right). \tag{C.18}$$

The full derivation, discussion, and examples can be found in [70].

Appendix D

Algorithms for Certainty Grids

D.1 Integrated Exploration Algorithm

A popular approach to choosing the exploration path when using Certainty Grids is to propose several potential destinations based on certain criteria, evaluate the expected utilities for each one, and pick the one with the highest expected value. The exploration action sequence adopted in this work is shown in Figure D.1. The algorithm is called Integrated Exploration, referring to the balanced evaluation of alternative motion actions from the point of view of information gain, localization quality, and motion cost.

The frontier method [242] is used to identify areas of interest which tend to lie on the edge of the explored and the unexplored regions. Figure D.2 shows an example of the frontier destinations for a partially built CG map. Other criteria for proposing destinations are certainly possible. For example, if the vehicle pose uncertainty is getting dangerously large, locations where the chances of relocalizing are high may be suggested.

The total expected value of driving to a particular destination and making observations there is calculated as a weighted sum of the utilities of information gain, action, and communication. The utility of action is further divided into the integral utility of navigation to the destination $\tilde{\rho}_{iAN}^{k+t,k}$ and the terminal utility of localization quality $\tilde{\rho}_{iAL}^{k+K,k}$

$$\tilde{V}_i^K(\pi_{iA}) \approx (1-\lambda)\tilde{\rho}_{iB}^{k+K,k} + \lambda \left(\sum_{t=0}^{K} \tilde{\rho}_{iAN}^{k+t,k} + \tilde{\rho}_{iAL}^{k+K,k} \right) + 0, \tag{D.1}$$

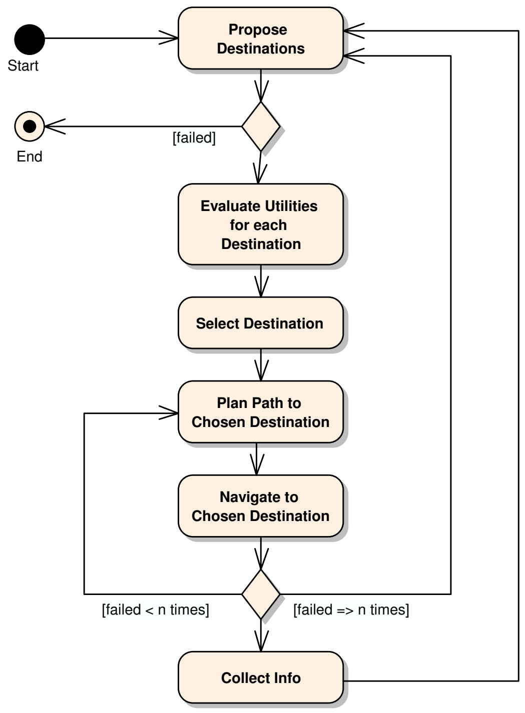


Figure D.1: Integrated Exploration algorithm.

where λ is a weighting coefficient. The result is approximate because the information gain along the path to the destination is ignored.

D.2 Utility of Information Gain

The main goal of an exploration strategy is to collect information about the world. The information utility is designed to favor destinations which offer higher information gain. For each CG cell c_i within the estimated sensed region W and each observation j, the information gain is computed as the mutual information of the belief and the observation. The total information gain is the sum over all cells in the sensed region W and all observations k

$$\tilde{\rho}_{iB}^{k+K,k} = \sum_{W} \sum_{t} \mathcal{I}^{b^k_{xi}}(x^k, z_i^{k+t,k}). \tag{D.2}$$

In practice, the expression for $\tilde{\rho}_{iB}^{k+K,k}$ can be simplified if the platform takes multiple observations at the destination point. It can be assumed that all information will be collected

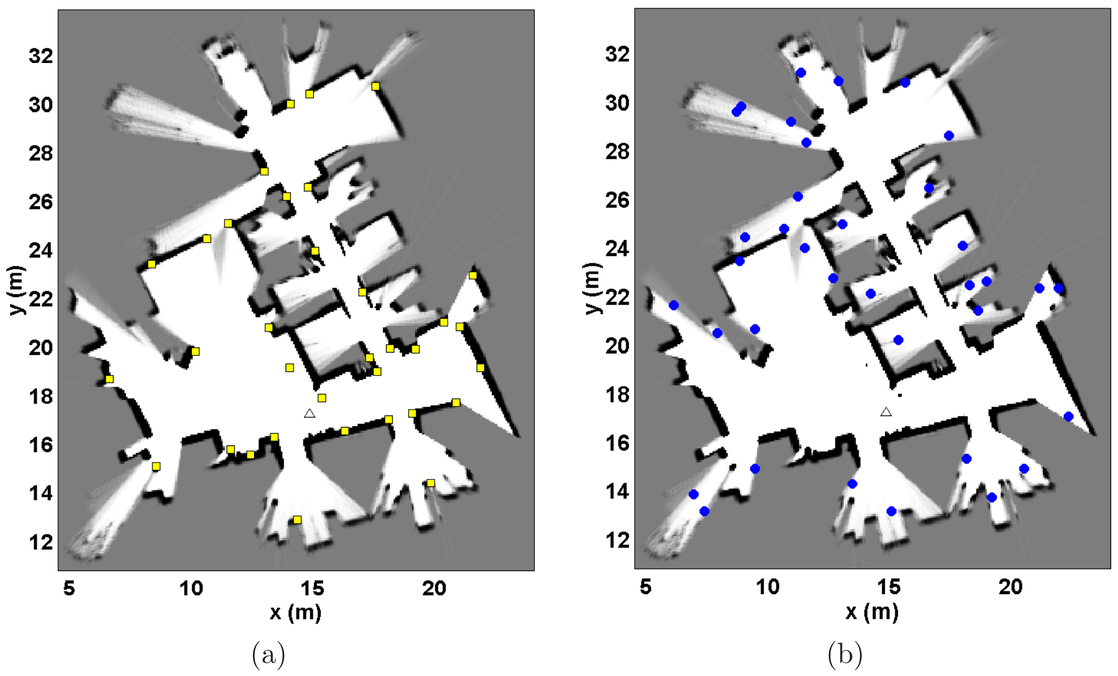


Figure D.2: Illustration of the planning process for a section of an experimentally obtained map: (a) the CG and the SLAM feature maps and (b) proposed destination frontiers. Also shown: the current vehicle position (△), SLAM landmarks (□) and potential destinations (○).

about the region visible from the destination point. The information utility is found by summing the entropy in all cells within the estimated sensed region

$$\tilde{\rho}_{iB}^{k+K,k} = \sum_{W} \mathcal{E}n(b_{xi}^{k}). \tag{D.3}$$

For a realistic estimate of the information gain from a destination point, the size and shape of the region covered by the sensor must be predicted. For the case of a range bearing sensor such as laser range finder, a ray tracing algorithm is employed. A screen shot in Figure D.3 illustrates the process.

Figure 5.16(b) shows the entropy distribution for a partially built occupancy map. Darker regions have higher entropy and, therefore, less information about them is available. The never-visited territory is completely unknown and has maximum entropy.

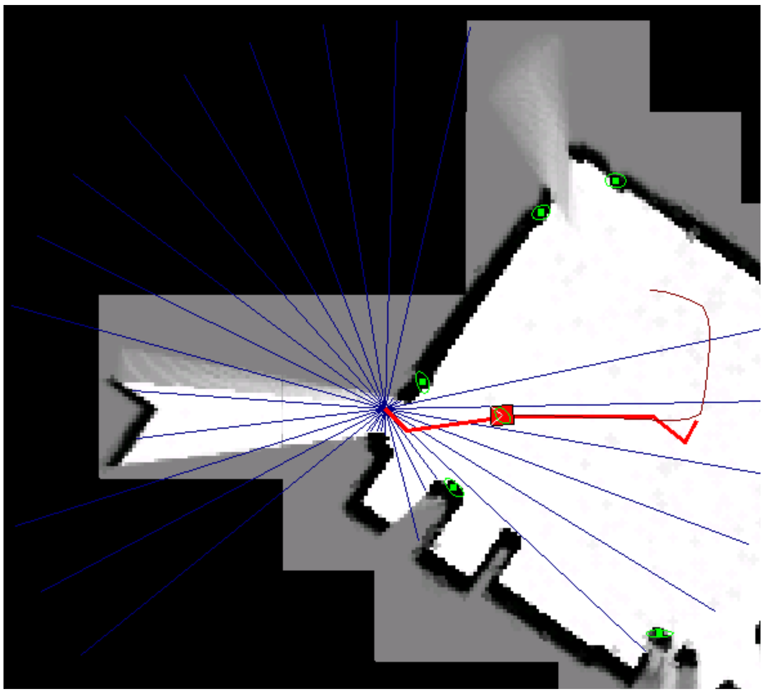


Figure D.3: Ray tracing used to estimated the shape of the sensed region. The total observed area is estimated by summing the area of sectors along each ray. The accuracy can be traded for processing time by varying the number of rays.

D.3 Utility of Navigation

Long travel time between destinations reduces the efficiency of exploration. The navigation utility is used to make shorter drives more appealing. The navigation component of total utility is

$$\tilde{\rho}_{iAN}^{k+K,k} = \sum_{t=0}^{K} \tilde{\rho}_{iAN}^{k+t,k} = \frac{1}{\sum_{t=0}^{K} t} = \frac{1}{T}, \tag{D.4}$$

where T is the total expected time it takes to drive to the destination. The hyperbolic penalty imposed by the $1/T$ term, however, makes the far away destination points prohibitively expensive. An alternative which has performed better in practice is a polynomial utility term in the form of $(T^* - T)^{\lambda_n}$, where T^* is the platform's maximum range and λ_n is a scaling parameter. Figure D.4 shows linear, quadratic, and cubic options compared to the hyperbolic one.

The expected time of travel T is found by computing the numerical *navigation function* [118]. Additional terms can be easily added to the navigation utility as long as they are

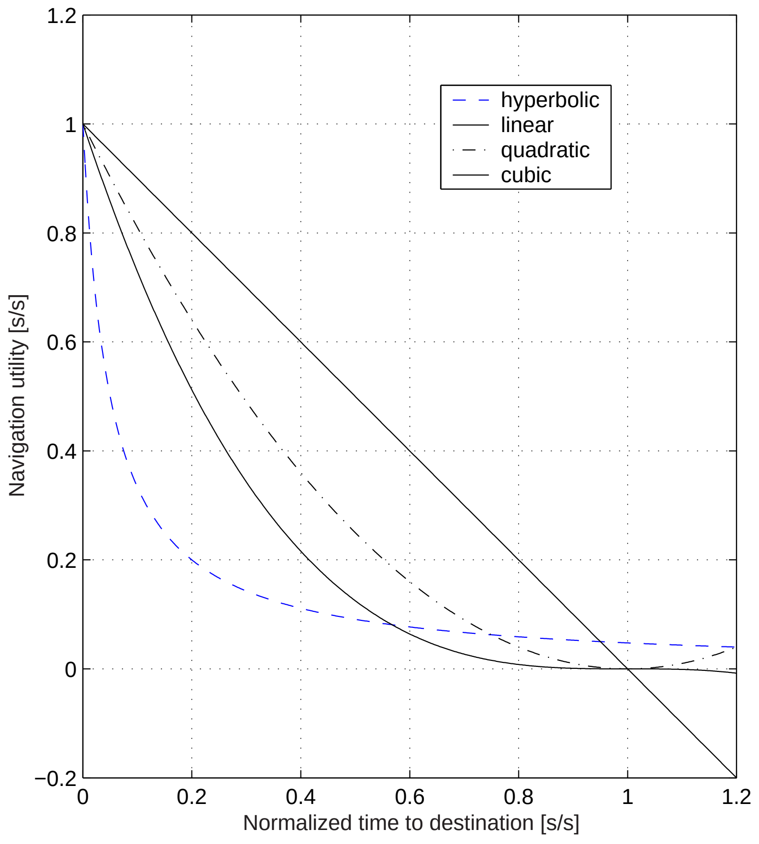


Figure D.4: Options for re-scaling the navigation utility.

expressed in units of time, such as penalty for large angle turns which could be slow on real life robots.

Figure 5.16(e) shows the navigation utility calculated for a partially built occupancy map. Darker areas are closer and therefore are more appealing. Notice that the algorithm produces navigation utility for all destinations at once from the current starting point.

D.4 Utility of Localizability

Collecting data about the previously unmapped territory is of little use if this information cannot be integrated properly into the map due to high vehicle localization error. The localization utility is used to distinguish between destinations with different localization quality. Conceptually, the quality of localization at any given point x_i is determined by the uncertainty in vehicle position achievable at that point. The uncertainty is described by the state covariance matrix $\mathbf{P}_{vv}(x_i, t)$ at the point x_i at time t.

To provide a uniform basis of comparison between different locations, the *localizability* metric is defined as the minimum vehicle covariance achievable by re-localizing a lost vehicle at a given location by observing only the features visible from that location. The number of observations is allowed to vary within $k \in [1\ \infty)$. The advantage of this definition is that the metric is clearly a function of both the map quality and the environment itself.

The details of calculating localizability depend on the specific localization algorithm. For the case of SLAM, the vehicle position uncertainty is represented by a covariance matrix $\mathbf{P}_{vv}$. In Appendix D.5, the upper bound on the vehicle covariance matrix achievable after re-localization given the current feature map is derived. To convert the re-localized vehicle covariance matrix $\mathbf{P}_{vv}^k$ to a scalar utility, the Shannon information measure is used as defined in Equation B.12

$$U_i^L = -\mathcal{E}n(\mathbf{P}_{vv}^k) = -\frac{1}{2}\log((2\pi e)^n|\mathbf{P}_{vv}^k|), \tag{D.5}$$

where n is the number of vehicle states and $\mathbf{P}_{vv}^k$ is the vehicle state covariance after k observations given by Equation D.11. Figure 5.16(c) shows the localizability metric for a partially built occupancy map. The number of observations was set to $k = 10$.

Calculation of the localizability metric is an expensive operation. It is only performed during the planning stage and only for a limited number of proposed destinations. The costliest part of the current implementation is ray-tracing to determine visibility of beacons. The possibility of calculating the metric for the entire map or part of the map is currently under investigation. This information could then be incorporated into the path planning operation. Such a map would conceptually be very similar to the one described in [193] where the map is computed *off-line* from an *a priori* map.

D.5 Localizability Metric for SLAM Algorithm

The platform is localized using the Simultaneous Localization and Mapping (SLAM) algorithm [44, 123, 240]. The method uses the Extended Kalman Filter (EKF) algorithm to optimally estimate the state vector $\mathbf{x}$, containing both the vehicle states $\mathbf{x}_v$ and the feature states $\mathbf{x}_m$, as well as the associated covariance matrix $\mathbf{P}$

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_v \\ \mathbf{x}_m \end{bmatrix}, \mathbf{P} = \begin{bmatrix} \mathbf{P}_{vv} & \mathbf{P}_{vm} \\ \mathbf{P}_{vm}^T & \mathbf{P}_{mm} \end{bmatrix}. \tag{D.6}$$

Details of the SLAM algorithm can be found in [44].

We now define the localizability metric for the case of SLAM feature-based localization. Assuming a partially known feature map and a partially known occupancy map, the following steps are taken:

1. Start with a completely unknown vehicle location (infinite covariance) and a partially known feature map $\mathbf{P}_{mm}$
2. Using the occupancy map, estimate which of the known features are observable from the location in question
3. Keep the vehicle stationary and estimate the vehicle covariance $\mathbf{P}_{vv}^k$ after k observations of visible features

In the derivation of $\mathbf{P}_{vv}(k)$ it is convenient to use the information form of the Kalman filter described in Appendix C.2. The initial information matrix for a "teleported" vehicle is

$$\mathbf{P}_{ox}^{-1} = \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{P}_{om}^{-1} \end{bmatrix}. \tag{D.7}$$

where $\mathbf{P}_{om}$ is the covariance matrix of the landmarks *observable* from the location in question. The unobservable features do not influence the vehicle covariance, greatly reducing the size of the problem. The zero sub-matrix in the upper-left corner reflects the lack of information about the initial vehicle location. The off-diagonal terms are also set to zero. The update equation is

$$\mathbf{P}^{-1}(k \mid k) = \mathbf{P}^{-1}(k \mid k-1) + \begin{bmatrix} \mathbf{H}_v^T \\ \mathbf{H}_m^T \end{bmatrix} \mathbf{R}^{-1} \begin{bmatrix} \mathbf{H}_v^T & \mathbf{H}_m^T \end{bmatrix}. \tag{D.8}$$

where $\mathbf{H}_v$ and $\mathbf{H}_m$ are the appropriately sized Jacobians of the generally non-linear observation function, and $\mathbf{R}$ is the variance of the sensor noise, assumed to be non-singular.

Because the localizability metric is defined for a stationary vehicle, the linearization of the observation function has to be performed only once and the information state vector does not have to be updated. For the same reason, the process noise is zero ($\mathbf{Q}(k) = \mathbf{0}$) and the prediction stage is simply

$$\mathbf{P}^{-1}(k \mid k-1) = \mathbf{P}^{-1}(k-1 \mid k-1). \tag{D.9}$$

Following [44], the Equations D.7 through D.9 are combined to find the information matrix after k observations

$$\mathbf{P}^{-1}(k \mid k) = \mathbf{P}_{ox}^{-1} + k \begin{bmatrix} \mathbf{H}_v^T\mathbf{R}^{-1}\mathbf{H}_v & \mathbf{H}_v^T\mathbf{R}^{-1}\mathbf{H}_m \\ \mathbf{H}_m^T\mathbf{R}^{-1}\mathbf{H}_v & \mathbf{H}_m^T\mathbf{R}^{-1}\mathbf{H}_m \end{bmatrix}. \tag{D.10}$$

Invoking the matrix inversion lemma twice, the vehicle covariance after k observations can be found. We are interested primarily in the sub-matrix corresponding to the vehicle states $\mathbf{P}_{vv}$

$$\mathbf{P}_{vv}^k = \left[\mathbf{H_v}^T\left[\tfrac{1}{k}\mathbf{R} + \mathbf{H}_m\mathbf{P}_{om}\mathbf{H}_m^T\right]^{-1}\mathbf{H}_v\right]^{-1}. \tag{D.11}$$

Note that in this form, the intermediate matrices are symmetric positive definite which allowing the use of computationally efficient matrix inversion algorithms. In the limit for $k \to \infty$, the vehicle covariance reaches steady state at

$$\mathbf{P}_{vv}^\infty = \left[\mathbf{H}_v^T\left[\mathbf{H}_m\mathbf{P}_{om}\mathbf{H}_m^T\right]^{-1}\mathbf{H}_v\right]^{-1}. \tag{D.12}$$

If the vehicle were to re-localize in practice at the given location, its true steady-state re-localized uncertainty would be not higher than the variance in Equation D.12. The upper bound property of $\mathbf{P}_{vv}^\infty$ follows from the result that the map covariances in the SLAM algorithm cannot increase [44].

As an illustration, consider a vehicle in 2-D space, equipped with a range and bearing sensor. The non-linear observation model is

$$\mathbf{z}k = \begin{bmatrix} z_r(k) \\ z_\theta(k) \end{bmatrix} = \begin{bmatrix} \sqrt{(x_v - x_i)^2 + (y_v - y_i)^2} \\ \arctan(\frac{y_v - y_i}{x_v - x_i}) - \psi_v \end{bmatrix} + \begin{bmatrix} \omega_r \\ \omega_\theta \end{bmatrix}. \tag{D.13}$$

where (x_v, y_v, ψ_v) and (x_i, y_i) are coordinates of the vehicle and a landmark at time k.

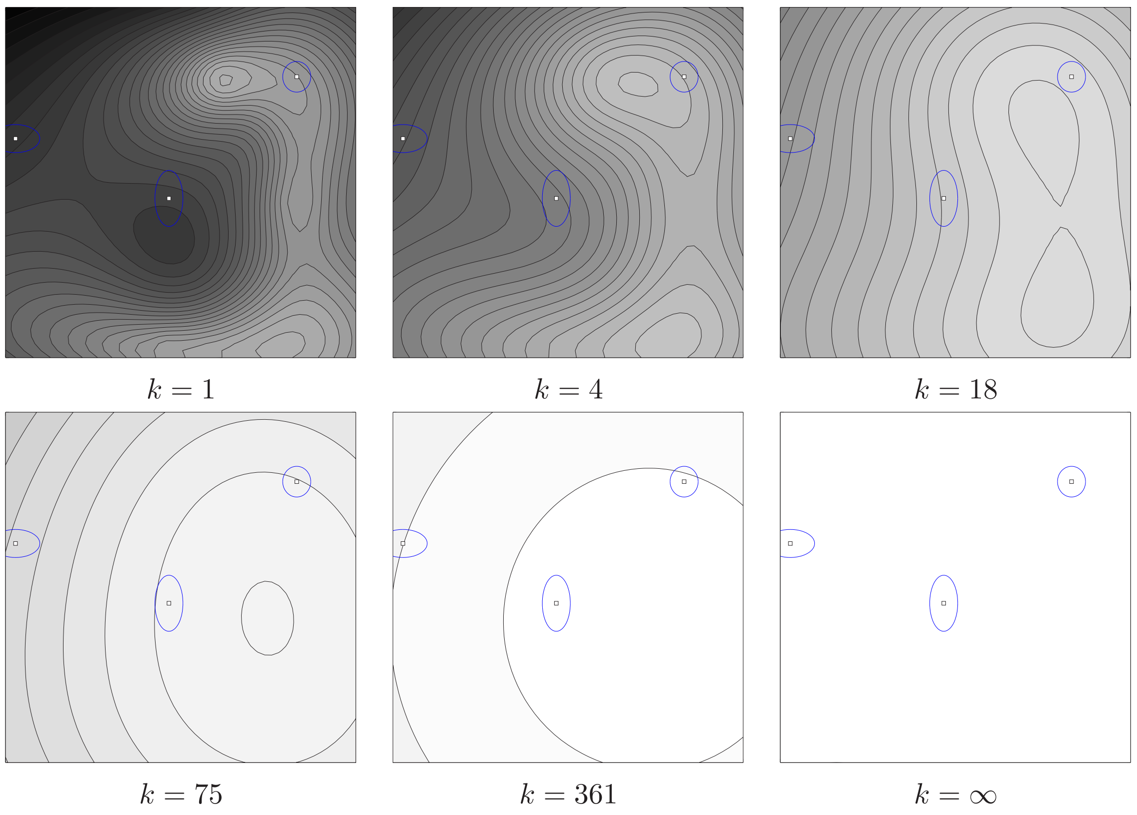


Figure D.5: An example of vehicle covariance after k observations. Darker color indicates higher uncertainty in vehicle localization. Initial feature uncertainties are drawn as ellipses

The localizability metric for an area around three uncorrelated landmarks is shown in Figure D.5. Results for different number of observations are shown. In this example there are no obstacles and the step of finding observable features is not necessary. In the limiting case ($k = \infty$), the localizability metric has the same value for all locations. In this case the localizability metric is simply a projection of the initial map uncertainty from the feature space into the vehicle space. For a finite number of observations, however, the vehicle covariance depends also on the relative configuration of the vehicle and the landmarks.

The dominant factors in the value of localizability are the number of landmarks visible from a given point and the uncertainty in the landmark locations. To illustrate the dependence between the number of visible features and localizability, the plot of localization utility (see Equation D.5) for different numbers of visible landmarks is shown in Figure D.6. The results are for all the cells of the map segment in Figure 5.16(c) $k = 10$. The strong dependence of

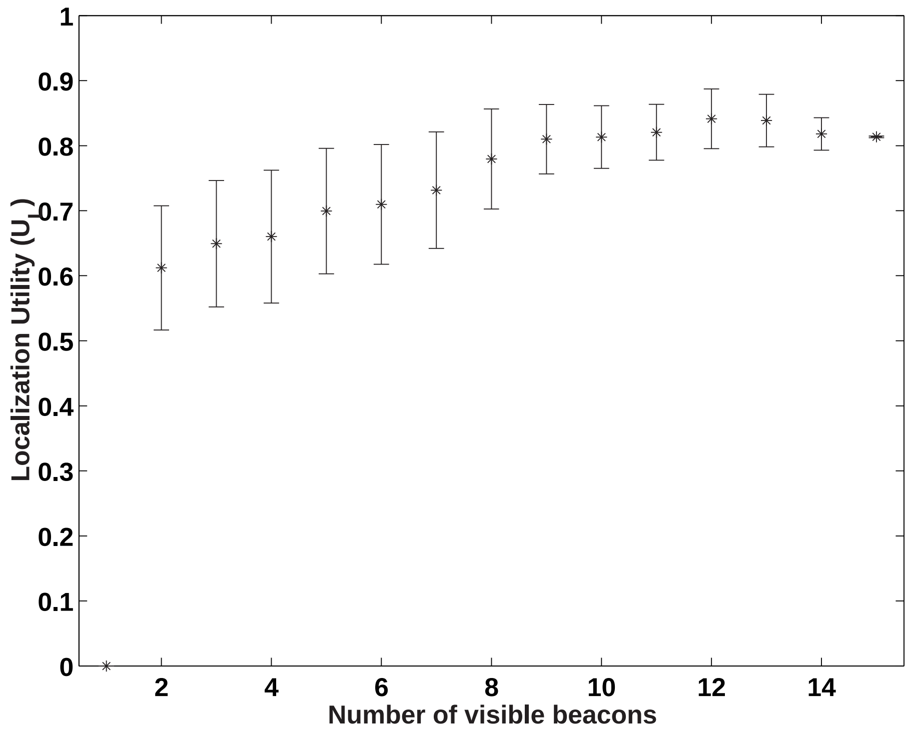


Figure D.6: Dependence of localizability on the number of visible beacons, the mean ($*$) and one standard deviation bounds are shown

localizability on the number of visible beacons is readily seen as well as the variation within each group.

Bibliography

[1] J. Agre and L. Clare. An integrated architecture for cooperative sensing networks. *Computer*, 33(5):106–108, 2000.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[4] R. Alami, L. Aguilar, H. Bullata, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, and F. Robert. A general framework for multi-robot cooperation and its implementation on a set of three Hilare robots. In *International Symposium on Experimental Robotics (ISER'95)*, pages 26–39, Stanford, CA, 1995.

[5] J.S. Albus. 4D/RCS: A reference model architecture for intelligent unmanned ground vehicles. In *SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, Orlando, FL, USA, 2002.

[6] M.G. Alford and P.K. Varshney. A layered architecture for multisensor data fusion systems. In *Asilomar Conference on Signals, Systems, and Computers*, volume 1, pages 416 –419, 1999.

[7] R. Alur, A. Das, J. Esposito, R. Fierro, Y. Hur, G. Grudic, V. Kumar, I. Lee, J.P. Ostrowski, G. Pappas, J. Southall, J. Spletzer, and C.J. Taylor. A framework and architecture for multirobot coordination. In *Seventh International Symposium on Experimental Robotics (ISER'00)*, Honolulu, Hawaii, USA, 2000.

[8] G.R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, 1991.

[9] Y. Bar-Shalom. *Multitarget-Multisensor Tracking: Advanced Applications.* Artech House, Norwood, 1990.

[10] Y. Bar-Shalom, X.R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation.* John Wiley & Sons, Inc., 2001.

[11] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[12] L. Bass. Software architecture design principles. In G.T. Heineman and W.T. Councill, editors, *Component-based software engineering: putting the pieces together.* Addison-Wesley, Boston, 2001.

[13] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice.* SEI series in software engineering. Addison-Wesley, Boston, 2nd edition, 2003.

[14] P. Bauer, M. Sichitiu, R. Istepanian, and K. Premaratne. The Mobile Patient: Wireless distributed sensor networks for patient monitoring and care. In *IEEE Conference on Information Technology Applications in Biomedicine*, pages 17–21, 2000.

[15] T. Berg and H. Durrant-Whyte. On distributed and decentralized estimation. In *American Control Conference*, pages 2273–4, 1992.

[16] J.O. Berger. *Statistical decision theory and Bayesian analysis.* Springer series in statistics. Springer-Verlag, New York, 2nd edition, 1985.

[17] D.P. Bertsekas. *Dynamic programming: deterministic and stochastic models.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[18] J.A. Besada, J. de Diego, G. de Miguel, J.R. Casar, and J. Garcia. Generic software architecture for development of data fusion systems. In *the Fifth International Conference on Information Fusion*, volume 1, pages 432–439, 2002.

[19] F. Bourgault. Experiments in coordinated decentralized search. In *9th International Symposium on Experimental Robotics 2004 (ISER'04)*, Singapore, 2004.

[20] F. Bourgault, A.A. Makarenko, S.B. Williams, B. Grocholsky, and H.F. Durrant-Whyte. Information based adaptive robotic exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'02)*, Lausanne, Switzerland, 2002.

[21] F. Bourgault, T. Furukawa, and H.F. Durrant-Whyte. Coordinated decentralized search for a lost target in a Bayesian world. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'03)*, pages 48–53, Las Vegas, NV, USA, 2003.

[22] A. Brooks, A. Makarenko, T. Kaupp, S. Williams, and H. Durrant-Whyte. Implementation of an indoor Active Sensor Network. In *9th International Symposium on Experimental Robotics 2004 (ISER'04)*, Singapore, 2004.

[23] A. Brooks, S. Williams, and A. Makarenko. Automatic online localization of nodes in an Active Sensor Network. In *IEEE International Conference on Robotics and Automation (ICRA'04)*, volume 5, pages 4821–4826, New Orleans, LA, USA, 2004.

[24] A.M. Brooks and S. Williams. Tracking people with networks of heterogeneous sensors. In *Australian Conference on Robotics and Automation (ACRA'03)*, Brisbane, Australia, 2003.

[25] R.A. Brooks. Intelligence without reason. In *12th International Joint Conference on Artificial Intelligence*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann.

[26] T. Burke and H.F. Durrant-Whyte. Kinematics for modular wheeled mobile robots. In *IEEE Conference Intelligent Robotics (IROS)*, volume 2, pages 1279 – 1286. 1993.

[27] H. Carvalho, W. Heinzelman, A. Murphy, and C. Coelho. A general data fusion architecture. In *the 6th International Conference on Information Fusion (Fusion'03)*, volume 2, pages 1465–1472, 2003.

[28] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. *SIGCOMM Computer Communication Review*, 31(2 suppliment):20–41, 2001.

[29] S. Challa and G.W. Pulford. Joint target tracking and classification using radar and ESM sensors. *IEEE Transactions on Aerospace and Electronic Systems*, 37(3): 1039–1055, 2001.

[30] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3):90–110, 2002.

[31] I. Clarke, T.W. Hong, and O. Sandberg. Protecting free expression online with Freenet. *IEEE Internet Computing*, (Jan-Feb):40–49, 2002.

[32] P. Clements. *Documenting software architectures: views and beyond.* Addison-Wesley, Boston, 2003.

[33] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies.* SEI series in software engineering. Addison-Wesley, Boston, 2002.

[34] W. Cohen. Adaptive mapping and navigation by teams of simple robots. *Robotics and Autonomous Systems*, 18:411–434, 1996.

[35] M. Collins-Cope. Component based development and advanced oo design. White paper, Ratio Group Ltd., 2001. URL `http://www.ratio.co.uk`.

[36] Solipsys Corporation. Tactical component network: Overview. White paper, Solipsys Corp., 2000.

[37] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. Wiley, New York, 1991.

[38] P. Dario, A. Menciassi, C. Stefanini, and D. Accoto. Miniaturization of biomedical micromachines. In *2nd Annual International IEEE-EMB Special Topic Conference on Microtechnologies in Medicine and Biology*, pages 291–296, 2002.

[39] A. D'Costa and A.M. Sayeed. Data versus decision fusion in wireless sensor networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'03)*, volume 4, pages 832–835, 2003.

[40] M. Dekhil and T.C. Henderson. Instrumented logical sensor systems – practice. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3103–3108, 1998.

[41] F. Derbel. Reliable wireless communication for fire detection systems in commercial and residential areas. In *IEEE Wireless Communications and Networking (WCNC'03)*, volume 1, pages 654–659, 2003.

[42] P. Diehl, M. Saptharishi, J. Hampshire, and P. Khosla. Collaborative surveillance using both fixed and mobile unattended ground sensor platforms. In *SPIE's 13th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls (AeroSense'99)*, Orlando, FL USA, 1999.

[43] M. Dietl, J.-S. Gutmann, and B. Nebel. Cooperative sensing in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, pages 1706–1713, Maui, Hawaii, USA, 2001.

[44] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csobra. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. on Robotics and Automation*, 17(3):229–241, 2001.

[45] M. Duarte and Y.-H. Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 64(7):826–838, 2004.

[46] H.F. Durrant-Whyte. *Integration, coordination, and control of multi-sensor robot systems.* The Kluwer international series in engineering and computer science; SECS 36. Robotics. Kluwer Academic Publishers, Boston, 1988.

[47] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Carnegie Mellon University, 1989.

[48] A. Elfes. Robot navigation: Integrating perception, environmental constraints and task execution within a probabilistic framework. In *Reasoning with Uncertainty in Robotics. International Workshop (RUR '95)*, pages 93–129, Amsterdam, Netherlands, 1995.

[49] W. Emmerich. *Engineering distributed objects.* John Wiley & Sons, Chichester, 2000.

[50] D. Estrin, R. Govindan, J.S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, Seattle, WA, USA, 1999.

[51] Q. Fang, F. Zhao, and L. Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 165–176, Annapolis, Maryland, USA, 2003.

[52] H.J.S. Feder, J.J. Leonard, and C.M. Smith. Adaptive mobile robot navigation and mapping. *International Journal of Robotics Research*, 18(7):650–668, 1999.

[53] J. Feng, F. Koushanfar, and M. Potkonjak. System-architectures for sensor networks issues, alternatives, and directions. In *IEEE International Conference on Computer Design*, pages 226–231, 2002.

[54] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence.* Addison-Wesley, Harlow, 1999.

[55] M. Fernandez. *Failure Detection and Isolation in Decentralised Multisensor Systems.* PhD thesis, The University of Oxford, 1993.

[56] T. Fong and C. Thorpe. Robot as partner: Vehicle teleoperation with collaborative control. In *NRL Workshop on Multi-Robot Systems*, pages 195–202, Washington, DC, USA, 2002.

[57] International Organization for Standardization. *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.* Number ISO/IEC 7498-1:1994. 1994.

[58] G.L. Foresti and L. Snidaro. A distributed sensor network for video surveillance of outdoor environments. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages 525–528, 2002.

[59] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing Magazine*, 2(3):24–33, 2003.

[60] M.P.J. Fromherz, T. Hogg, Y. Shang, and W.B. Jackson. Modular robot control and continuous constraint satisfaction. In *Workshop on Modelling and Solving Problems with Constraints (IJCAI '01)*, 2001.

[61] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass., 1994.

[62] E. Gat. On three-layer architectures. In D. Kortenkamp, R.P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. 1998.

[63] B.P. Gerkey, R.T. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *The 11th International Conference on Advanced Robotics (ICAR'03)*, pages 317–323, Coibra, Portugal, 2003.

[64] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. EmStar: a software environment for developing and deploying wireless sensor networks. In *USENIX Technical Conference*, pages 283–296, 2003.

[65] D.F. Gordon, W.M. Spears, O. Sokolsky, and I. Lee. Distributed spatial control, global monitoring and steering of mobile physical agents. In *the IEEE International Conference on Information, Intelligence, and Systems*, pages 681–688, 1999.

[66] R. Grabowski, L.E. Navarro-Serment, C.J.J. Paredis, and P.K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, 2000.

[67] S. Grime. *Communication in Decentralized Sensing Architectures.* PhD thesis, University of Oxford, 1992.

[68] S. Grime and H. Durrant-Whyte. Data fusion in decentralized sensor networks. *Control Engineering Practice*, 2(5):849–863, 1994.

[69] M.L. Griss. Software agents as next generation software components. In G.T. Heineman and W.T. Councill, editors, *Component-based software engineering: putting the pieces together*. Addison-Wesley, Boston, 2001.

[70] B. Grocholsky. *Information-Theoretic Control of Multiple Sensor Platforms*. PhD thesis, The University of Sydney, 2002.

[71] B. Grocholsky, A. Makarenko, T. Kaupp, and H. Durrant-Whyte. Scalable control of decentralised sensor platforms. In *The 2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, pages 96–112, Palo Alto, CA, USA, 2003.

[72] Object Management Group. *Trading Object Service Specification 1.0*. 2000. URL `http://www.omg.org`.

[73] Object Management Group. *CORBA Components Version 3.0*. 2002. URL `http://www.corba.org`.

[74] Object Management Group. *Naming Service Specification 1.2*. 2002. URL `http://www.omg.org`.

[75] Object Management Group. *Unified Modeling Language Specifications 2.0*. 2004. URL `http://www.omg.org`.

[76] G.T. Heineman and W.T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley, Boston, 2001.

[77] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, and M.A. Perillo. Middleware to support sensor network applications. *IEEE Network Magazine*, 18(1):6–14, 2004.

[78] T.C. Henderson and E. Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, 1(2):169–193, 1984.

[79] T.C. Henderson, M. Dekhil, S. Morris, Y. Chen, and W.B. Thompson. Smart sensor snow. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1377–1382, 1998.

[80] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *the 9th international conference on Architectural support for programming languages and operating systems*, pages 93–104, Cambridge, MA, US, 2000.

[81] P. Ho. *Organisation in Decentralised Sensing*. PhD thesis, Oxford University, 1995.

[82] Y.-C. Ho. Team decision theory and information structures. *Proceedings of the IEEE*, 68(6):644–654, 1980.

[83] Y.-C. Ho and K.a.-C. Chu. Team decsision theory and information structures in optimal control problems - part i. *IEEE Transactions on Automatic Control*, 17(1): 15–22, 1972.

[84] Y.-C. Ho and K.a.-C. Chu. Team decsision theory and information structures in optimal control problems - part ii. *IEEE Transactions on Automatic Control*, 17(1): 13–28, 1972.

[85] Y.-C. Ho, M.P. Kastner, and E. Wong. Teams, signaling and information theory. *IEEE Transactions on Automatic Control*, AC-23(2):305–311, 1978.

[86] G. Hoblos, M. Staroswiecki, and A. Aitouche. Optimal design of fault tolerant sensor networks. In *IEEE International Conference on Control Applications*, pages 467–472, Anchorage, AK, USA, 2000.

[87] J. Hoffman and P. Riggan. The design and flight performance of the multispectral, airborne FireMapper system. In *5th International Airborne Remote Sensing Conference*, San Francisco, CA, USA, 2001.

[88] C. Hofmeister, R. Nord, and D. Soni. *Applied software architecture*. Addison-Wesley, Reading, Mass.; Harlow, 2000.

[89] A. Hoover and B.D. Olsen. Sensor network perception for mobile robotics. In *the IEEE International Conference on Robotics and Automation (ICRA'00)*, volume 1, pages 342–347, San Francisco, CA, USA, 2000.

[90] B. Horling, R. Mailler, M. Sims, and V. Lesser. Using and maintaining organization in a large-scale distributed sensor network. In *the Workshop on Autonomy, Delegation, and Control (AAMAS'03)*, Melbourne, Australia, 2003.

[91] D.F. Hougen, M.D. Erickson, P.E. Rybski, S.A. Stoeter, M. Gini, and N.P. Papanikolopoulos. Autonomous mobile robots and distributed exploratory missions. In *Distributed Autonomous Robotic Systems (DARS'00)*, pages 221–230, Knoxville, TN, 2000.

[92] K. Houston and D. Norris. Software components and the UML. In G.T. Heineman and W.T. Councill, editors, *Component-based software engineering: putting the pieces together*, pages 243–262. Addison-Wesley, Boston, 2001.

[93] A. Howard, M. Mataric, and G. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *the 6th Int. Conf. on Distributed Autonomous Robotic Systems (DARS'02)*, pages 299–308, Fukuoka, Japan, 2002.

[94] A. Howard, L.E. Parker, and G.S. Sukhatme. The SDR experience: Experiments with a large-scale heterogeneous mobile robot team. In *International Symposium on Experimental Robotics (ISER'04)*, Singapore, 2004.

[95] T. Huntsberger, G. Rodriguez, and P. Schenker. Robotics challenges for robotic and human Mars exploration. In *ROBOTICS'00*, pages 84–90, Albuquerque, NM, USA, 2000.

[96] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A.J. Ganino, M. Garrett, S.S. Joshi, and P.S. Schenker. CAMPOUT: a control architecture for tightly coupled coordination of multirobot systems for planetary surface

exploration. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 33(5): 550–559, 2003.

[97] L. Iftode, C. Borcea, A. Kochut, C. Intanagonwiwat, and U. Kremer. Programming computers embedded in the physical world. In *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 78–85, 2003.

[98] Merriam-Webster Inc. *Merriam-Webster's collegiate dictionary.* Merriam-Webster, Springfield, Mass, 10th edition, 2001.

[99] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *the 6th annual international conference on Mobile computing and networking (MobiCom'00)*, pages 56–67, Boston, MA, USA, 2000.

[100] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11 (1):2–16, 2003.

[101] R. Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization.* Dover Publications, Mineola, N.Y., 1965.

[102] S.S. Iyengar, D.N. Jayasimha, and D. Nadig. A versatile architecture for the distributed sensor integration problem. *IEEE Transactions on Computers*, 43(2):175–185, 1994.

[103] L. Jia, R. Rajaraman, and C. Scheideler. On local algorithms for topology control and routing in ad hoc networks. In *the 15th annual ACM symposium on parallel algorithms and architectures*, pages 220–229, San Diego, CA, USA, 2003.

[104] H. Jonkers. Interface-centric architecture descriptions. In *Working IEEE/IFIP Conference on Software Architecture*, pages 113–124, 2001.

[105] S. Julier and J. Uhlmann. General decentralised data fusion with covariance intersection (CI). In D. Hall and J. Llinas, editors, *Handbook of Data Fusion.* CRC Press, 2001.

[106] J. Kahn, R. Katz, and K. Pister. Emerging challenges: Mobile networking for "smart dust". *Journal of Comm. Networks*, pages 188–196, 2000.

[107] J.N. Kapur. *Measures of Information and their Application.* John Wiley, 1994.

[108] T. Kaupp. *Scalable Human-Robot Interactions in Active Sensor Networks.* MS thesis, ACFR at University of Sydney, 2003.

[109] M. Kearns, M.L. Littman, and S. Singh. Graphical models for game theory. In *Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

[110] C. Keating, R. Rogers, R. Unal, D. Dryer, A. Sousa-Poza, R. Safford, W. Peterson, and G. Rabadi. System of systems engineering. *Engineering Management Journal*, 15(3):36–48, 2003.

[111] M.M. Kokar and J.A. Tomasik. Data vs. decision fusion in the category theory framework. In *4th International Conference on Information Fusion (Fusion'01)*, 2001.

[112] F. Kon. *Automatic Configuration of Component-Based Distributed Systems.* PhD thesis, University of Illinois, 2000.

[113] K. Konolige, C. Ortiz, R. Vincent, A. Agno, Michael, Eriksen, B. Limketkai, M. Lewis, L. Briesemeister, Enrique, Ruspini, D. Fox, J. Ko, B. Stewart, and L. Guibas. CentiBOTS: Large scale robot teams. In *International Workshop on Multi-Robot Systems*, pages 193–204, Washington, DC, USA, 2003.

[114] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, and J. Ko. Centibots: Very large scale distributed robotic teams. In *International Symposium on Experimental Robotics*, Singapore, 2004.

[115] B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in wireless sensor networks. In *DEBS'02*, 2002.

[116] P. Kruchten. Modeling component systems with the Unified Modeling Language. In *ICSE'98 Workshop*, 1998.

[117] P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, 1995.

[118] J.-C. Latombe. *Robot Motion Planning.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, 1991.

[119] J. Leal. *Stochastic Environment Representation.* PhD thesis, The University of Sydney, 2002.

[120] O. Lebeltel, P. Bessiere, J. Diard, and E. Mazer. Bayesian robots programming. *Autonomous Robots*, 16(1):49–79, 2004.

[121] J.-H. Lee and H. Hashimoto. Intelligent space - its concept and contents. *Advanced Robotics Journal*, 16(4):265–280, 2002.

[122] J.-H. Lee and H. Hashimoto. Controlling mobile robots in distributed intelligent sensor network. *IEEE Transactions on Industrial Electronics*, 50(5):890–902, 2003.

[123] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localisation for an autonomous robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS '91)*, pages 1442–1447, Osaka, Japan, 1991.

[124] V. Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1:88–111, 1998.

[125] V. Lesser and D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, 1983.

[126] V. Lesser, K. Decker, N. Carver, A. Garvey, D. Neiman, N. Prasad, and T. Wagner. Evolution of the gpgp domain-independent coordination framework. Computer Science Technical Report TR-98-05, University of Massachusetts at Amherst, Jan 1998.

[127] V. Lesser, M. Atighetchi, B. Benyo, B. Horling, A. Raja, R. Vincent, T. Wagner, X. Ping, and S.Z. Zhang. The intelligent home testbed. In *Autonomy Control Software Workshop*, Seattle, WA, USA, 1999.

[128] J. Liu, J. Reich, and F. Zhao. Collaborative in-network processing for target tracking. In *Journal on Applied Signal Processing*, 2002.

[129] J. Liu, D. Petrovic, and Feng Zhao. Multi-step information-directed sensor querying in distributed sensor networks. In *IEEE International Conference on Acoustics, Speech, and Signal Processing(ICASSP '03)*, volume 5, pages 145–148, 2003.

[130] J. Llinas, C. Bowman, G. Rogova, A. Steinberg, E. Waltz, and F. White. Revisions and extensions to the JDL data fusion model II. In *International Conference on Information Fusion*, pages 1218–1230, Stockholm, Sweden, 2004.

[131] A. Longshaw. Choosing between COM+, EJB, and CCM. In G.T. Heineman and W.T. Councill, editors, *Component-based software engineering: putting the pieces together.* Addison-Wesley, Boston, 2001.

[132] M. Lopez-Sanchez, R.L.d. Mantaras, and C. Sierra. Incremental map generation by low cost robots based on possibility / necessity grids. In *Uncertainty in Artificial Intelligence*, pages 351–357, Providence, RI, USA, 1997.

[133] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *8th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 55–66, 2002.

[134] R.H. Ludwig. The role of technology in modern warfare. In *Software Technology Conference*, 1992.

[135] R.C. Luo and M.G. Kay. Multisensor integration and fusion in intelligent systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):901–931, 1989.

[136] D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[137] A. Makarenko, T. Kaupp, and H. Durrant-Whyte. Scalable human-robot interactions in active sensor networks. *IEEE Pervasive Computing Magazine*, 2(4):63–71, 2003.

[138] A. Makarenko, T. Kaupp, B. Grocholsky, and H. Durrant-Whyte. Human-robot interactions in active sensor networks. In *Computational Intelligence in Robotics and Automation (CIRA'03)*, volume 1, pages 247–252, Kobe, Japan, 2003.

[139] R. A. Malan and D. Bredemeyer. Functional requirements and use cases. Technical report, Bredemeyer Consulting, June 1999. URL `http://www.bredemeyer.com`.

[140] R.A. Malan and D. Bredemeyer. Defining non-functional requirements. Technical report, Bredemeyer Consulting, August 2001. URL `http://www.bredemeyer.com`.

[141] R.A. Malan and D. Bredemeyer. Software architecture: Central concerns, key decisions. Technical report, Bredemeyer Consulting, 2003. URL `http://www.bredemeyer.com`.

[142] R.A. Malan and D. Bredemeyer. The visual architecting process. Technical report, Bredemeyer Consulting, 2003. URL `http://www.bredemeyer.com`.

[143] J. Manobianco. Global environmental MEMS sensors (GEMS): A revolutionary observing system for the 21st century. NASA institute for advanced concepts, phase I final report, 2002.

[144] J. Manyika. *An Information Theoretic Approach to Data Fusion and Sensor Management.* PhD thesis, The University of Oxford, 1993.

[145] J. Manyika and H.F. Durrant-Whyte. *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach.* Ellis Horwood Series in Electrical and Electronic Engineering. Ellis Horwood, 1994.

[146] J. Marschak. Elements for a theory of teams. *Management Science*, 1(2):127–137, 1955.

[147] J. Marschak and R. Radner. *Economic theory of teams.* Yale University Press, New Haven, CT, 1972.

[148] M.C. Martin and H. Moravec. Robot evidence grids. Technical report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, March, 1996 1996.

[149] R.C. Martin. Granularity. *C++ Report*, 1996.

[150] P.S. Maybeck. *Stochastic models, estimation and control.* Mathematics in science and engineering; v. 141. Academic Press, New York, 1979.

[151] W.M. Merrill, K. Sohrabi, L. Girod, J. Elson, F. Newberg, and W. Kaiser. Open standard development platforms for distributed sensor networks. In *SPIE, Unattended Ground Sensor Technologies and Applications IV (AeroSense'02)*, volume 4743, pages 327–337, Orlando, FL, USA, 2002.

[152] S. Meyer and A. Rakotonirainy. A survey of research on context-aware homes. In *the Australasian information security workshop conference on ACSW frontiers*, pages 159–168, Adelaide, Australia, 2003.

[153] G.A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97, 1956.

[154] N. Minar and M. Hedlund. A network of peers: Peer-to-peer models through the history of the internet. In A. Oram, editor, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly, 2001.

[155] N. Minar, M. Gray, O. Poop, R. Krikorian, and P. Maes. Hive: distributed agents for networking things. *IEEE Concurrency*, 8(2):24–33, 2000.

[156] S. Moorehead, R. Simmons, and W.R.L. Whittaker. Autonomous exploration using multiple sources of information. In *IEEE International Conference on Robotics and Automation (ICRA'01)*, volume 3, pages 3098–3103, 2001.

[157] J. Mostow. Terminator 3: Rise of the machines, 2003. URL `http://www.terminator3.com/`.

[158] A.L. Murphy, G.P. Picco, and G.-C. Roman. LIME: A middleware for physical and logical mobility. In *21st International Conference on Distributed Computing*, pages 524–533, 2001.

[159] D. Musicki, R. Evans, and S. Stankovic. Integrated probabilistic data association. *IEEE Transactions on Automatic Control*, 39(6):1237 –1241, 1994.

[160] A. Mutambara. *Decentralised Estimation and Control with Applications to a Modular Robot*. PhD thesis, University of Oxford, 1995.

[161] A.G.O. Mutambara. *Decentralized estimation and control for multisensor systems*. CRC Press, Boca Raton, 1998.

[162] L.E. Navarro-Serment, R. Grabowski, C.J.J. Paredis, and P. Khosla. Modularity in small distributed robots. In *SPIE conference on Sensor Fusion and Decentralized Control in Robotic Systems II*, pages 297–306, 1999.

[163] E Nettleton. *Decentralised Architectures for Tracking and Navigation with Multiple Flight Vehicles*. PhD thesis, University of Sydney, 2003.

[164] E. Nettleton, H. Durrant-Whyte, and S. Sukkarieh. A robust architecture for decentralised data fusion. In *The 11th International Conference on Advanced Robotics (ICAR'03)*, pages 1425–1430, Coimbra, Portugal, 2003.

[165] D. Nicholson, C. Lloyd, S. Julier, and J. Uhlmann. Scalable distributed data fusion. In *the Fifth International Conference on Information Fusion*, volume 1, pages 630–635, Sunnyvale, CA, USA, 2002.

[166] S. Nolfi and D. Floreano. *Evolutionary robotics: the biology, intelligence, and technology of self-organizing machines*. Intelligent robots and autonomous agents. MIT Press, Cambridge, Mass., 2000.

[167] Institute of Electrical and Electronics Engineers. IEEE recommended practice for architectural description of software-intensive systems, 2000.

[168] J.M. Ooi, S.M. Verbout, J.T. Ludwig, and G.W. Wornell. A separation theorem for periodic sharing information patterns in decentralized control. *IEEE Transactions on Automatic Control*, 42(11), 1997.

[169] A. Oreback and H.I. Christensen. Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14(1):33–49, 2003.

[170] L.E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[171] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[172] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1988.

[173] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[174] Gnutella Project, 1999. URL `http://www.gnutella.com`.

[175] Orca-Robotics Project, 2004. URL `http://orca-robotics.sourceforge.net`.

[176] Player/Stage Project, 2001. URL `http://playerstage.sourceforge.net`.

[177] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley series in probability and mathematical statistics. Applied probability and statistics section. Wiley, New York, 1994.

[178] D. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI Research*, 16:389–423, 2002.

[179] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, 2002.

[180] B. Rao. *Data Fusion Methods in Decentralised Sensing Systems*. PhD thesis, The University of Oxford, 1991.

[181] B. Rao, H. Durrant-White, and H. Hu. A fully decentralized multi-camera, multi-sensor data fusion system. *International Journal of Robotics Research*, 12(1):20–44, 1993.

[182] B.S.Y. Rao and H. Durrant-Whyte. A decentralized bayesian algorithm for identification of tracked targets. *IEEE Transactions of Systems Man and Cybernetics*, 23(6): 1683–1698, 1993.

[183] B.S.Y. Rao and H.F. Durrant-Whyte. A decentralized Bayesian algorithm for identification. In *IEEE Conference on Decision and Control*, volume 2, pages 827–828, 1990.

[184] B.S.Y. Rao, J.M. Manyika, and H.F. Durrant-Whyte. Decentralized algorithms and architecture for tracking and identification. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS'91)*, volume 2, pages 1095–1100, 1991.

[185] E. Rechtin. *Systems Architecting: Creating and Building Complex Systems*. Prentice-Hall, 1991.

[186] W.D. Rencken and H.F. Durrant-Whyte. A quantitative model for adaptive task allocation in human-computer interfaces. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4):1072–1090, 1993.

[187] M. Ridley, E. Nettleton, S. Sukkarieh, and H. Durrant-Whyte. Tracking in decentralised air-ground sensing networks. In *Proceedings of the Fifth International Conference on Information Fusion*, volume 1, pages 616–623. Sunnyvale, 2002.

[188] M. Ridley, B. Upcroft, L.L. Ong, S. Kumar, and S. Sukkarieh. Decentralised data fusion with parzen density estimates. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Melbourne, Australia, 2004.

[189] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing Magazine*, 1(4):74–83, 2002.

[190] M. Rosencrantz, G. Gordon, and S. Thrun. Locating moving entities in indoor environments with teams of mobile robots. In *The 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 233–240, 2003.

[191] M. Rosencrantz, G. Gordon, and S. Thrun. Decentralized sensor fusion with distributed particle filters. In *the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 493–500, 2003.

[192] J.S. Rosenschein and G. Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers*. Artificial intelligence. MIT Press, Cambridge, Mass., 1994.

[193] N. Roy, W. Burgard, D. Fox, and S. Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *the IEEE International Conference on Robotics and Automation (ICRA '99)*, volume 1, pages 35–40, Detroit, MI, USA, 1999.

[194] P.E. Rybski, I. Burt, T. Dahlin, M. Gini, D.F. Hougen, D.G. Krantz, F. Nageotte, N. Papanikolopoulos, and S.A. Stoeter. System architecture for versatile autonomous and teleoperated control of multiple miniature robots. In *IEEE International Conference on Robotics and Automation (ICRA'01)*, volume 3, pages 2917–2922, 2001.

[195] N.R. Sandell, P. Varaiya, M. Athans, and M.G. Safonov. Survey of decentralized control methods for large scale systems. *IEEE Transactions on Automatic Control*, AC-23:108–128, 1978.

[196] M. Saptharishi, K. Bhat, C. Diehl, C. Oliver, M. Savvides, A. Soto, J. Dolan, and P. Khosla. Recent advances in distributed collaborative surveillance. In *SPIE Proceedings on Unattended Ground Sensor Technologies and Applications (AeroSense'00)*, pages 199–208, 2000.

[197] D.C. Schmidt. Why software reuse has failed and how to make it work for you. *C++ Report*, 11(1), 1999.

[198] J.C. Scholtz. Human-robot interaction: Creating synergistic cyber forces. In *NRL Workshop on Multi-Robot Systems*, pages 177–184, Washington, DC, USA, 2002.

[199] D. Schulz, D. Fox, and J. Hightower. People tracking with anonymous and ID sensors using Rao-Blackwellised particle filters. In *International Joint Conf. Artificial Intelligence (IJCAI'03)*, pages 921–926. Morgan Kauffman, 2003.

[200] J.R. Searle. *Speech acts: an essay in the philosophy of language.* Cambridge University Press, London, 1969.

[201] R.C. Shah and J.M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC'02)*, volume 1, pages 350–355, 2002.

[202] R.C. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: modeling a three-tier architecture for sparse sensor networks. In *the First IEEE Sensor Network Protocols and Applications*, pages 30–41, 2003.

[203] M. Shaw and P. Clements. A field guide to boxology: preliminary classification of architectural styles for software systems. In *International Computer Software and Applications Conference (COMPSAC'97)*, pages 6–13, 1997.

[204] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8(4):52–59, 2001.

[205] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *the 7th annual international conference on Mobile computing and networking (MobiCom'01)*, pages 272–287, Rome, Italy, 2001.

[206] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[207] G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme. Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks. In *IEEE International Conference on Robotics and Automation (ICRA'02)*, volume 2, pages 1143–1148, 2002.

[208] F. Silva, J. Heidemann, and R. Govindan. Network routing API 9.1. Technical report, Information Sciences Institute, USC, June 2003.

[209] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In *7th International Symposium on Experimental Robotics (ISER'00)*, 2000.

[210] R. Simmons, T. Smith, M.B. Dias, D. Goldberg, D. Hershberger, A.T. Stentz, and R.M. Zlot. A layered architecture for coordination of mobile robots. In *the 2002 NRL Workshop on Multi-Robot Systems*, pages 103–112, Washington, DC, USA, 2002.

[211] R.G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *AAAI National Conference on Artificial Intelligence*, pages 852–858, Austin, TX, USA, 2000.

[212] B. Sinopoli, C. Sharp, L. Schenato, S. Shaffert, and S.S. Sastry. Distributed control application within sensor networks. *Proceedings of the IEEE*, 91(8):1235–1246, 2003.

[213] W.D. Smart and L.P. Kaelbling. Effective reinforcement learning for mobile robots. In *the IEEE International Conference on Robotics and Automation (ICRA'02)*, pages 3404 –3410, 2002.

[214] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000.

[215] D. Soni, R.L. Nord, and C. Hofmeister. Software architecture in industrial applications. In *the 17th international conference on Software engineering*, pages 196–207, 1995.

[216] A. Soto, M. Saptharishi, A. Trebi, J. Dolan, and P. Khosla. Cyber-ATVs: Dynamic and distributed reconnaissance and surveillance using all terrain UGVs. In *the International Conference on Field and Service Robotics (FSR'99)*, pages 329–334, 1999.

[217] C. Stachniss and W. Burgard. Mapping and exploration with mobile robots using coverage maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'03)*, pages 467–472, Las Vegas, NV, USA, 2003.

[218] L.D. Stone, T.L. Corwin, and J.B. Hofmann. Technical documentation of Nodestar. Formal report AT-95-003, Naval Research Laboratory, Dec 1995.

[219] L.D. Stone, C.A. Barlow, and T.L. Corwin. *Bayesian multiple target tracking.* Artech House, London, 1999.

[220] L. Subramanian and R.H. Katz. An architecture for building self-configurable systems. In *First Annual Workshop on Mobile and Ad Hoc Networking and Computing (MobiHOC'00)*, pages 63–73, 2000.

[221] S. Sukkarieh, E. Nettleton, J.-H. Kim, M. Ridley, A. Goktogan, and H. Durrant-Whyte. The ANSER project: Data fusion across multiple uninhabited air vehicles. *International Journal of Robotics Research*, 22(7/8):505–540, 2003.

[222] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction.* Adaptive computation and machine learning. MIT Press, Cambridge, Mass., 1998.

[223] Sparx Systems. Enterprise Architect user guide, 2004. URL `http://www.sparxsystems.com`.

[224] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming.* ACM Press; Addison-Wesley, New York London, 2nd edition, 2002.

[225] Crossbow Technology. Mica 2 wireless measurement system. Product Specification 6020-0042-04 Rev B, 2004.

[226] A. Tews, M.J. Mataric, and G.S. Sukhatme. Scaling high level interactions between humans and robots. In *AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*, pages 196–202, Stanford, CA, USA, 2003.

[227] S. Tilak, N.B. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless microsensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2):28–36, 2002.

[228] V. Tsiatsis, S.A. Zimbeck, and M.B. Srivastava. Architecture strategies for energy-efficient packet forwarding in wireless sensor networks. In *International Symposium on Low Power Electronics and Design*, pages 92–95, 2001.

[229] R.M. Turner and E.H. Turner. Organization and reorganization of autonomous oceanographic sampling networks. In *IEEE International Conference on Robotics and Automation (ICRA'98)*, volume 3, pages 2060 –2067, 1998.

[230] I. Ulrich and J. Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA '98)*, pages 1572–1577, 1998.

[231] C. Urmson, J. Anhalt, M. Clark, T. Galatali, J.P. Gonzalez, J. Gowdy, A. Gutierrez, S. Harbaugh, M. Johnson-Roberson, H. Kato, P.L. Koon, K. Peterson, B.K. Smith, S. Spiker, E. Tryzelaar, and W.L. Whittaker. High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004. Technical Report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University, 2004.

[232] S. Utete. *Network Management in Decentralised Sensing Systems.* PhD thesis, University of Oxford, 1994.

[233] S. Utete and H.F. Durrant-Whyte. Routing for reliability in decentralized sensing networks. In *American Control Conference*, volume 2, pages 2268–2272, 1994.

[234] S. Utete and H.F. Durrant-Whyte. Reliability in decentralised data fusion networks. In *IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI'94)*, pages 215–221, 1994.

[235] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, 18(4):493–497, 2002.

[236] J. van Dam. *Environment Modelling for Mobile Robots: Neural Learning for Sensor Fusion.* PhD thesis, University of Amsterdam, 1998.

[237] M. Velonaki, D. Rye, S. Schedding, and S. Williams. Fish–Bird: An interactive art installation utilising autonomous objects. In *Ars Electronica Time Shift – The World in 25 years*, Linz, Austria, 2004.

[238] M. Weiser. The computer for the twenty-first century. *Scientific American*, 265(3): 94–101, 1991.

[239] B.B. Werger. Ayllu: Distributed port-arbitrated behavior-based control. In *Distributed Autonomous Robot Systems (DARS'00)*, Knoxville, TN, USA, 2000.

[240] S.B. Williams. *Efficient Solutions to Autonomous Mapping and Navigation Problems.* PhD thesis, The University of Sydney, 2001.

[241] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin. Topology control protocols to conserve energy in wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, submitted, 2003.

[242] B. Yamauchi. A frontier based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'97)*, pages 146–151, Monterey, CA, USA, 1997.

[243] B. Yamauchi. Frontier-based exploration using multiple robots. In *Second International Conference on Autonomous Agents (AGENTS'98)*, pages 47–53, Minneapolis, MN, USA, 1998.

[244] Y. Yao and J.E. Gehrke. Query processing in sensor networks. In *1st Biennial Conference on Innovative Data Systems Research (CIDR'03)*, Asilomar, CA, USA, 2003.

[245] M. Yokoo, O. Etzioni, T. Ishida, and N. Jennings, editors. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems.* Springer Series on Agent Technology. Springer Verlag, 1 edition, 2001.

[246] J.S. Zhang, C.K. Mohan, P. Varshney, C. Isik, K. Mehrotra, S. Wang, Z. Gao, and R. Rajagopalan. Intelligent control of building environmental systems for optimal evacuation planning. In *International Conference on Indoor Air Quality Problems and Engineering Solutions*, Research Triangle Park, NC, 2003.

[247] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.

[248] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach. *Proceedings of the IEEE*, 91(8):1199–1209, 2003.